Learning-Based CPU Power Modeling

Ajay Krishna Ananda Kumar and Andreas Gerstlauer
Department of Electrical and Computer Engineering
The University of Texas at Austin
{ajaykrishna1111, gerstl}@utexas.edu

***Abstract*—With the end of Dennard scaling, energy efficiency has become an important metric driving future processor architectures, particularly in the fields of mobile and embedded devices. To support rapid, power-aware micro-architectural design space exploration, it is important to accurately quantify the power consumption of the processors early in the design flow and at a high level of abstraction. Existing CPU power models rely on either generic analytical power models or simple regression-based techniques that suffer from large inaccuracies. More recently, machine learning techniques have been proposed to build accurate power models. However, existing approaches still require slow RTL simulations or have only been demonstrated for fixed-function accelerators at higher levels.**

In this work, we present a machine learning-based approach for power modeling of programmable CPUs at the micro-architecture level. Our models provide cycle-accurate and hierarchical power estimates down to sub-block granularity. Using only high-level information that can be obtained from micro-architecture simulations, we extract representative features and develop low-complexity learning formulations that require a small number of gate-level simulations for training. Results show that our hierarchically composed model predicts cycle-by-cycle power consumption of RISC-V processor core within 2.2% of a gate-level power estimation on average.

***Index Terms*—Machine learning, power modeling, micro-architecture simulation**

I. Introduction

With the breakdown of Dennard scaling, power consumption, especially that of processors, is a first-order concern in all modern chips. Accurately quantifying the power consumption through power analysis in early design stages is crucial for power-aware hardware and processor design.

Accurate power estimation relies on gate-level analysis, which comes at the cost of long simulation times and available only in very late phases of the design flow. At the register-transfer level (RTL), industry tools such as PowerArtist and PowerPro can provide accurate aggregate power estimates sufficient to highlight coarse-grain RT-level power saving opportunities. Regression-based approaches [1], [2], [3] support building power models at a finer granularity, but at the expense of decreased accuracy. More recently, advanced machine learning (ML) approaches using deep neural networks (DNNs) have demonstrated the capability for highly accurate RTL power estimation [10]. However, deep learning requires a large amount of training data to be obtained from gate-level reference simulations. Furthermore, the need for slow RTL simulations limits the usefulness and extent of design space exploration that is possible with any RTL power estimation.

This work was supported by Samsung GRO and NSF grant CCF-1763848.

Early design space exploration of CPUs is most commonly performed at an abstract micro-architecture level. Traditionally, spreadsheet-based or generic analytical power models [4] are used to provide power estimates at this level. However, such models have been shown to be highly inaccurate [5]. Regression methods have also been applied instead to model power at higher instruction and micro-architecture levels [6], [9], but they often similarly suffer from larger inaccuracies due to the challenge of modeling the non-linear power characteristics of the underlying circuits accurately at such high levels of abstraction. Advances in machine learning have made it possible to accurately capture such complex relationships. At the same time, training and inference costs should not negate the speed benefits of working at a higher abstraction level. This rules out expensive deep learning approaches. Instead, dedicated learning formulations that can achieve high accuracy with low complexity need to be developed. Such approaches have recently been provided for fixed-function accelerators [7], but they have not yet been applied to model programmable processors above RTL.

In this work, we present a micro-architecture level machine learning-based power model of programmable CPUs. Using high-level activity information available from micro-architecture simulations, we extract features and develop learning formulations that can capture correlations with minimal training overhead and complexity. Our models are trained on gate-level simulations of small instruction sequences. Trained models can then provide highly accurate cycle-by-cycle power estimates in a hierarchical fashion at the complete core level and down to different CPU sub-blocks. The specific contributions of this work are:

- We identify key representative features for modeling of common CPU blocks with high predictability and low complexity.
- We explore advanced non-linear regressors for power modeling of different micro-architectural blocks in a CPU with low training overhead and high accuracy.
- We propose a hierarchical model composition that accounts for glue logic in super-block power modeling.
- We apply our approach to power modeling of a RISC-V core. Results show that a decision tree based model can predict cycle-accurate power with less than 2.2% mean absolute error (MAE).

