ANN-based Performance Estimation of Embedded Software for RISC-V Processors

Weiyan Zhang[1] Mehran Goli[1,2] Alireza Mahzoon[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, mahzoon, drechsler}@uni-bremen.de

***Abstract*—The demand for optimized and efficient embedded software is increasing in many applications such as the *Internet of Things* (IoT) or other *Cyber-Physical Systems* (CPS). Hence, early performance analysis of embedded software is essential to perform *Design Space Exploration* (DSE), ensure efficiency, and meet time-to-market constraints. 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.**

In this paper, we present a novel *Artificial Neural Network* (ANN)-based approach that allows a fast and accurate performance estimation of embedded software for RISC-V processors in the early design phases. This can significantly reduce the burden on designers to perform DSE. The proposed approach takes advantage of the dynamic analysis technique and analytical models and does not require any microarchitecture-related parameters such as cache misses, cache hits, and memory-level parallelism. We compare our proposed microarchitecture-independent approach with state-of-the-art in terms of speed and accuracy. Our experiments on various benchmarks demonstrate that the proposed approach achieves a speed-up of $4.41\times$ compared to a RISC-V *Virtual Prototype* (VP) at the *Electronic System Level* (ESL), while the estimation results have only a *Mean Absolute Percentage Error* (MAPE) of 2%.

***Index Terms*—RISC-V, performance estimation, embedded software, artificial neural network**

I. Introduction

As an open and free *Instruction Set Architecture* (ISA), RISC-V gained enormous momentum in recent years. It provides designers with a variety of advantages including modularity, simplicity, and extensibility. The benefits of RISC-V have attracted both academia and industry to use it for a variety of embedded system applications such as the *Internet of Things* (IoT) or other *Cyber-Physical Systems* (CPS). Moreover, as technology evolves, applications of embedded systems are ubiquitous and indispensable in our daily life now, such as mobile phones, smartwatches, printers, etc. The demand for them is increasing rapidly, and the trend is growing in momentum every year. At the same time, the specifications of embedded systems become more and more complex, and there are plenty of design choices ranging from the choice of components, algorithms, operating systems, etc. A systematic exploration process is required to find the optimal solution and make the design decisions easier. Due to the complexity of the exploration process, exploring embedded software usually requires substantial time and effort. In order to meet time-to-market constraints, fast *Design Space Exploration* (DSE) is necessary to speed up the entire process. Performance estimation at the early stage of the design process is a promising solution that can significantly improve the DSE of embedded software by avoiding the iterative process of slow software simulation.

Performance estimation techniques can be divided into three main categories: static analysis, dynamic analysis, and hybrid analysis. For static analysis, the data is collected without simulating or executing the software. Therefore, it is unable to identify the runtime features (e.g., cache misses or hits) that could cause problems at runtime. For dynamic analysis, the code is executed, and the software behavior at runtime can be tracked. Dynamic analysis is preferred when functional behavior needs to be checked. Hybrid analysis is a combination of both static and dynamic analysis techniques to form a unified statistics-gathering module. Using this approach, it is possible to selectively implement only the most effective features that each analysis technique has to offer. This characteristic allows it to provide a level of utility unmatched by any single-purpose technique. In addition to being classified as static, dynamic, and hybrid analysis techniques, the level of abstraction characterizes different performance estimation methods.

Over the last five years, several RISC-V simulators have been proposed at different levels of abstraction. *Register Transfer Level* (RTL) implementations such as RSD [1] can provide accurate cycle counts but are slow to simulate, while functional simulators such as RV8 [2] can simulate very fast but cannot provide the designer with cycle counts or timing information. Recently, as part of the RISC-V ecosystem, *Virtual Prototypes* (VPs), such as the open-source RISC-V VP [3], have been introduced to facilitate early software development and testing, as well as other system-level purposes. At the *Electronic System Level* (ESL) [4], VP is considered the first implementation of real hardware that can perform cycle-accurate simulation. Essentially, a VP is an executable software model of the entire hardware platform, usually implemented in SystemC (a C++-based library), which avoids the need for physical hardware prototypes required for hardware/software co-design [5]. It

This work was supported in part by the German Federal Ministry of Education and Research (BMBF) within the project ECXL under contract no. 01IW22002, and by the University of Bremen's graduate school SyDe, funded by the German Excellence Initiative.

979-8-3503-9851-9/22/$31.00 ©2022 IEEE

also helps to reduce time-to-market.

