Machine Learning-Based Microarchitecture-Level Power Modeling of CPUs

Ajay Krishna Ananda Kumar, Sami Alsalamin,
Hussam Amrouch, *Member, IEEE,* and Andreas Gerstlauer, *Senior Member, IEEE*

Abstract—Energy efficiency has emerged as a key concern for modern processor design, especially when it comes to embedded and mobile devices. It is vital to accurately quantify the power consumption of different micro-architectural components in a CPU. Traditional RTL or gate-level power estimation is too slow for early design-space exploration studies. By contrast, existing architecture-level power models suffer from large inaccuracies. Recently, advanced machine learning techniques have been proposed for accurate power modeling. However, existing approaches still require slow RTL simulations, have large training overheads or have only been demonstrated for fixed-function accelerators and simple in-order cores with predictable behavior. In this work, we present a novel machine learning-based approach for microarchitecture-level power modeling of complex CPUs. Our approach requires only high-level activity traces obtained from microarchitecture simulations. We extract representative features and develop low-complexity learning formulations for different types of CPU-internal structures. Cycle-accurate models at the sub-component level are trained from a small number of gate-level simulations and hierarchically composed to build power models for complete CPUs. We apply our approach to both in-order and out-of-order RISC-V cores. Cross-validation results show that our models predict cycle-by-cycle power consumption to within 3% of a gate-level power estimation on average. In addition, our power model for the Berkeley Out-of-Order (BOOM) core trained on micro-benchmarks can predict the cycle-by-cycle power of real-world applications with less than 3.6% mean absolute error.

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

1 Introduction

With the end 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. Power modeling has been widely researched across abstraction levels in the past. Figure 1 aims to categorize existing power modeling works at different levels of abstractions versus modeling effort, where shading is used to indicate relative accuracy of different approaches.

The ultimate gold standard for accurate power signoff is gate-level analysis, which comes at the cost of very long simulation times and available only in very late phases of the design flow. This has driven the need for power modeling at higher levels of abstraction. At the register-transfer level (RTL), industry tools such as PowerArtist [1] and PowerPro [2] can provide aggregate power estimates sufficient to highlight coarse-grain RTL power saving opportunities. Regression-based approaches [3], [4], [5], [6] support building power models at a finer granularity, but at the expense of similarly limited accuracy. More recently, advanced machine learning (ML) approaches using deep neural networks (DNNs) have demonstrated the capability for highly accurate RTL power estimation [7]. However, deep learning requires a large amount of training data (in the range of millions of samples) to be obtained from gate-level reference simulations. If collecting training information is as expensive as running the workload in question itself, the overheads can easily outweigh the benefits. Furthermore, the need for slow RTL simulations limits the usefulness and extent of design space exploration that is possible with any RTL power estimation.

Early design-space exploration of CPUs is most commonly performed at an abstract micro-architecture level. Traditionally, generic spreadsheet-based, analytical models [8] are used to provide power estimates at this level. However, such models have been shown to be highly inaccurate [9]. Regression-based methods have also been applied instead to either calibrate existing analytical models [10], [11] or to model power at higher instruction and micro-architecture levels [12], [13], but they similarly suffer from large inaccuracies due to the challenge of modeling the non-linear and data-dependent power characteristics of the underlying circuits accurately at such high levels of abstraction. Data-dependent activity, in particular, can have a significant impact on power even when averaged across a larger number of samples, but especially when aiming to model power variations at fine temporal and spatial granularity. At the same time, models that can predict power variations down to the sub-block level with cycle accuracy for complete workloads are crucial for the estimation of peak power consumption, power/thermal hot-spots, voltage noise, IR drops, etc. Traditional CPU power models at a fast micro-architecture level can not provide the necessary temporal and spatial information with the required accuracy to support such use cases.

Advances in ML have made it possible to accurately capture complex non-linear relationships with high accuracy and fast prediction. At the same time, training and inference costs should

A. K. Ananda Kumar and A. Gerstlauer are with the Department of Electrical and Computer Engineering, The University of Texas at Austin, Austin, TX, 78712 USA (e-mail: ajaykrishna1111@utexas.edu, gerstl@ece.utexas.edu).
S. Alsalamin is with Hyperstone®, Konstanz 78467, Germany (e-mail: salsalamin@hyperstone.com , sami@ppu.edu). (The work of Alsalamin was done in part when he was at Karlsruhe Institute of Technology (KIT))
Hussam Amrouch is with the Chair of Semiconductor Test and Reliability (STAR) in the Computer Science, Electrical Engineering Faculty, University of Stuttgart, 70174 Stuttgart, Germany (e-mail: amrouch@iti.uni-stuttgart.de).
Manuscript received xxx; revised xxx.

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 proposed for power modeling of fixed-function accelerators [14]. In previous work, we have developed an initial learning-based power modeling approach for simple embedded in-order CPUs above RTL [15]. In this paper, we extend our prior work to: (1) generalize our modeling approach to a wider range of processor micro-architectures including complex superscalar out-of-order (OoO) CPUs with deeply pipelined internal structures, (2) introduce novel feature selection and modeling concepts for advanced hardware aspects including data and clock gating, (3) apply our approach to power modeling of an industry-strength open-source OoO CPU, and (4) present more comprehensive and detailed experimental results for both in-order and out-of-order cores including validation on real-world application test sets and additional sensitivity studies.