The rest of this paper is organized as follows: Section II first highlights related work and Section III then gives an overview of our power modeling flow. Section IV provides details of our CPU power modeling approach. Experimental results are presented in Section V. Finally, Section VI provides a summary and concluding remarks.

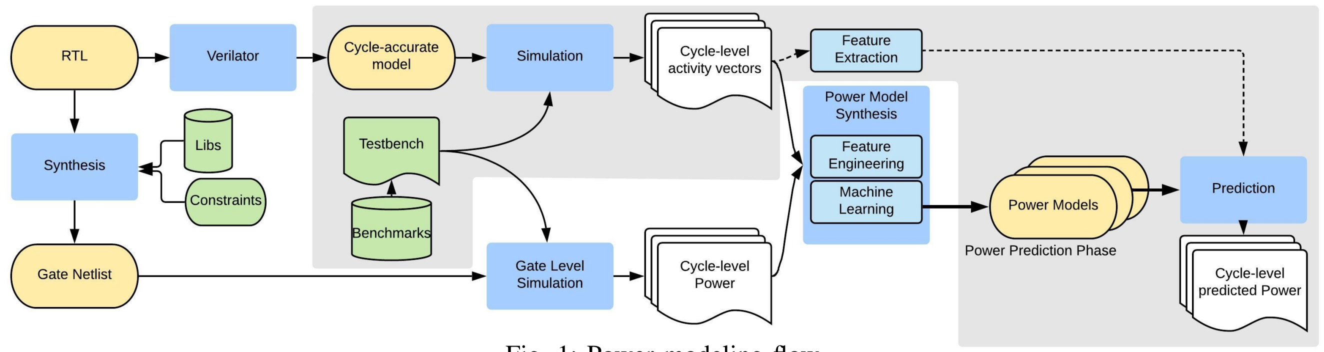


Fig. 1: Power modeling flow.

II. Related Work

Traditional approaches to micro-architecture level power (and performance) modeling can be classified broadly into analytical and regression based models. Analytical models such as [4] map sub-blocks to commonly used circuit structures and underlying physical technology models. Such approaches are generic and do not map well to one specific processor implementation. As such, they suffer from large inaccuracies [8]. It is possible to calibrate analytical models against low-level measurements [5], but the parameter fitting will limit interpretability at the sub-block level. Regression-based approaches instead rely on simulating an implementation, sampling and fitting generic regression equations for modeling CPU power at the pipeline or instruction level [6]. Other works [9] combine analytical approaches with regression equations formulated using pre-characterized power data from existing designs. However, all of these simple models still suffer from inaccuracies in modeling cycle-by-cycle power of a processor at fine sub-block granularity.

Several ML-based approaches for power modeling have recently been explored. PRIMAL [10] uses a convolutional neural network (CNN) for modeling RTL power trained from gate-level simulations using the combined activity of all registers in a design. Such a CNN-based model is very accurate but requires a large amount of training samples and training time compared to simple regression models. Traditional regression-based RTL models [1], [2], [3]. [3] propose various approaches for selecting critical signals and registers that are strongly correlated with power, but they are tied to simple, often linear models with limited accuracy. In all the above cases, proposed models rely on details available only at RTL or lower levels of abstraction during late design stages. Recent work [7] has shown the possibility of building ML-based power models at a higher, C++/ SystemC level of abstraction. The work proposes several feature selection and model decomposition techniques to enable highly accurate prediction using low-complexity non-linear regressors. However, it has only been demonstrated for fixed-function accelerator IPs. To the best of our knowledge, we are the first to apply a similar approach to programmable CPUs by adopting ML-based regression methods at the CPU sub-block level and hierarchically composing such models.

III. Methodology

Fig 1 shows an overview of our power modeling flow. The primary inputs are the gate level netlist and a cycle-accurate model of a processor. In this paper, we generate a cycle-accurate C++ model from the RTL description of the processor using the Verilator tool [12]. However, our approach only requires high-level activity information, and the Verilator model can be easily replaced with a high-level cycle-accurate, micro-architecture simulation model.

During the training phase, simulations are run at both gate and cycle-accurate levels using the same micro-benchmarks. Cycle-by-cycle per block reference power traces are generated using industry-standard gate-level simulations, and activity traces are extracted from the cycle-accurate model simulation. In the power model synthesis step, we extract features for the different functional blocks and apply functionality dependent feature selection and decomposition techniques to make the power model more accurate. Using extracted features and reference power values, a ML regressor is trained to learn the correlation between the decomposed features per block and the power consumed by that block across cycles. These learned models are then stored to be used during the prediction phase.