While the development of the aforementioned high-level abstraction implementation makes early embedded software development possible, there is a huge difference in simulation speed between functional simulators and cycle-accurate simulators (even at the ESL using VP) for *Application-Specific Instruction Set Processors* (ASIP). This places a heavy burden on embedded software DSE, which motivated our attention. *Machine Learning* (ML) algorithms have gained immense popularity in the current technological age due to their versatility and broad applicability. Some studies show their application in hardware/software co-design [6], [7]. With the rapid development of ML algorithms, especially *Artificial Neural Networks* (ANNs), analytical performance models are considered excellent tools for exploring the design spaces. Analytical models, on the other hand, can map the relationship between the number of executed instructions and performance to produce results very quickly. This makes the performance estimation independent of the microarchitecture implementation of a given RISC-V processor.

This paper aims at **providing a fast and accurate approach for performance estimation of embedded software for RISC-V processors using ANN** and microarchitecture independent *Predictive Models* (PMs). It enables the evaluation of the full design space with only one analysis step. We utilize the dynamic analysis technique (through a fast functional simulator) to track the runtime information of embedded software and take advantage of analytical models to quickly calculate the performance of embedded software. The extracted runtime trace is used as an input to the PMs to estimate the performance of embedded software. This means that the only requirement for the proposed approach is an accurate number of each executed instruction type. Experimental results demonstrate that the proposed approach can achieve a speed-up of $4.41\times$ compared to a RISC-V VP at the ESL (which has an order of magnitude faster simulation speed than the RTL counterpart). This allows designers to explore the design space of embedded software more efficiently.

The structure of the paper is as follows. Section II describes the related work. The proposed estimation method is presented in Section III, and experimental results are detailed and analyzed in Section IV. Conclusions are drawn in Section V.

II. Related Work and Motivation

The performance of embedded software is affected by its structure (e.g., data accesses/transfers, etc) and the components of the target system (e.g., the ISA and CPU speed). Techniques involving performance estimation at an early stage can be static [8], [9], dynamic [10], [11], or hybrid [12], [13]. An effective approach to make the execution time reasonable without sacrificing too much accuracy is to model both embedded software and the target system at an sufficiently high level of abstraction. However, as the level of abstraction rises (from assembly to high level languages like C), it becomes more and more difficult to take the structure of the software into account since the assembly code is increasingly removed from the abstract representation. Recently, most of the results reported in the literature are from the object code level and use dynamic analysis techniques, and most available tools mimic the behavior of the target system on which the object code can be executed without real hardware.

For RISC-V, there are a number of *Instruction Set Simulators* (ISS) at different levels of abstraction available ranging from slow RTL models with cycle accuracy such as RSD [1], cycle-accurate simulators such as SystemC-based VPs at the ESL [14] [3] to very fast functional simulators at the algorithmic level without cycle and timing information such as RV8 [2]. There is no ideal simulator that can bridge the gap between simulation speed and accuracy. To relieve the heavy burden on the DSE of embedded software, analytical performance models are considered.

Early analytical models [15], [16] were initially aimed at estimating and studying microarchitectural variations on pipeline and instruction-level parallelism. Recent approaches apply ML techniques to analytical models for cross-platform performance estimation. These approaches are based on dynamic or static analysis and utilize the extracted information to generate a set of PMs for the estimation of the number of instructions executed or clock cycles elapsed. These models can be categorized as linear or non-linear. In [17], linear regression-based PM is used to estimate the performance of the software and evaluate the method against an ARM v5 implementation. The method introduced in [18] employs neural networks to estimate the performance of the software and compares the results against a PowerPC 750 cycle-accurate model. In [19], PMs are built for the ARM926EJ-S and the LEON3 processor based on linear regression on different numerical features such as the number of times an operation appears in the code or is executed. The method presented in [20] proposes a generic framework for rapid performance estimation of embedded software on soft-core processors to select the best configuration with respect to performance. However, a common drawback for all aforementioned methods is that they need to characterize the processor ISA and consider the microarchitectural features of the underlying processor. It means that further manual effort by designers is required to extract such information from a given RISC-V implementation. This has two disadvantages: 1) they are very time-consuming, and 2) they are microarchitecture dependent i.e., for a new RISC-V implementation, the entire method must be adopted.

Recently, [21] proposes an approach based on linear regression to estimate performance of embedded software on a RISC-V processor. The method contains two PMs that rely on the total number of executed instructions or the instruction formats. However, no detailed runtime information is considered, nor is the non-linear behavior analyzed. This limits the method to only support a simple implementation of RISC-V ISA where only linear dependency is considered.