We present a comprehensive methodology to develop novel machine learning-based micro-architecture level power models for complex CPUs that can provide accurate power estimates at fine temporal and spatial granularity. Using high-level activity information available from micro-architecture simulations, we extract features and develop learning formulations that can capture correlations with minimal complexity. Our models are trained on gate-level simulations of small micro-benchmarks. Trained models can then provide highly accurate cycle-by-cycle power estimates for arbitrary application workloads in a hierarchical fashion at the complete core level and down to different micro-architecture sub-blocks. Our specific contributions are:

- We present a generic and systematic methodology for feature selection and feature engineering to model common in-order and out-of-order CPU structures using activity information available at the micro-architectural level.
- We explore advanced non-linear regressors for power modeling of different micro-architectural blocks in CPUs with low training overhead and high accuracy.
- We propose a hierarchical model composition approach that supports power models at sub-block granularity while accounting for glue logic in super-block power.
- We demonstrate our power modeling approach on RI5CY, an in-order RISC-V core from the PULP platform, and the Berkeley Out-of-Order Machine (BOOM), a superscalar Out-of-Order (OoO) RISC-V core. We identify key representative features for modeling of common CPU blocks with high predictability and low complexity. Our hierarchically composed power models for RI5CY and BOOM cores have an average error rate of 2.2% and 2.9%, respectively, compared to final placed & routed gate-level power estimation. We have released our setup including pre-trained power models at [16].

The rest of the paper is organized as follows: after a review of related work in Section 2, we provide an overview of our modeling flow in Section 3. Details of our approach are provided in Section 4. Section 5 presents experimental results. Finally, Section 6 concludes with a summary and outlook.

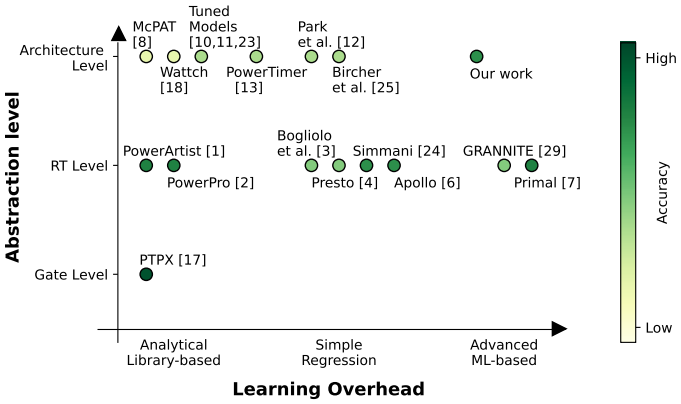


Fig. 1. CPU power modeling landscape.

2 Related Work

Figure 1 gives an overview of some of the key existing works and categorizes them based on the level of abstraction and the approach they use for modeling. In the following sections, we will further review and contrast the works in each category.

Analytical and library-based models: The common industry approach to get highly accurate power estimates for a design is to use power-characterized standard cell libraries with commercial tools such as Synopsys PrimeTime PX [17] by feeding in gate-level netlist and simulation results. Even though they are very accurate and can provide fine-grain power estimates, the requirement for a netlist necessitates that such analysis occurs late in the design process, where slow gate-level simulations hamper micro-architectural exploration. Tools such as PowerArtist [1] and PowerPro [2] have been developed to speed-up the process of providing power feedback and can provide a course-grain power estimate at the register transfer level (RTL). However, even RTL simulations are typically too slow for early design space exploration of CPUs at an abstract micro-architecture level. Analytical power models [8], [18], [19] that are coupled either with analytical performance models [20], [21] or with micro-architectural software simulators such as gem5 [22] are generally used during this exploration phase. Tools such as McPAT [8] analytically model the power at the physical technology level, using physical parameters of the devices and wires, and then map the sub-blocks to commonly used circuit structures and underlying physical technology models.

Such approaches do not account for data dependencies and only provide coarser-grain average power estimates based on overall activity information. As such, they are inherently not able to accurately estimate fine-grain cycle-by-cycle power variations, and thus target a different (but complementary) goal and scope than our work. Furthermore, they are generic, do not map well to specific implementations, and suffer from large inaccuracies even when carefully tuned to the evaluated processor. Prior studies [9], [23] have repeatedly shown the inaccuracy of their basic versions, which are based on technology assumptions that were valid when these tools were developed in some cases over 10 years ago. It is possible to calibrate analytical models against low-level measurements [10], [11], [23], but they are fundamentally still limited by their coarser-grain nature and the parameter fitting will limit interpretability at the sub-block level. The work in [23], for example, showed less than 6% error for the BOOM core when predicting average power at whole benchmark granularity. By contrast, our work predicts

average benchmark power with less than 1.5% error while also being able to predict cycle-level power at up to 4% error. In general while coarse-grain approaches are often sufficient for high-level explorations, fine-grain cycle-accurate models targeted in our work can complement such approaches by allowing for estimation of time-varying effects such as peak power, voltage droop, etc.

Statistical & regression-based models: In contrast to analytical approaches, regression-based power modeling is an extensively researched area at higher abstraction levels. Regression-based RTL models [3], [4], [5], [6], [24] propose various approaches for selecting critical signals and registers strongly correlated with power and training regression models from gate-level power analysis results These approaches typically deploy simple linear models, which are fast but do not capture the non-linear relationship in complex designs [7] and hence are limited in accuracy while requiring slow RTL simulations. By contrast, regression-based approaches at the architecture level rely on simulating an implementation, sampling and fitting generic regression equations for modeling CPU power at the pipeline or instruction level [12]. Methodologies based on correlating performance counters to power are another common power modeling approach at the architecture level [25], [26], [27], [28]. Other works [13] combine analytical approaches with regression equations formulated using pre-characterized power data from existing designs. However, all of these models still suffer from inaccuracies in modeling data-dependent, cycle-by-cycle power of a processor at fine sub-block granularity.