During the prediction phase, the full workload to analyze is simulated in the cycle-accurate model. Feature extraction and decomposition is applied to the activity information extracted from the simulation and previously trained models are used to predict cycle-by-cycle power per block hierarchically up to the full core level. Hierarchically decomposed power models down to the sub-block level thereby enable micro-architecture level exploration as pre-trained blocks can be arranged in different compositions and only blocks that are modified need to be re-trained or analytically scaled.

IV. Power Model

Effectiveness of supervised learning approaches depends on engineering features that are highly correlated with the values to be predicted as well as on selection of appropriate learning models that can capture underlying correlations with low overhead. Power consumption of a circuit specifically is sensitive to certain key contributor signals [1], [2]. ML-based hierarchical power modeling of CPUs thus involves the following steps: (i) identification of key contributing activity information, (ii)

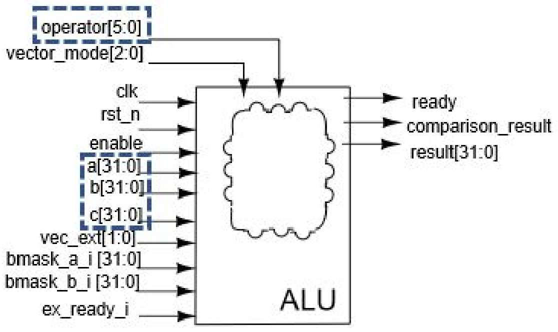


Fig. 2: ALU block with signals selected for prediction.

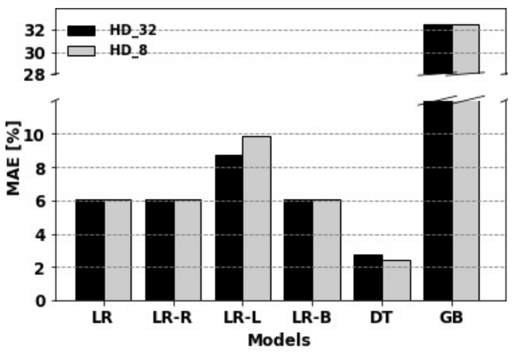


Fig. 3: ALU feature and model selection.

mapping of key contributing signals to features and feature engineering, (iii) model selection for each block, and (iv) super-block power model composition. This section covers the functionality dependent feature and model selection techniques for the three common categories of blocks found in the CPU, as well the handling of super-blocks for power modeling.

We limit feature selection to activity data that can be extracted from cycle-accurate performance models. We evaluate six linear as well as non-linear ML regressors to model the power consumption of different blocks in the CPU: (i) least-squares linear regression, (LR) (ii) linear regression with l2-norm regularization (LR-R), (iii) linear regression with L1 prior regularization (LR-L), (iv) a linear model with l2 regularization where the priors over the hyperparameters are chosen to be gamma distributions (LR-B), (v) a decision tree based regressor (DT), and (vi) a gradient boosting model of equivalent complexity with a regression tree fitted on the negative gradient of the loss function in each stage (GB). Details about our experimental setup are described in Section V.

A. *Data-dominant blocks*

Blocks that form the datapath in processors and that perform similar operations every cycle are categorized as data-dominant blocks. Intuitively, the power consumption of these blocks strongly depends on the activity of the data they process. Hamming distance (HD) has been widely used as a feature to concisely capture such data activity. At the same time, hamming distance of the entire data word has weak correlation to power. This is because of the difference in the circuit components that each toggling bit can effectively activate. For most of the commonly used datapath components, bits far off spatially (LSBs vs. MSBs) differ significantly, while those closer together (e.g. bits 0 and 1) show similar power behavior as a function of toggling activity. Based on this observation, byte-wise feature decomposition and hamming computation can provide good accuracy power models for these blocks, while still retaining simplicity of using hamming distance of byte groups in place of single bit switching traces as features.