Due to the huge space for exploration and the current trend to use RISC-V processors, there is a need to develop performance estimation methods that are accurate and fast without microarchitecture analysis, taking into account the performance impact of advanced features such as caching, branch prediction, and pipelines. However, there is no PM to

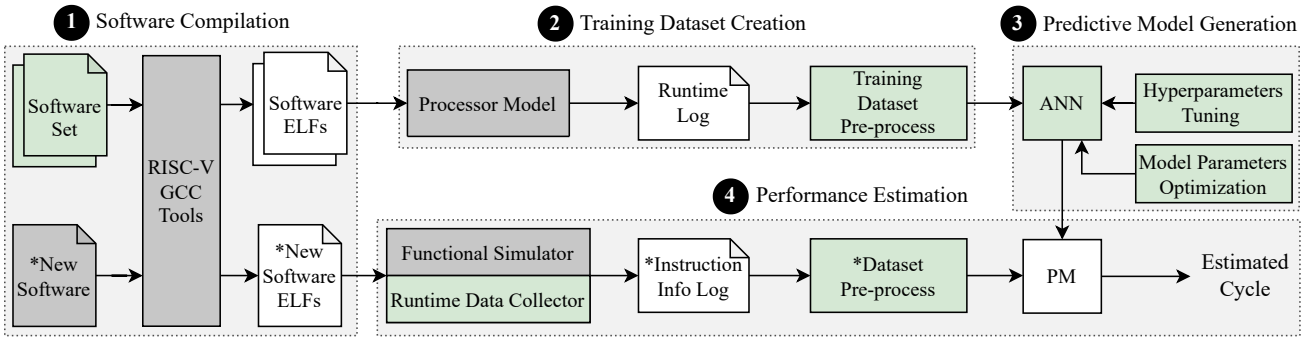


Fig. 1: The proposed performance estimation methodology overview.

describe these non-linear behaviors of RISC-V processors. We aim to fill this research gap by proposing a novel analytical performance estimation model that does not rely on any static analysis but instead takes advantage of a fast functional simulator to extract an accurate trace of the execution of a given piece of software. *Artificial Neural Networks* (ANNs) have been chosen as they can generalize the behavior of the features even when the process to be modeled is highly non-linear. Since the inputs of the generated PMs are only related to the runtime trace of the executed software on the functional simulator, the proposed approach does not require any microarchitecture information (i.e., a black-box analysis) and can be performed on any microarchitecture implementation of the RISC-V ISA.

III. ML-based Performance Estimation Methodology

The proposed methodology is depicted in Fig. 1, consisting of four main phases which are:

1) generating a set of software with different features for the training phase and compiling them with the RISC-V GNU Toolchain to create RISC-V execution files,
2) creating training dataset using the collected data from VP as the input of the learning phase,
3) generating performance PM using ML algorithm, and
4) validating the generated PM by estimating the performance of a set of standard new embedded software.

In the following, we describe each phase of the proposed approach in detail.

A. Training Programs Generation and Compilation

In order to obtain an effective ML-based performance PM that can generalize to the new data and avoid the over-fitting problems, it is necessary to collect a comprehensive set of training data. As shown in the first phase of Fig. 1, a set of software is provided that can cover ISA instructions. They are simple to complex programs with different inputs, parameters and features. The programs are functionally completely different to cover various instructions of the RISC-V ISA and are totally independent and different from the new software used in the validation phase (the fourth phase in Fig. 1). This difference makes our PM more general and increases the possibility of predicting the performance of new software from different domains. The new software used in the validation phase is standard benchmarks from [22]–[24] used in different application domains such as *Digital Signal Processing* (DSP), cryptography, and image recognition.

The execution of a given program on the RISC-V processor requires the *Executable and Linkable Format* (ELF) of the program to be generated during the compliation process. In general, there are two compilation options which are direct compilation and cross-compilation. Direct compilation refers to compiling the software on RISC-V using the RISC-V toolchain to produce a RISC-V ELF, while cross-compilation refers to compiling the software on other platforms using the RISC-V toolchain to produce a RISC-V ELF. Considering the compile-time and multitasking capabilities, we take advantage of the cross-compilation from our host computer to a RISC-V processor where the RISC-V GNU compiler toolchain [25] is used to produce the RISC-V ELFs. In this paper, three RISC-V ISAs are considered: base integer instruction for the 32-bit architecture (RV32I), its M extension (RV32IM), and its C extension (RV32IC).

B. Training Dataset Creation

In order to estimate the performance (i.e., accurate number of executed cycles) of embedded software in the early stage of the design process, it is important to focus on the information which is available at this stage. The available information that can be obtained without real hardware execution is 1) a detailed runtime history of executed instructions, and 2) an accurate number of executed cycles. VPs at the ESL can provide designers with total abstract models of hardware platforms. To do this, SystemC-based RISC-V VP [3]—a processor model—is considered, which is open-source, can be configured to mimic the behavior of the RISC-V HIFIVE1 board, and is shown to be cycle-accurate [14]. The ELFs of the software set are executed on the RISC-V VP. However, a detailed count of every instruction is not provided by the RISC-V VP. To extract this information, we implement a runtime data collector module and integrate it into the RISC-V VP. By executing the ELF file of each software on the RISC-V VP, the instruction counts and the number of executed cycles are extracted and stored in the *Runtime Log* file.