Advanced machine-learning based models: Recently advanced ML-based approaches for power modeling have been explored to capture the non-linear power-feature correlation. GRANNITE [29] use graph neural networks to relate RTL activity to gate-level toggle rates, but it can only predict average power at coarse temporal granularity (achieving around 5% error at a granularity of 1000 cycles for a similar in-order RISC-V core as used in our work). PRIMAL [7] uses a convolutional neural network (CNN) for modeling cycle-by-cycle 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 (5% cycle-by-cycle error as reported in [7] for an in-order RISC-V core), but requires a large number of training samples and training time compared to simple regression models. The proposed model also relies on details available only at RTL or lower levels of abstraction, and thus is not ideal for early design space exploration. Recent work [14] has shown the possibility of building ML-based power models at the C++/SystemC source 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 with predictable behavior. Our proposed approach is aimed at modeling programmable CPUs by adopting similar supervised learning-based regression methods at the CPU sub-block level, and then hierarchically composing such models.

3 Overview

Figure 2 shows an overview of our power modeling flow. The flow follows a supervised learning methodology with a training and prediction phase. The primary inputs are the gate-level netlist (for training) and a cycle-accurate micro-architecture simulation model of a processor (for training and prediction). Due to the lack

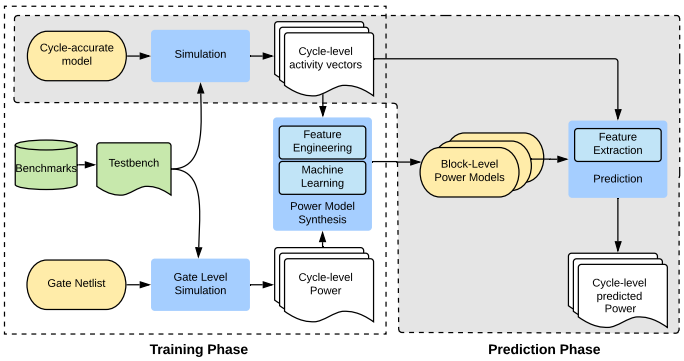


Fig. 2. Power modeling flow.

of availability of cycle-accurate simulators for our chosen RISC-V cores, in this work, we generate a cycle-accurate C++ model from the RTL description of the processor using the Verilator tool [30]. 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 power analysis tools from the 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 systematic feature selection and decomposition techniques depending on the functionality and attributes of the blocks. Using extracted features and reference power values, an ML regressor is trained to learn the correlation between the decomposed features per block and the power consumed by that block across cycles. The per-block learned models are then stored to be used during the prediction phase.

During prediction, the actual workload to analyze is simulated in the cycle-accurate model. Feature extraction and decomposition are applied to the extracted activity information only, and previously trained models are used to predict cycle-by-cycle power per block hierarchically up to the full core level. Our models are trained specific to a processor and its implementation, but hierarchically decomposed power models down to the sub-block level enable micro-architecture design space exploration, where pre-trained blocks can be arranged in different compositions, and only blocks that are modified need to be re-trained or analytically scaled. Note that to that increase flexibility, models can be parameterized to capture different processor or sub-block configurations (such as ROB sizes) in a single model without requiring re-training. This requires training models on different configuration parameters as additional features. There are typically only a limited set of discrete configuration settings. Models can either be pre-trained for all settings, or model can be setup to learn the interpolation between a subset of trained configurations.

In this work, our models target the CPU core only. However, our approach ultimately works by modeling power at a general RTL sub-component level and could in concept also be applied to other un-core components by selecting the right features following the principles outlined in this paper. Modeling power of memories is different as they are not based on normal RTL structures but regular

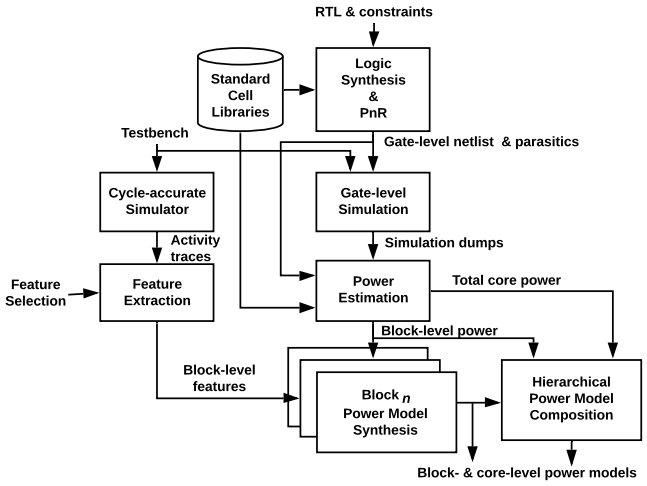


Fig. 3. Power model synthesis methodology.

memory arrays. For such structures, other existing approaches such as CACTI [31] can provide accurate estimates.

In summary: Our models are aimed at predicting power at cycle-by-cycle and sub-component level. At this fine granularity, data-dependent effects play a significant role. As such, features at equivalent fine granularity are required, where we present a methodology for selection of features that only rely on abstracted data at black-box component interfaces available from high-level micro-architecture simulations. Our work is complementary to approaches at this level that are aimed at only predicting averaged power at coarser temporal granularity using coarse-grain average activation numbers. While machine learning models adopted in this work use standard machine learning toolboxes, unlike deep learning approaches that are applied directly on raw data and require a large amount of training data, we aim to develop learning formulations that can predict power using only abstracted data and requiring low training overhead. The success or failure of such approaches depends to a large extend on developing an appropriate modeling formulation and engineering appropriate features. This is inherently problem-specific, but has only been studied at lower levels of abstraction (RTL or below) so far. Our key contribution is in providing a methodology for feature selection and hierarchical model de-composition that supports power modeling at the micro-architectural block level with high accuracy and low training overhead. To the best of our knowledge, no such approach has been presented at this level of abstraction before.