Figure 2 shows a basic block diagram for the ALU in the RISC-V core used in our experiments (see Section V for details). We select the ALU operands and the ALU operating mode input as the only signals used for prediction and traced in the cycle-accurate simulations. Figure 3 summarizes the mean absolute prediction error (MAE) of using byte-wise decomposition of features (HD_8) compared to using the hamming distance of the entire word (HD_32) for data inputs across different learning models. As results show, a decision tree based model achieves significantly better accuracy than other models, where byte-wise feature decomposition helps to improve accuracy by an additional 0.25%.

We apply a similar byte-wise decomposition of operand input to other datapath units in the processor's execution stage. In case of the multiplier, the multiplier of the RISC-V core maintains an internal state for sub-word selection during the 4-cycle MULH (multiplication with upper word result) operation. This sub-word selection signal is control input for subsequent cycles of a multi-cycle operation and as such included as feature.

B. *Control blocks*

Blocks that activate different portions of the underlying circuit depending on the current value of different control signals are categorized as control-dominated blocks. For these blocks, both the switching of the data as well the current value of the control input determines the power consumption at every cycle. Consequently, both the current value as well as the switching activity (captured as hamming distance) are used as features for these blocks. For example, instruction decode and register read pipeline stages in a processor will activate different portions of the circuit depending on the state of the different control signal, such as the opcode, the source register to be read, etc.

Specifically, in case of the instruction decoder, the instruction word is the data that the decode stage processes in each cycle. Figure 4 shows the different bitfields in an instruction of the RISC-V IM instruction set. Rather than a generic byte-wise decomposition, the instruction word is sliced based on the sub-field boundaries in the instruction format to allow the model to learn the relation of each sub-fields with power. Figure 5 shows the error trend across different learning models with different feature decomposition techniques: HD_32 (hamming distance of the entire instruction word), HD_16 (hamming distance of half-words), HD_8 (hamming distance per byte), HD_BF (hamming distance per bitfield), ST+HD_BF (current value and hamming distance per bitfield). Again, a decision

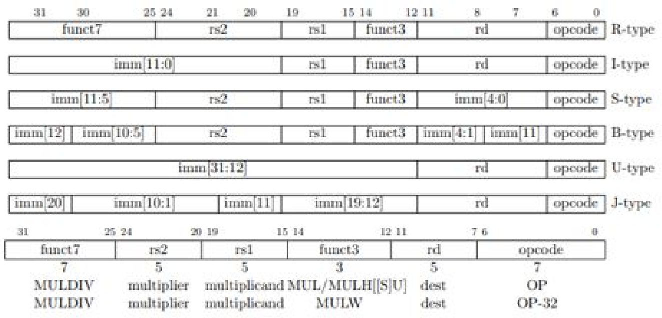


Fig. 4: Instruction word bitfields.

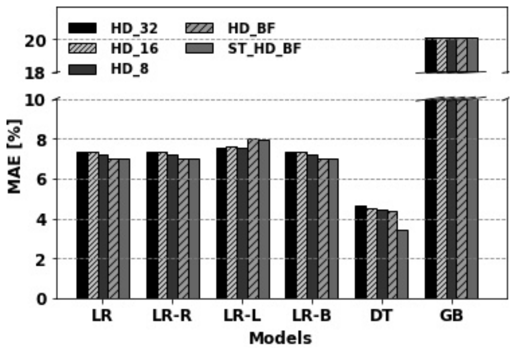


Fig. 5: Instruction decode stage feature and model selection.

tree model provides the best accuracy, where feeding both the current value and hamming distance per bitfield into the model provides between 0.9% and 1.2% better accuracy than other decompositions.

C. Memory blocks

Blocks such as physical register file and different buffers, whose main functionality is buffering of data with possibly multiple readers and writers are categorized as memory blocks. From our analysis, the majority of the sequential component's clock power has very low variance at cycle level granularity and can be very easily modeled as a constant bias term in regression models. The variance in power consumption in these blocks is dominated by the switching of muxes and routing logic driven by the data that is being written/read from at the current cycle. Based on this rationale, the input and output ports are selected as signals for the register file. Hamming distances of the entire word can be used as features since all the bits traverse similar paths in these blocks.