TABLE I: Extracted Parameters from the *Runtime Log* File

Parameters	Description
$x_{i,1} \cdots x_{i,39}$	Instruction counts for `LUI, AUIPC, JAL, JALR, BEQ, BNE, BLT, BGE, BLTU, BGEU, LB, LH, LW, LBU, LHU, SB, SH, SW, ADDI, SLTI, XORI, ORI, ANDI, SLLI, SRLI, SRAI, ADD, SLL, SLT, SLTU, XOR, SRL, SRA, OR, AND, SUB, SLTIU, FENCE, ECALL`
$x_{i,40} \cdots x_{i,47}$	Instruction counts for `MUL, MULH, MULHSU, MULHU, DIV, DIVU, REM, REMU`
y_i	Total cycles

After extracting the runtime data, the training dataset can be generated, and it consists of two main elements which are inputs (the *predictors*) and the corresponding output (the *response*). A pair of predictors and responses is considered one observation. According to the performance equation, the number of executed cycles is related to the instruction counts. The training dataset T consists of a set of observations and is defined as follows:

$$T = \{t_i | t_i = \{y_i, (x_{i,1}, \cdots x_{i,M})\}; 1 \leq i \leq N\} \quad (1)$$

where each observation t is an $(M+1)$-dimensional vector where M is the number of instructions of the corresponding ISA. The size of T equals the number of training programs N.

In order to create clean code and increase code performance in the next phase, the data needs to be pre-processed (e.g., re-shaping, vectorization) to form the training dataset before being used to create the predictive model. Therefore, the two elements of training dataset T are vectorized as an $N \times M$ matrix $\mathbf{X}$ and an $N \times 1$ column vector $\mathbf{y}$, represented as follows:

$$\mathbf{X} = \begin{bmatrix} x_{1,1} & \cdots & x_{1,M} \\ \vdots & \ddots & \vdots \\ x_{N,1} & \cdots & x_{N,M} \end{bmatrix}; \mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix} \quad (2)$$

Table I illustrates the details of the parameters extracted from the *Runtime Log* for the i_{th} program. RV32I contains 40 unique instructions, but the EBREAK instruction is used to return control to a debugging environment. Thus, it is excluded in this paper and the RV32I instructions are reduced to 39 in total (i.e., $M = 39$). In addition to the base integer instructions, RV32IM contains eight standard integer multiplication and division instructions. During decoding, the compressed instructions in RV32IC are expanded to 32-bit base ISA, so it contains the same number of instructions as RV32I.

C. Predictive Model Generation

Due to their simplicity and adaptability to the non-linear behavior of software performance estimation, feedforward and backpropagation ANNs are used in the ML process to create performance predictive models in this phase. Our networks are composed of an input layer, one or several hidden layers, and an output layer. Each layer may have a different number of neurons and different activation functions. Fig. 2 shows schematically the structure of the ANN. The ANN with 1 hidden layer (or $\leq$ 2 hidden layers) is defined as *Shallow Neural Network* (SNN) while the network with multiple hidden layers is defined as *Deep Neural Network* (DNN). The matrix $\mathbf{X}$ obtained in the previous phase is fed into the input layer. The instruction counts in the matrix are sent forward through different connections from one neuron to another in the network. Each neuron holds a number bias and has an activation function, and each connection holds a weight. Once they reach the output layer, the estimated cycle is obtained. This procedure is called feedforward. Backpropagation is for calculating the gradients efficiently. The process always starts from the output layer and propagates backward, updating weights and biases for each layer to obtain the desired output in the output layer.

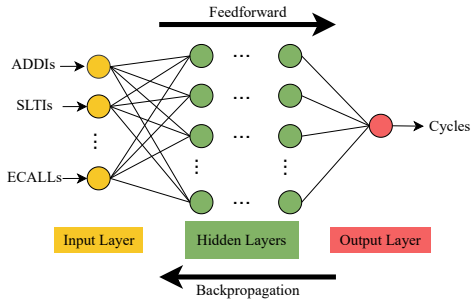


Fig. 2: ANN-based performance estimation model.