4 Power Model Synthesis

The effectiveness of supervised learning approaches depends on the selection of features that are highly correlated with the values to be predicted as well as appropriate learning models that can capture underlying correlations with low overhead. In particular, the power consumption of a circuit is sensitive to certain key contributor signals [3], [4]. ML-based hierarchical power modeling of CPUs thus involves the following steps: (i) identification of key contributing activity information, (ii) 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 presents our approach for feature engineering and model selection for common categories of micro-architecture blocks, advanced techniques to model common attributes and structures found in typical CPU implementations as well as the handling of super-blocks for power modeling.

Figure 3 details our power modeling methodology. We compose power models hierarchically on a micro-architectural component basis as defined by the block hierarchy in the processor's RTL description. We synthesize the RTL netlist and perform gate-level simulations to extract per-block cycle-by-cycle power estimates. In this work, we synthesize RTL descriptions such that the RTL block hierarchy in the gate-level netlist is maintained. In case of flattening during synthesis, power estimation must be extended with support for attributing the power consumption of each cell to blocks. Per-block reference estimates are combined with block-level features to synthesize a power model for each block, where block-level features are extracted from activity information collected during micro-architecture simulations following our feature selection approach described in the following sections. Finally, sub-block power models are hierarchically composed into power models at the super-block and whole-CPU levels.

We limit our feature selection to activity information that can be extracted from cycle-accurate, micro-architectural performance models, such as MARSS [32], SimpleScalar [33] or gem5 [22] augmented to trace the data activity. We evaluate the following 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) linear regression with l2 regularization and gamma distributions as hyperparameter priors (LR-B), (v) a decision tree based regressor (DT), (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), and (vii) a random-forest model with the number of estimators fitted to match the decision tree complexity (RF). We compare our ML-based models against a model predicting average power across the training set (Avg). We applied an additional decision tree-based feature sub-selection in all cases. We have chosen these regressors because they are commonly used approaches in higher-level power modeling. Traditionally, linear regressors in various forms have been used. In addition, decision trees were found to be a good fit for power modeling [14], [34], [35], where gradient boosting and random forest extensions of decisions trees were also evaluated. Other lower-level (RTL) approaches used deep neural networks. However, such approaches use raw signal data being as features and incur significant training overhead that negates many benefits of higher-level modeling. In this work, we target ML approaches that can work with limited feature sets and low training overhead.

4.1 Feature Selection

The power consumption of any micro-architectural block can be accurately modeled with a supervised machine learning model by capturing the correlation of power to signal activities used as features. We aim to model a block's power using only high-level activity of major micro-architecture signals connecting blocks. As such, we develop black-box power models that only leverage the activity of each block's external inputs as features.

Figure 4 shows our feature selection approach for a generic micro-architecture block. Available high-level signal information can be broadly categorized into data signals and control signals based on

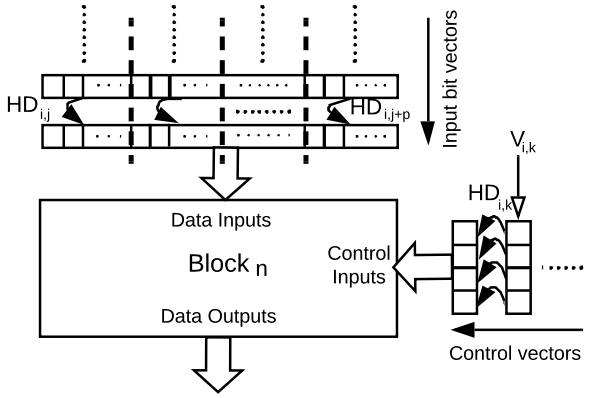


Fig. 4. Feature selection approach.

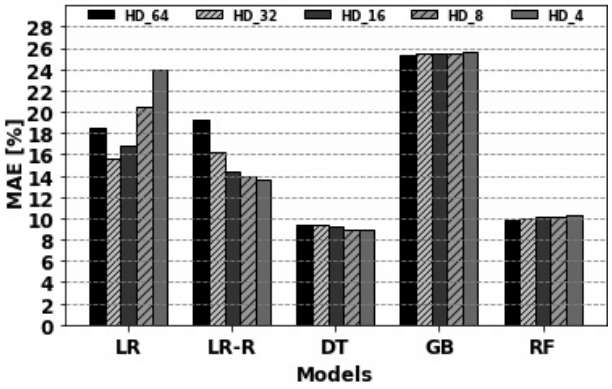


Fig. 5. ALU feature and model selection.

the functionality they trigger and the path they traverse in the CPU blocks. Consequently, feature selection and feature engineering are performed based on a categorization of different input signals for efficiently capturing the varying power behavior. The following sub-sections describe our methodology for the common signal categories and for signals with mixed characteristics.

4.1.1 Data signals

Data signals include all the multi-bit signals processed by different datapath circuits, routed, buffered, and compared in different pipeline stages of a CPU. Intuitively, the power consumption strongly depends on the toggling activity of these multi-bit signals. Hamming distance (HD) has been widely used as a feature to capture such activity concisely. At the same time, HD of the entire multi-bit data has weak correlation to power. This is due to the difference in 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. By contrast, those closer together (e.g. bits 0 and 1) show similar power behavior as a function of toggling activity.