D. Model composition

To handle super-blocks in hierarchical power modeling, there are two possible approaches: (i) synthesize a separate power model for the super-block, or (ii) compose a power model for the super-block from the component power models. Though the first approach can generate accurate power models with the right set of features, the second approach has the advantage of reduced power model synthesis time and better architectural exploration support. However, the composition approach suffers from inaccuracies due to the additional glue logic that is present in the super-block not being modeled.

TABLE I: Power statistics of RI5CY core blocks.

Block	**Cells (cb/sq)**	**Avg.**	**Std. dev.**	**Max**	**Min**
Fetch	3853 / 316	0.29mW	0.05mW	0.42mW	0.11mW
Decode	14561 / 1631	0.49mW	0.12mW	1.01mW	0.35mW
Execute	11248 / 123	0.21mW	0.05mW	0.58mW	0.17mW
LS_unit	1250 / 42	0.03mW	0.02mW	0.13mW	0.02mW
CSR	5481 / 846	0.25mW	0.01mW	0.31mW	0.18mW
PMP	12643 / 1	0.16mW	0.06mW	0.54mW	0.12mW
Core	49904 / 3050	1.48mW	0.27mW	2.63mW	0.99mW

Such glue logic can be a significant contributor to total super-block power. From our gate-level analysis, glue logic at the core level can consume about 5% of the average total power in a very simple CPU and can contribute up to 10% on a cycle-by-cycle basis. Our approach to solving this problem is to treat the glue-logic as a virtual block and synthesize a power model for it. This is achieved by subtracting the sum of component powers from the total power during training to obtain the reference power for the glue logic block. During prediction, the glue logic block then forms a part of the composed super-block power model. Super-block power modeling, model composition with and without glue logic will be evaluated in Section V.

V. Experiments

We have evaluated our proposed approach to model a power consumption of a RISC-V based processor, the open-source RI5CY core that is part of the PULP platform [11] developed at ETH Zurich and the University of Bologna. It is a 4-stage, in-order 32-bit RISC-V processor core. It fully implements the RV32IMFC ISA and many other PULP-specific extensions such as post incrementing load-store, MAC operations, and hardware loops. For this work, the floating-point module was not instantiated. Figure 6 shows the major blocks in the core. The hierarchical decomposition for power modeling purposes is highlighted with dotted boxes. For generality, the physical memory protection (PMP) unit and control and status register (CSR) block with all the performance counters instantiated are also included for power modeling purposes. However, CSR is not included as part of Execute_stage power model, but as a separate block.

The open-source RI5CY core RTL is synthesized with the Nangate 45nm PDK using Synopsys Design compiler (L-2016.03-SP5). Seven benchmarks (aes_cbc, conv2d, fdctfst, fft, fir, keccak, matmul) from the pulpino test suites are chosen for training and evaluation of the models. The benchmarks are compiled using the riscv-gnu-toolchain and object code is used for the simulations. Synopsys VCS is used for the zero-delay gate-level simulation of the RI5CY core (@25MHz) with the chosen benchmarks using provided inputs. Activity vectors at sub-block and per-cycle level are then extracted from the simulation dumps, and Primetime PX (PTPX) is run in time-based power mode to generate cycle-by-cycle reference power numbers. Table I shows the gate counts and power statistics of the top level blocks when analyzed at the gate level using PTPX, while running all the seven benchmarks back to back.

The Scikit-learn Python package is used for model synthesis and prediction.