The ANN-based model architecture is determined by a set of hyperparameters. In contrast with normal parameters of an ML algorithm, hyperparameters are parameters that must be set before the algorithm is executed, as opposed to normal parameters of an algorithm that are not fixed before execution but optimized during training. To find the optimal hyperparameters, the hyperparameter tuning technique is used to explore a range of possibilities and choose the ideal model architecture. In our experiments, five hyperparameters are tuned: learning rate, number of hidden layers, number of neurons in each layer, activation function in each layer, and the number of epochs. After hyperparameter tuning, the best model with less error is obtained.

Hyperparameters are not model parameters. Model parameters such as weights and bias are learned during the training phase when a loss function is optimized using backpropagation to calculate the gradients. In the *Optimization* module (Fig.1, phase 3), *Mean Squared Error* (MSE), defined in (3), where the squared L2 norm is computed as the sum of squared errors, is used as a cost function, and the *Adaptive Moment Estimation* (Adam) optimizer is used to minimize the MSE. The parameter $\hat{\boldsymbol{y}}$ is the predicted output of ANN. The PM is generated when Adam converges to the optimal solution. For each RISC-V ISA, a different PM is generated.

$$MSE = \frac{1}{N} \|\boldsymbol{y} - \hat{\boldsymbol{y}}\|_2^2 \quad (3)$$

D. Utilization of Predictive Model

In this phase, the PM is tested by a set of standard and new embedded software. The new embedded software is compiled

to generate the ELF file for the RISC-V processor as described in Section III-A. Since the generated ELF file is compact and is not a human-readable file, we take advantage of an open-source functional simulator RV8 [2] to execute the ELF file, which can provide a detailed trace of the executed instructions. By implementing a runtime data collector module in RV8, instruction information including instruction counts is stored after execution. The extracted instruction counts of K pieces of embedded software are then pre-processed into a matrix $\mathbf{X}_{test}$ as shown in (4) and used as inputs of the PM to estimate the total number of cycles required by the embedded software.

$$\mathbf{X}_{test} = \begin{bmatrix} x_{1,1} & \cdots & x_{1,M} \\ \vdots & \ddots & \vdots \\ x_{K,1} & \cdots & x_{K,M} \end{bmatrix} \tag{4}$$

To assess the quality of the generated PMs, we tested a new set of embedded software on the PMs. Predictions post-process can be applied to the estimated cycles. The new software ELF is obtained in phase 1 of Fig. 1. The *Runtime Log* containing the real cycle of the new embedded software is generated by executing the ELF file on RISC-V VP. In order to check the performance of our PMs, the *Absolute Percentage Error* (APE) for new embedded software is calculated and *Mean Absolute Percentage Error* (MAPE) for the overall new embedded software set, defined in (5), is applied. Values for percent errors and MAPE are between zero to one, where a value of one stands for an error of 100%.

$$MAPE = \frac{100\%}{K} \left\| \frac{\mathbf{y} - \hat{\mathbf{y}}}{\mathbf{y}} \right\|_1 \tag{5}$$

IV. Experimental Results

The proposed approach was evaluated against a set of standard benchmarks provided by Embench [24], TACLeBench [23] and RV8-bench [22]. These benchmarks cover different domains, such as signal processing, cryptography, and sorting algorithms, and are freely available and designed specifically for embedded systems. For each benchmark, the simulation behavior is extracted by the *Runtime Data Collector* module (Fig. 1) and stored in the *Runtime Log* file. The extracted information is then pre-processed to generate a test dataset, which will be used to validate the PMs. The *Runtime Data Collector* module was written in C. The programs for the pre-processing datasets and for creating the PMs were written in Python. Three ISAs were considered, i.e., RV32I, RV32IM, and RV32IC. For each ISA, one PM based on the instruction counts was generated. We evaluated the simulation speed and accuracy of the generated PMs by comparing them with the SystemC-based RISC-V VP model [14]. The experimental evaluation is described in two parts. Section IV-A illustrates the structure of PMs. Section IV-B gives the experimental results for each ISA and briefly discusses the obtained results to evaluate the quality of the proposed method. The ANN for each PM was constructed with the open-source library Keras [26]. We conducted our experiments on a Linux system running at 1.4 GHz with 38 GB of RAM and an AMD Ryzen 7 PRO 4750U processor.

A. Structure of Predictive Models

Several experiments were conducted to determine more robust hyperparameters for ANNs. For different PMs, some hyperparameters were changed. We provided a wide hyperparameter search space. Random search [27] is used as a tuner. Each hyperparameter is randomly chosen from a given search space. MSE is used as an optimizer parameter. Table II summarized the parameters and the search space used in our tuning approach. The search space along with the selected values are also listed. The first column lists the tuned hyperparameters. The column search space defines the domain through which the tuner searches. The next three columns show the best values of PMs for three ISAs where PM1, PM2, and PM3 correspond to RV32I, RV32IM, and RV32IC, respectively.