Based on this observation, multi-bit data can be decomposed into smaller contiguous bit-groups. The HD of these bit-groups can be separately captured to obtain features with a strong correlation to the power consumed. As shown in Figure 4, signal vectors applied to the data inputs of a block are decomposed into bit groups, and Hamming distances $HD_{i,j}$ between bit vectors of group j in cycles i and $i+1$ are computed and used as features for prediction.

Figure 5 shows the mean absolute prediction error (MAE) of different decomposition for the ALU block of the BOOM core as described in Section 5. We compare decompositions using hamming distances at varying bit-widths ranging from 4-bit groups (HD_4) to whole data words (HD_64) for the ALU operands across different learning models. As results show, a finer granularity can improve accuracy across models. At the same time, a too fine granularity can lead to increased model complexity and overfitting. Optimal granularity depends on the model, where non-linear models can learn variations independently and benefit little from decomposition. A decision tree has the best accuracy across models. In case of a DT model, byte-wise hamming decomposition (HD_8) improves accuracy by an additional 0.29%, while DT starts overfitting for nibble-wise decomposition and becomes slightly worse. Based on this analysis, we model data signals with byte-wise decomposition, which provides good accuracy while still retaining the simplicity of using the hamming distance of byte groups in place of single bit switching traces as features and avoiding overfitting.

4.1.2 Control signals

Control signals include inputs that determine the mode of operation of different CPU blocks. Each mode typically activates different portions of the circuit and can consume a very different amount of energy. Therefore, it is crucial to capture which specific portion of the circuit is active or idle in each cycle to develop accurate power models. Also, mode switches can cause significant power variances at the cycle-by-cycle level. For example, a shift from normal read to update mode of a branch predictor would cause significant power deviation. As shown in Figure 4, for control signals, we propose to use the current bit-wise values $V_{i,k}$ and Hamming distances $HD_{i,k}$ of the control input k in cycle i as features to model the absolute power consumption and power variance, respectively.

4.1.3 Mixed data/control signals

Although most of the signals of the common CPU blocks can be independently categorized as data or control signals, some block inputs possess mixed characteristics and hence need special handling. For example, for an instruction decoder, the instruction word affects both the mode and the data that the decode stage processes in each cycle. 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-field with power, as shown in Figure 6. We thereby decompose such mixed data/control signals into bit-fields that correspond to the smallest granularity across all types of instruction words. Figure 7 shows the error trend across different learning models with different feature decomposition approaches. We compare using the hamming distance of the entire instruction word (HD_32), of half-words (HD_16), per byte (HD_16) and per bitfield (HD_16) with using both the current value and the hamming distance per bitfield (V+HD_BF). In general, decoder power as a function of input activity does not vary significantly across instruction fields, and decomposition provides few benefits. However, using both values and hamming distances to capture modal behavior can improve results in some cases. Again, a decision tree model provides the best accuracy, where feeding both the current value and hamming distance per bit field into the model provides between 0.9% and 1.2% better accuracy than other decompositions.

Note that that our approach is based on modeling power at a sub-component and cycle level. As such, it should be applicable to any design that can be broken down into such micro-architectural

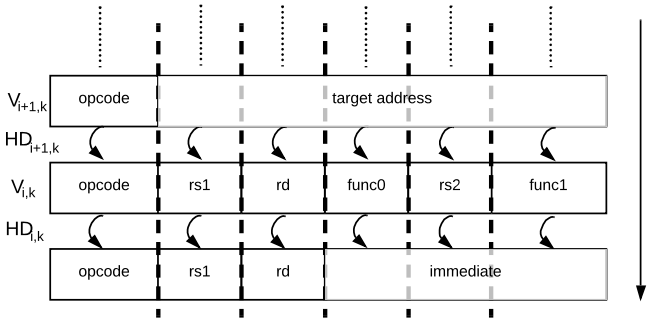


Fig. 6. Instruction word bit-field decomposition.

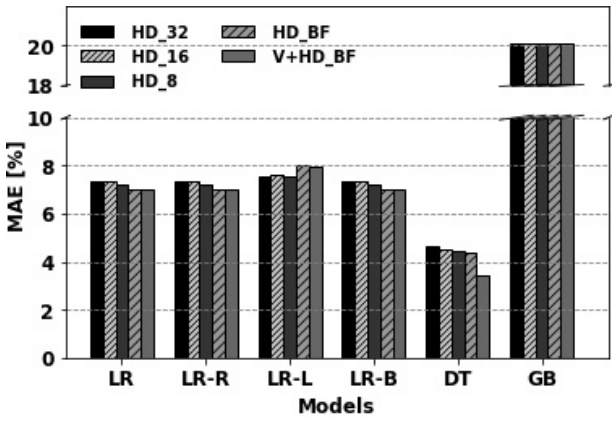


Fig. 7. Decode stage feature and model selection.

sub-components. Specifically, while we evaluate our approach on RISC designs in this paper, many CISC processors internally break down complex instructions into micro-operations that are executed on a RISC-like internal core. However, a CISC front-end will have additional complexity and in particular, irregularity, e.g. in instruction formats. This may require separating models and performing feature decomposition specific to each instruction class. If the instruction class is provided as additional feature, a decision tree should be able to inherently learn such a decomposition. Alternatively, one can learn different models for each instruction class and apply the correct model at prediction time.

4.2 Advanced Features

In addition to normal data and control inputs, some blocks can have behavior that requires special modeling. This section describes our methodology for modeling advanced attributes and structures in a typical CPU implementation.