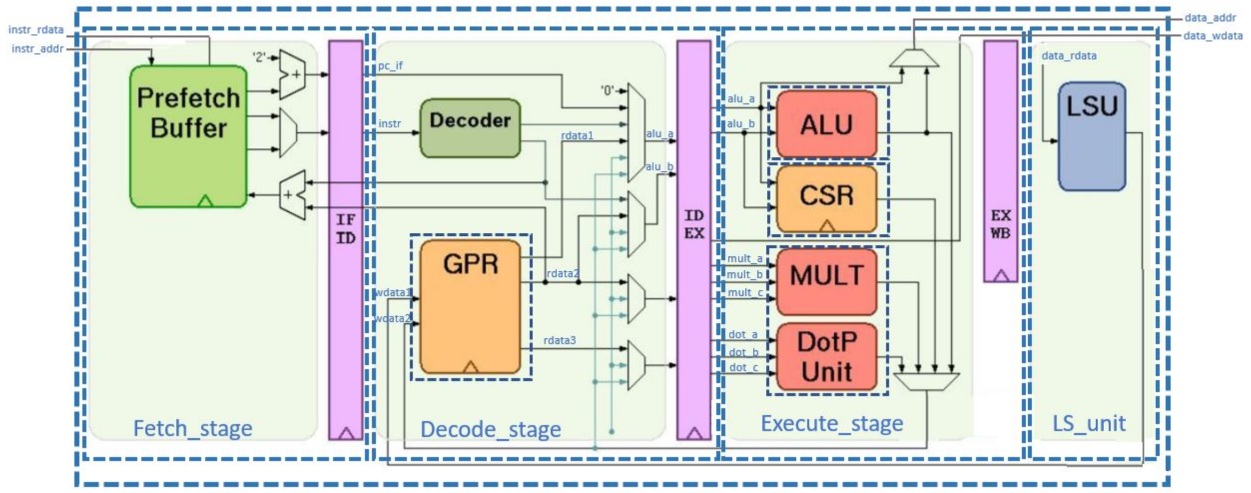
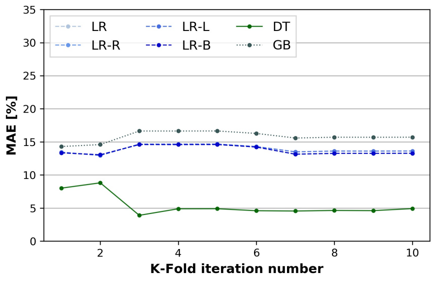
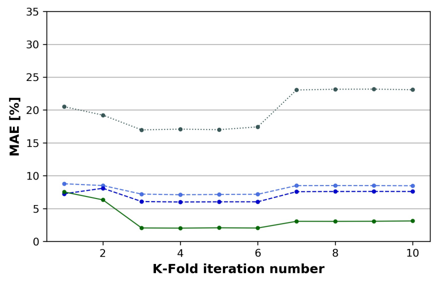


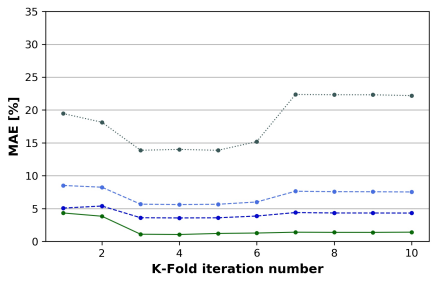
Fig. 6: RI5CY core and its sub-block decomposition.



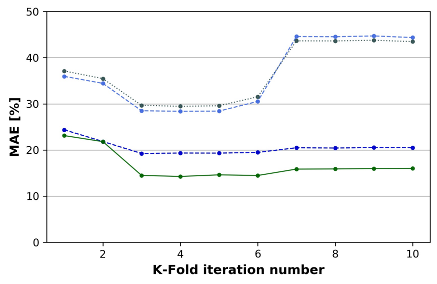
(a) Fetch_stage



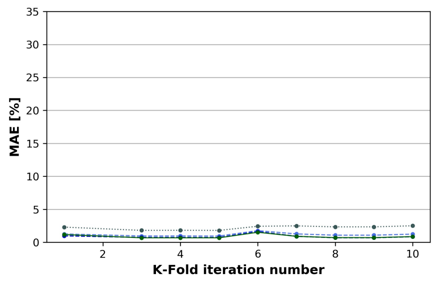
(b) Decode_stage



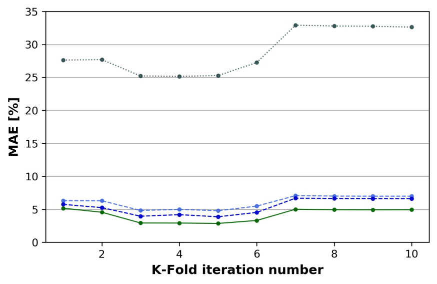
(c) Execute_stage



(d) LS_unit



(e) CSR



(f) Pmp_unit.

Fig. 7: 10-fold cross-validation results.