In the training phase, we took advantage of 167 programs as training software to create the training dataset and generate the PM1. The total number of executed instructions for the training software set ranges from 4.2×10^3 to 7.2×10^9. To generate the PM2, 133 programs were used as training software sets. The total number of executed instructions is between 2.1×10^5 to 3.0×10^8. For PM3, the total number of executed instructions of 125 training programs ranges from 1.5×10^6 to 3.7×10^7. For each PM, the ANN has a fixed number of neurons in the input and output layers, and its detailed structure is obtained by minimizing the MSE during the training phase. Take PM1 as an example, the input layer has 39 neurons and 1 neuron in the output layer. For the given training dataset, the best model calculated by the tuner is a DNN with 3 hidden layers, each with 208, 384, and 16 neurons. To validate the quality of the generated PMs, the experimental results of testing each PM against a set of benchmarks are shown in the following subsection.

B. Performance of RISC-V Predictive Models

The new software used in the validation phase is completely different from the software used in the training phase to construct the training dataset. To validate each PM, 10 benchmarks were used. Fig. 3 and Table III demonstrate the overall embedded software performance estimation accuracy of applying the 10 standard benchmarks to our proposed performance estimation approach on RV32I, RV32IM, RV32IC and to [21]. Moreover, they show the speed-up comparison between our approach and RISC-V VP. Fig. 3 illustrates the results for RV32I and RV32IM. In the case of RV32I and RV32IM, we compare the accuracy of the proposed approach to the method presented in [21]. The quality of each PM is measured using the MAPE (i.e., Average) metric introduce in (5). The speed-up comparison is based on each execution time. The execution time of the proposed approach consists of two main parts which are 1) the required time for the RV8 simulator to generate the runtime trace of a given piece of software and 2) the time used by PM to estimate the cycle count of the software. Due to the fact that PM requires significantly less time than RV8 simulation, the time spent on RV8 can be seen as the total execution time to estimate the number of cycles the software requires. The results for RV32IC are presented in Table III. The first column lists the name and

TABLE II: Hyperparameter Tuning Results of ANN based on Random Search

Parameters	Search space	PM1	PM2	PM3
Learning rate	0.002 to 0.06	0.012	0.03	0.026
# hidden layers	1 to 9	3	6	9
# neurons in each hidden layer	16 to 512	208/384/16	256/224/464/ 112/288/96	304/16/16/16/ 16/16/16/16/16
Activation in each hidden and output layer	sigmoid/softsign/tanh/ selu/elu/exponential/ LeakyReLU/relu	softsign/softplus/sigmoid/ softplus	sigmoid/sigmoid/selu/ softsign/exponential/ sigmoid/LeakyReLU	relu/sigmoid/sigmoid/sigmoid/ sigmoid/sigmoid/sigmoid/ sigmoid/sigmoid/elu
Epochs	1 to 350	24	12	46

source of the benchmark. The following two columns show the total number of executed instructions (*#instr-exec.*) and *Lines of Code* (LoC) of each benchmark, respectively. In the *VP* column, the cycle count (*#Cycle*) and execution time (*Time*) reported from RISC-V VP are presented. Column *Ours* consists of four subcolumns showing the results obtained from the PM. The first two subcolumns indicate the execution time of the proposed approach and the obtained speed-up compared to the RISC-V VP. The following two subcolumns represent the estimated number of cycles reported from PM and the percent errors between estimated cycles and actual cycles reported from RISC-V VP, respectively

For RV32I, the MAPE in estimating the performance of the entire software set with our approach is about 2.1%, while the worse-case estimation error is less than 6.1%. In comparison to [21], for most benchmarks, our approach achieves better accuracy. For the overall benchmarks, our approach shows better accuracy in performance estimation compared to [21] with the MAPE 3.7% and achieves an average speed-up of $3.7\times$ over RISC-V VP. In [21], the linear regression formulas based on the total number of instructions and the instruction formats for RV32I and RV32IM are given. We took advantage of more detailed runtime information (i.e., the number of each instruction), and ANNs which provide a better estimation model for non-linear behavior. Fig. 3b demonstrates that our proposed approach is also effective for RV32IM, and it can estimate the number of cycles up to $3.9\times$ faster on average than RISC-V VP while a MAPE of 2.00% is obtained. In comparison to [21] where the worst-case error value of 26.5% and MAPE 5.0% are reported, the proposed approach has a worst-case error of 4.00% and MAPE 2.00%. This means that the linear regression-based methods may have certain limitations, especially if no linear dependency is available. Due to the MAPE metric, our proposed approach also outperforms [21] for the case of RV32IM. Since [21] takes advantage of the ML technique for performance estimation, its execution time is almost similar to ours.