4.2.1 *Buffer modeling*

One common attribute of CPU blocks is storing data, meta-data, and control information such as micro-ops in buffers with possibly multiple readers and writers. The majority of sequential component clock power has very low variance at cycle-level granularity. It can be 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 being written/read in the current cycle. Based on this rationale, the read and write data and addresses are selected and treated as data signals to model buffer properties, as shown in Figure 8.

Notably, such a buffering attribute is common in many superscalar out-of-order CPU blocks, ranging from a branch target buffer to a re-

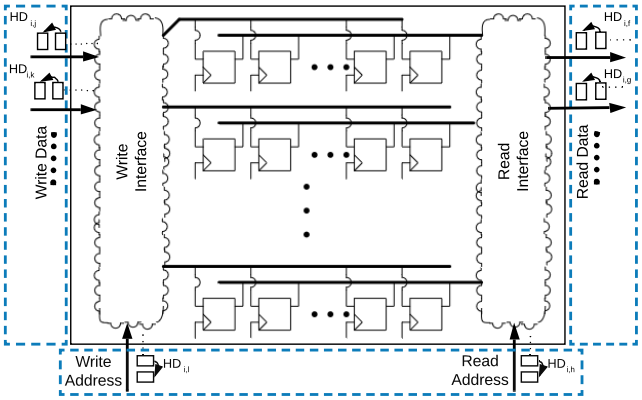


Fig. 8. Buffer modeling.

order buffer. Even though the underlying sequential buffering logic is similar (varying only in bit width and partition), these blocks have widely different read and write characteristics—from indexed to fully associative read/write and from partial data being processed on read to full data being used. Also, the availability of corresponding signal features in high-level simulations varies among blocks. For instance, addresses to a physical register file that buffers the operand data can differ between high-level model and implementation due to equivalent but different renaming implementations. Although an ideal approach is to use all the inputs (data and addresses) as features, we limit ourselves to features available in a typical high-level simulation.

4.2.2 *Pipeline modeling*

CPU blocks, such as functional units, can be internally pipelined, where the number of pipeline stages is typically already decided at the micro-architecture level for most blocks. Intuitively, the power consumption of such pipelined structures depends not only on the current input to a block but also on the partially processed input stored in internal pipeline registers. However, a high-level simulator will typically not accurately model the entire pipeline of a sub-block. It may therefore not contain enough information for developing a stage-by-stage power model for such internally pipelined micro-architecture blocks.

To model internally pipelined structures, we instead store and use the *history* of the last D inputs $HD_{i,j}, HD_{i-1,j}, \ldots, HD_{i-D-1,j}$ and, in case of control inputs, $V_{i,j}, V_{i-1,j}, \ldots, V_{i-D-1,j}$ as features [14], where D is the internal pipeline depth. The activity of internal pipeline registers and hence the power of internal stages will be correlated to previous primary inputs of a block depending on the depth of the pipeline. By giving the input history as features, the pipeline behavior and the impact on power consumption can be learned together. Note that our modeling of power at the micro-architectural sub-block level inherently captures behavior and activity of different concurrent execution units and superscalar pipeline stages at the whole CPU level.

4.2.3 *Data and clock gating*

Data and clock gating are standard techniques to prevent sub-circuits from unnecessary toggling when the output is not needed. Although efficient as power-saving techniques, they present challenges and significantly affect the ability of high-level models to capture underlying circuit behavior.

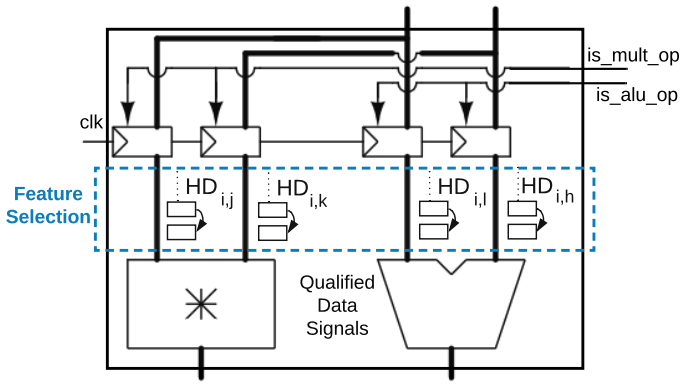


Fig. 9. Data gating.

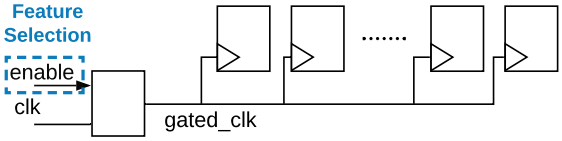


Fig. 10. Clock gating.

In case of data gating, instead of muxing unneeded outputs, the enable signals of input latches to combinational blocks are disabled such that whole sub-block will not toggle. Common examples include splitting the input paths of different execution units in a multi-function ALU and gating specific paths based on the incoming micro-op, as shown in Figure 9. The power consumption of such a gated circuit is dependent on the toggling of the gated or control-qualified signals rather than the primary inputs. We use the history of data applied to a gated path and calculate byte-wise hamming distance per path based on the stream of actively applied control-qualified inputs. This provides improved accuracy compared to a default approach that uses unqualified data in combination with enable signals as control features.

Data gating can be implemented by gating the clock that feeds the input data latches. However, clock gating [36] is a general low-level circuit design technique that synthesis tools can automatically apply to any buffer to save clock tree power. Different clock gaters with complicated conditions, as shown in Figure 10, increase the variance in power consumption and thus make it challenging to model power at a high level. The degree of power variance due to the clock being switched between gated and ungated state depends on its load. We propose the following approach to model combinational clock gating, commonly inserted during logic synthesis, on heavily populated buffer blocks (blocks with only a buffering attribute). These buffer blocks typically have certain micro-architecturally visible conditions that determine the usage of some segment or the whole of the buffer. For example, a branch snapshot buffer might only be needed an incoming branch is mapped to the specific branch tag. Such conditions can be formulated as control signal features for modeling these clock gated buffer units. We recognize the complexity in modeling generic sequential clock gating circuits and leave this for future work.