A. *Cross-validation results*

10-fold cross-validation is used for the evaluation of the feature correlation and accuracy of the models on the cumulative data samples constructed from the 7 benchmarks. This resampling procedure nullifies the model bias on the data split and thereby allow us to evaluate the pure feature correlations to power. We use cycle-by-cycle mean absolute error (MAE) of values predicted by each model compared to gate-level power estimation, normalized to mean reference power of the block as our evaluation metric.

Figure 7 summarizes the 10-fold cross-validations results for the 6 core-level sub-blocks. We can observe that the decision tree (DT) based model performs well compared to the linear models in all cases. As has been demonstrated in earlier work [7], decision tree-based data representations efficiently capture the inherent non-linear but typically discrete power behaviour of design blocks. Gradient boosting based models are of similar complexity, but perform poorly in most of the cases; this shows the necessity of more depth to capture the non-linearity. As listed in Table I, the CSR block has the least variance in power in the modeled core, which explains the high accuracy of the power models. On the other hand, low power but high variance blocks such as the LS_unit show poor accuracy in modeling with the evaluated models. Accurately modeling the power consumption of the blocks in this category needs more study and considered for future work. For this work, due to its small contribution to the total power of the core, a model with 16.6% error rate for the LS_unit is sufficient for gaining high accuracy for the core composed power model.

TABLE II: Top decision tree features for different blocks.

Block	Features (Importances)
Fetch_stage	HD(instr_addr) (0.32), instr_rdata (0.27),instr_addr (0.22), HD(instr_rdata) (0.19)
Decode_stage	HD(instr[24:20]) (0.70), HD(alu_a) (0.14), HD(instr[31:25]) (0.05), instr[11:7] (0.02), instr[24:20] (0.01), HD(instr[11:7]) (0.01), HD(alu_b) (0.01), instr[19:15] (0.01), instr[6:0] (0.01), instr[31:25] (0.01)
Execute_stage	HD(alu_a[7:0]) (0.51), HD(alu_a[23:16]) (0.19), HD(alu_operator) (0.10), HD(mult_a[7:0]) (0.07), HD(alu_b[23:16]) (0.03), HD(alu_b[7:0]) (0.02), HD(alu_a[15:8]) (0.02), HD(alu_b[31:24]) (0.02), HD(alu_a[31:24]) (0.02), HD(mult_b[7:0]) (0.01)
LS_unit	HD(data_rdata[15:8]) (0.52), HD(b[7:0]) (0.20), HD(b[31:24]) (0.07), HD(data_rdata[7:0]) (0.04), HD(a[7:0]) (0.02), HD(a[15:8]) (0.02), HD(b[15:8]) (0.02), HD(a[23:16]) (0.01), HD(a[31:24]) (0.01), HD(data_wdata[15:8]) (0.01)
CSR	HD(csr_wdata) (0.95), HD(pc_if) (0.03), HD(branch_i) (0.01)
Pmp_unit	HD(data_addr) (0.95), HD(instr_addr) (0.04)
Core	HD(pc_if) (0.69), HD(data_addr) (0.16), HD(instr_rdata[31:25]) (0.04), HD(csr_wdata) (0.02), instr_rdata (0.01), HD(alu_operator) (0.01), HD(alu_a) (0.01), instr_addr (0.01), HD(data_rdata[7:0]) (0.004), HD(instr_addr) (0.004)

TABLE III: Predicted power statistics of decision tree (DT) based power model.

Block	Avg. Power	Max power	Min Power	MAE	Max error	Avg. error
Fetch_stage	0.29mW	0.42mW	0.11mW	5.38%	66.0%	0.26%
Decode_stage	0.49mW	1.01mW	0.35mW	3.43%	58.0%	0.14%
Regfile	0.26mW	0.40mW	0.22mW	4.44%	64.0%	0.08%
Execute_stage	0.21mW	0.58mW	0.17mW	1.84%	90.9%	0.02%
ALU	0.11mW	0.35mW	0.07mW	4.28%	147%	0.10%
Multiplier	0.09mW	0.49mW	0.09mW	0.30%	235%	0.09%
LS_unit	0.03mW	0.13mW	0.02mW	16.65%	210%	0.34%
CSR	0.25mW	0.31mW	0.18mW	1.17%	16.9%	0.01%
Pmp_unit	0.16mW	0.54mW	0.12mW	4.15%	114%	0.01%
Core (standalone)	1.48mW	2.63mW	0.99mW	1.07%	40.7%	0.04%
Core (composed)	1.48mW	2.63mW	0.99mW	3.37%	34.7%	2.83%
Core (w/ glue logic)	1.48mW	2.63mW	0.99mW	2.15%	34.8%	0.006%