As shown in Table III, the speed-up achieved by our approach over the RISC-V VP is up to $4.41\times$ for RV32IC while the MAPE is 2.64%. Since [21] does not support RV32IC, no comparison is performed for this case.

The experimental results demonstrate that the proposed approach based on ANN provides designers with a fast performance estimation solution and promising results even with a small training dataset. The fast simulation time in phase 4 as shown in Fig. 1 is due to two main reasons. First, we utilized a fast functional simulator to obtain the runtime trace of embedded software including accurate information about the number of instructions and their types, compared to the extra effort required by the VP to execute the software to obtain the accurate number of cycles. Second, we took advantage of ML-based PMs that can quickly estimate the performance of the embedded software. The MAPE values in each table show a small gap between the cycle counts obtained from our PMs and VP, indicating the effectiveness of our proposed approach. In comparison to other simple algorithms like linear regression (which use only input and output nodes for estimation), neural networks obtain better results in predictive analytics due to hidden layers. Furthermore, neural networks have a large number of free parameters (the weights and biases between interconnected neurons) which enables them to fit highly complex data (when trained correctly) that other models cannot.

V. Conclusions

In this paper, we presented a novel ANN-based approach for estimating the performance of a given piece of embedded software on a RISC-V processor at the early stage of the design process. We illustrated how the performance of a given piece of embedded software on RISC-V processor can be estimated by taking advantage of only the runtime trace of the software through a fast functional simulator. In our experiments, a cycle-accurate RISC-V VP with advanced features such as cache, pipelines, and branch prediction was used to evaluate the speed-up and estimation accuracy on a set of benchmarks. A MAPE of 2.00% and speed-up up to $4.41\times$ have been obtained. With the use of ANN, we showed that even small datasets can produce promising results. Our proposed method facilitates the early performance estimation of embedded systems for DSE and is independent of microarchitecture implementation.

References

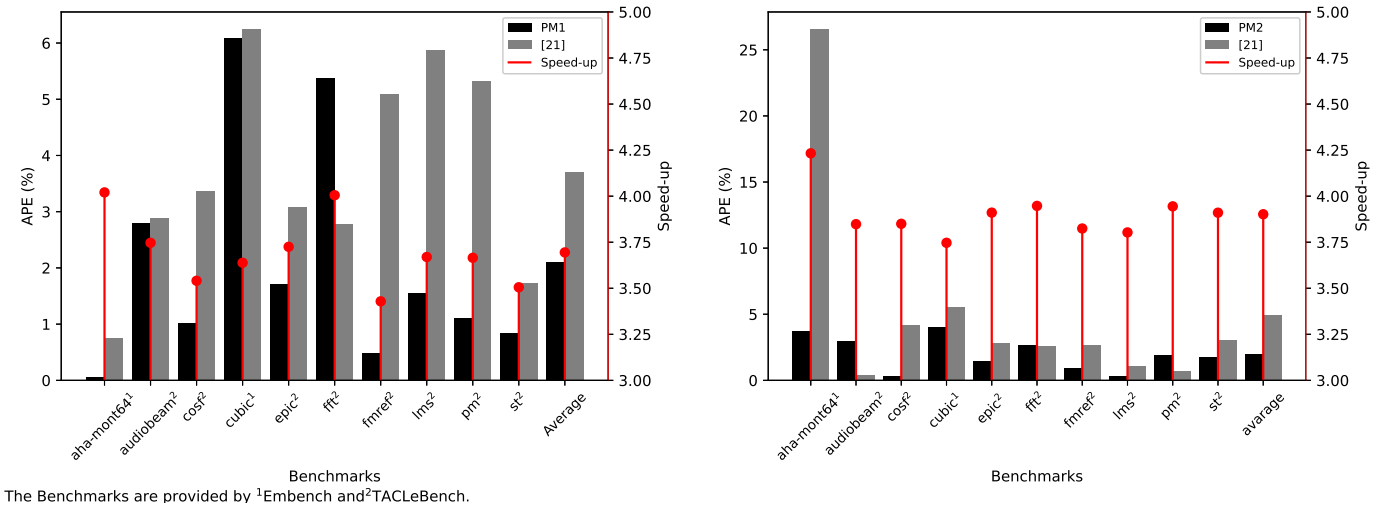
[1] S. Mashimo, A. Fujita, R. Matsuo, S. Akaki, A. Fukuda, T. Koizumi, J. Kadomoto, H. Irie, M. Goshima, K. Inoue *et al.*, "An open source FPGA-optimized out-of-order RISC-V soft processor," in *2019 International Conference on Field-Programmable Technology (ICFPT)*. IEEE, 2019, pp. 63–71.