4.3 Hierarchical Model Composition

We compose sub-block models into models at the super-block and whole core level in a hierarchical fashion. In case of multi-core processors, our approach can be applied to each sub-component

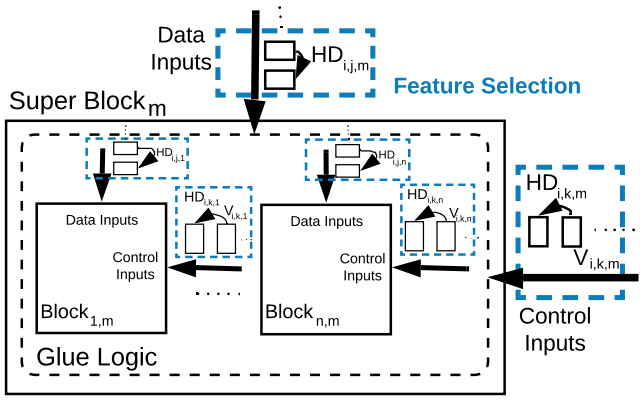


Fig. 11. Hierarchical power modeling.

in each core and then hierarchically composed at the core and multi-core level. There are two possible approaches for hierarchical composition: (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. Although 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 present in the super-block not being modeled.

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 is to treat the glue logic as a virtual block and synthesize a separate power model for it, as shown in Figure 11. This will allow selecting a smaller set of features and simpler model for the glue logic than what would be required to model the complete super-block. Training the glue logic model is achieved by subtracting the sum of component powers from the total super-block 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 and model composition with and without glue logic will be evaluated in Section 5.

5 Experiments

We evaluated our approach by modeling the power consumption of two RISC-V processors, a simple in-order core and a superscalar out-of-order core. Specifically, we model the open-source RI5CY core that is part of the PULP platform [37] developed at ETH Zurich and the University of Bologna, and the Berkeley Out-of-Order Machine (BOOM), an open source RISC-V implementation of an out-of-order processor [38] in its Medium configuration. Our complete power modeling setup including pre-trained models for both RISC-V cores are available in open-source form at [16].

The RI5CY RTL and Chisel-generated BOOM RTL were synthesized with the Nangate 45nm PDK [39] using Synopsys Design compiler (L-2016.03-SP5). Due to the lack of memory generators, buffers in both cores were synthesized as flip-flop arrays. The clock tree was synthesized and the gate-level netlist was placed, routed and back-annotated with physical design information using

TABLE 1
RI5CY training benchmarks.

Test	Description	Cycles
aes_cbc	Small code version of AES	77,332
conv2d	2D convolution	17,713
fdctfst	From ffmpeg libavcodec/jfdctfst.c	4,863
fft	Fast fourier transform	112,370
fir	10 Coefficient FIR filter	48,757
keccak	Sha3 baseline implementation	607,795
matmul	Matrix multiplication	660,901

TABLE 2
BOOM training benchmarks.

Test	Description	Cycles
dhrystone	Synthetic embedded integer benchmark	242,630
mm	Floating-point matrix multiply	289,521
multiply	Software implementation of multiply	70,134
median	1D three-element median	36,735
vvadd	Vector-vector add	21,794
towers	Recursive Towers of Hanoi	22,094
spmv	Sparse matrix-vector multiply	151,015
qsort	Quick sort	450,577

Cadence Innovus (16.13-s045) for the BOOM core. Synopsys VCS (M-2017.03-SP2) was used for running zero-delay gate-level simulations at 25MHz and 333Mhz for the RI5CY and BOOM cores, respectively. The golden reference traces of cycle-by-cycle power used for training and validation were obtained using PrimeTime PX running in time-based power mode. The Scikit-learn [40] python package is used for model synthesis and prediction.

For both cores, we used the official test suites provided with each core to train models. Seven benchmarks from the Pulpino test suite [41] listed in Table 1 were chosen for training and cross-validation of the RI5CY core models. The benchmarks are compiled using the riscv-gnu-toolchain, and object code is used for the simulations. For training and validating of the BOOM core power models, we use 8 micro-benchmarks from the riscv-tests suite [42] summarized in Table 2. In addition, we test our trained model on one iteration of the CoreMark benchmark suite [43] and the FFT benchmark taken from [44], summarized in Table 3. In our experiments, we focus on running baremetal simulations and treat the address translation related logic, such as the TLB, as overhead power as there are no actual translations involved. Note that CoreMark was designed to represent complex real-world mobile workloads combining different functionalities and application kernels into one benchmark. Since power models are ultimately driven by activity information, as long as test cases cover behavior that is representative of the diversity in cycle-by-cycle activity, the accuracy evaluation and results should generalize to a broad range of applications.

Table 4 and Table 5 summarize how modeling concepts are applied to different blocks in each core. In our experiments, we model the glue portion of the core, which is dominated by routing structures, by selecting the data signals from the instruction and data cache interfaces as features. Although our cycle-accurate model is generated from RTL, we include only micro-architecturally invariant signals as features. For example, we model the physical register file with only the data signals as the rename tags can vary between RTL and cycle-accurate micro-architectural models.

TABLE 3
BOOM test benchmarks.

Test	Description	Cycles
CoreMark	CPU benchmark by EEMBC	475,842
FFT	Fast Fourier Transform	626,268

TABLE 4
Power modeling features of RI5CY blocks.