B. Learning rate, feature ranking and accuracy

Figure 8 shows the training curve and learning rate of different blocks for the best fold. The main learning overhead is the time required for reference gate-level simulations. As Figure 8 shows, in all cases, the decision tree model is able to learn power behavior with less than 300K cycle-level samples and hence instructions needing to be simulated, compared to the 2.2M samples required to train the CNNs in [10]. Table II lists the major features selected by the decision tree arranged in ascending order of importance (normalized to 1), where HD(x) denotes that the hamming distance of x. This ranking can convey additional information about the power behavior to drive power optimizations.
Finally, Table III summarizes the performance of the decision tree (DT) based power model for different blocks, including hierarchical composition of the core using either a standalone model or as the sum of sub-block models with and without a dedicated glue logic model. Modeling the core power with single DT model has a mean absolute error of 1.07%. By contrast, building a power model for core by composing block-level power models has a much higher MAE of 3.37%. This can be compensated and made more accurate by modeling the glue logic as a virtual block (with 2.15% MAE).

VI. Summary and Conclusions

In this paper, we presented a hierarchical power modeling approach that supports development of simple yet accurate power models for CPUs at micro-architecture levels of abstractions. Our approach provides cycle-accurate power estimates at sub-block granularity with low training overhead using features that are extracted from micro-architecture simulations. Using this approach, a decision tree based hierarchically composed model is built for a RISC-V core that can predict cycle-by-cycle power with less than 2.2% error rate.

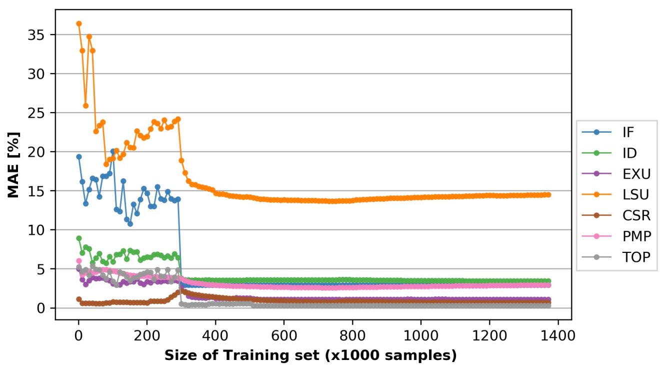


Fig. 8: Learning curve for different blocks.

References

[1] A. Bogliolo et al. Regression-Based RTL Power Modeling. *Transactions on Design Automation of Electronic Systems*, 2000.
[2] D. Sunwoo et al. PrEsto: An FPGA-Accelerated Power Estimation Methodology for Complex Systems. In *FPL*, 2010.
[3] J. Yang et al. Early Stage Real-Time SoC Power Estimation using RTL Instrumentation. In *ASPDAC*, 2015.
[4] S. Li et al. McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures. In *MICRO*, 2009.
[5] W. Lee, et al. PowerTrain: A learning-based calibration of McPAT power models. In *ISLPED*, 2015.
[6] Y. H. Park, et al. A multi-granularity power modeling methodology for embedded processors. *IEEE TVLSI*, Apr. 2011.
[7] D. Lee, et al. Learning-Based, Fine-Grain Power Modeling of System-Level Hardware IPs. *ACM TODAES* 23(3), 2018.
[8] Xi, Sam Likun, et al. Quantifying sources of error in McPAT and potential impacts on architectural studies. In *HPCA*, 2015.
[9] D. Brooks, et al. New methodology for early-stage, microarchitecture-level power-performance analysis of microprocessors. *IBM Journal of Research and Development*, 2003.
[10] Y. Zhou, et al. PRIMAL: Power Inference using Machine Learning. In *DAC*, 2019.
[11] D. Rossi, et al. PULP: A parallel ultra low power platform for next generation IoT applications. In *HCS*, 2015.
[12] W. Snyder. Verilator and SystemPerl Environment. NASCUG, 2004.