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

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

[4] 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.

[5] ——, "Scalable simulation-based verification of systemc-based virtual prototypes," in *2019 22nd Euromicro Conference on Digital System Design (DSD)*. IEEE, 2019, pp. 522–529.



(a) RV32I (b) RV32IM

Fig. 3: Experimental Results of all Benchmarks used for Validation of PM on RV32I and RV32IM

TABLE III: Experimental Results of all Benchmarks used for Validation of PM on RV32IC

Benchmark	**#instr-exec.**	**LoC**	**VP [14]**		**Ours (PM3)**			
			#Cycle	**Time (ms)**	**Time (ms)**	**Speed-up**	**#Cycle**	**Error**
audiobeam[2]	7,536,543	6655	9,895,378	5142	1319	3.90	9,897,134	+0.02%
bigint[3]	872,838,943	581	1,071,777,343	573,967	139,584	4.11	1,116,592,512	+4.18%
epic[2]	81,706,654	982	102,365,017	49,040	13,407	3.66	103,162,696	+0.78%
fft[2]	3,678,523	492	4,838,065	2533	590	4.29	5,063,868	+4.67%
fmref[2]	19,660,756	648	24,708,410	12,370	3394	3.64	24,630,996	-0.31%
gsm_enc[2]	26,855,420	1793	34,116,433	17,404	4257	4.01	34,211,640	+0.28%
huff_enc[2]	1,016,104	393	1,646,751	843	191	4.41	1,562,465	-5.12%
isqrt[2]	1,002,079	114	1,840,168	835	196	4.26	1,685,697	-8.39%
lms[2]	5,814,944	691	7,288,470	3671	951	3.86	7,365,168	+1.05%
st[1]	18,327,848	117	23,469,543	10,542	3147	3.35	23,088,254	-1.62%
MAPE								2.64%

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

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

[7] M. Goli and R. Drechsler, "PREASC: Automatic portion resilience evaluation for approximating systemc-based designs using regression analysis techniques," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 25, no. 5, pp. 1–28, 2020.

[8] C. Marantos, K. Salapas, L. Papadopoulos, and D. Soudris, "A flexible tool for estimating applications performance and energy consumption through static analysis," *SN Computer Science*, vol. 2, no. 1, pp. 1–11, 2021.

[9] R. Nozal, "Optimizing performance and energy efficiency in massively parallel systems," Ph.D. dissertation, University of Cantabria, 2022.

[10] M. Oyamada, F. R. Wagner, M. Bonaciu, W. Cesario, and A. Jerraya, "Software performance estimation in MPSoC design," in *2007 Asia and South Pacific Design Automation Conference*. IEEE, 2007, pp. 38–43.

[11] X. Zheng, H. Vikalo, S. Song, L. K. John, and A. Gerstlauer, "Sampling-based binary-level cross-platform performance estimation," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. IEEE, 2017, pp. 1709–1714.

[12] R. Hundt, "HP caliper: A framework for performance analysis tools," *IEEE Concurrency*, vol. 8, no. 4, pp. 64–71, 2000.

[13] A. Srivastava, A. Edwards, and H. Vo, "Vulcan: Binary transformation in a distributed environment," MSR-TR-2001-50, Microsoft Research, Tech. Rep., 2001.

[14] 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.

[15] D. B. Noonburg and J. P. Shen, "Theoretical modeling of superscalar processor performance," in *Proceedings of the 27th annual International Symposium on Microarchitecture*, 1994, pp. 52–62.

[16] T. Sherwood, E. Perelman, and B. Calder, "Basic block distribution analysis to find periodic behavior and simulation points in applications," in *Proceedings 2001 International Conference on Parallel Architectures and Compilation Techniques*. IEEE, 2001, pp. 3–14.

[17] B. Franke, "Fast cycle-approximate instruction set simulation," in *Proceedings of the 11th international workshop on Software & compilers for embedded systems*, 2008, pp. 69–78.

[18] 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.

[19] M. Lattuada and F. Ferrandi, "Performance modeling of embedded applications with zero architectural knowledge," in *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, 2010, pp. 277–286.

[20] D. Wijesundera, A. Prakash, T. Srikanthan, and A. Ihalage, "Framework for rapid performance estimation of embedded soft core processors," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 11, no. 2, pp. 1–21, 2018.

[21] 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.

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

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

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

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

[26] "Keras api," https://keras.io/.

[27] "Random search," https://keras.io/api/keras_tuner/tuners/random/.