Block	Data	Control	Gating	Buffer	Pipeline
Fetch_stage	✓			✓	
Decode_stage	✓	✓		✓	
Execute_stage	✓	✓			
LS_unit	✓				
CSR				✓	
Pmp_unit	✓				
Glue logic	✓				

TABLE 5
Power modeling features of BOOM blocks.

Block	Data	Control	Gating	Buffer	Pipeline
Fetch controller (FC)	✓	✓		✓	✓
Branch Targ. Buff. (BTB)				✓	
Branch Predictor (BPD)				✓	
Decode unit - 0 (DEC0)	✓	✓			
Decode unit - 1 (DEC1)	✓	✓			
Rename Maptable (RNM)			✓	✓	
Rename Freelist (RNF)			✓	✓	
FP Maptable (FP_RNM)			✓	✓	
FP Freelist (FP_RNF)			✓	✓	
Issue unit (ISS)				✓	
Mem issue unit (M_ISS)				✓	
Iregister file (IRF)				✓	
Iregister read (IRR)	✓				
CSR				✓	
ALU	✓		✓		✓
CSR Exe Unit (CSRX)	✓		✓		✓
FP Pipeline (FP)	✓		✓	✓	✓
LSU	✓	✓		✓	
ROB		✓		✓	
Glue logic	✓				

Table 6 and Table 7 lists the major features selected by the decision tree for each core arranged in ascending order of importance, where 'X' denotes the value of signal X in the current cycle, 'DEL(X)' denotes the value of signal X in the previous cycle and 'HD(X)' denotes the hamming distance between values in the current and previous cycle of X. The normalized importance of each feature in a decision tree based power model is shown in brackets next to the feature. Such a feature ranking can convey additional information about the power behavior to drive power optimizations. Further details, including the full list of features selected for modeling of different blocks as well as their importance ranking for different learned models are available in [45]. In addition, graphical plots of all decision trees are available in [16].

5.1 Model Selection and Cross-Validation

We evaluated the overall accuracy of our models using 10-fold cross-validation on cumulative shuffled data samples collected from the micro-benchmarks. We use mean absolute error (MAE) of average power values predicted by each model at different granularities m compared to gate-level power estimation, normalized to mean reference power of the block as our evaluation metric:

$$MAE[\%] = \frac{\sum_{i=0}^{\lceil \frac{n}{m} \rceil - 1} \left| \frac{1}{m} \sum_{j=im}^{im+m-1} \hat{P}(j) - \frac{1}{m} \sum_{j=im}^{im+m-1} P(j) \right|}{\lceil n/m \rceil \cdot \frac{1}{n} \sum_{i=0}^{n-1} P(i)}, \quad (1)$$

where n is the total number of samples in the trace, m is the sample interval granularity used for prediction, $\hat{P}(i)$ is the predicted power and $P(i)$ is the reference power estimated by PrimeTime PX in the i^{th} sample. Unless otherwise noted, we report accuracy as cycle-by-cycle MAE with $m = 1$ in the remainder of this paper.

TABLE 6
Top decision tree features for different RI5CY 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)
Glue logic	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 7
Top decision tree features for different BOOM blocks.

Block	Features (Importances)
FC	HD(bchecker.io_br_targs_0[7:0]) (0.766), HD(fb.io_enq_bits_exp_insts_0[23:16]) (0.082), HD(BranchDecode 1.io_inst[7:0]) (0.027), HD(fb.io_enq_bits_exp_insts_0[15:8]) (0.026), HD(bchecker.io_btb_resp_bits_target[7:0]) (0.019)
BTB	HD(btb.btb_data_array.RW0_addr[5:0]) (0.57), HD(bim.bim_data_array_1.RW0_addr[8:0]) (0.21), HD(bim.bim_data_array_0.RW0_addr[8:0]) (0.13), bim.bim_data_array_0.RW0_en (0.02), HD(btb.btb_data_array_1.RW0_wdata_target[7:0]) (0.02)
BPD	HD(counter_table.d_W0_data_counter[1:0]) (0.87), HD(counter_table.d_W0_data_cfi_idx[1:0]) (0.13)
DEC0	HD(io_enq_uop_inst[6:0]) (0.8), HD(io_enq_uop_inst[31:25]) (0.18), HD(io_enq_uop_inst[24:20]) (0.02)
DEC1	HD(io_enq_uop_inst[6:0]) (0.8), HD(io_enq_uop_inst[31:25]) (0.16), HD(io_enq uop_inst[24:20]) (0.02), HD(io_enq_uop_inst[11:7]) (0.02)
RNM	DEL(io_ren_br_tags_0_valid) (0.41), io_ren_br_tags_0_valid (0.4), DEL(io_remap_reqs0 valid) (0.08), DEL(io_brinfo_mispredict) (0.05)
RNF	DEL(io_reqs_0) (0.5), DEL(io_reqs_1) (0.28), DEL(io_ren_br_tags_0_valid) (0.16), DEL(io_ren_br_tags_1_valid) (0.04)
FPM	DEL(io_ren_br_tags_0_valid) (0.43), io_ren_br_tags_0_valid (0.37), DEL(io_remap_reqs_1_valid) (0.06), DEL(io_ren_br_tags_1_valid) (0.05), DEL(io_brinfo_mispredict) (0.04)
FPF	DEL(io_reqs_1) (0.41), DEL(io_ren_br_tags_0_valid) (0.23), DEL(io_ren_br_tags_1_valid) (0.23), DEL(io_reqs_0) (0.12), DEL(io_dealloc_pregs_1_valid) (0.02)
ISS	HD(slots_15.slot_uop_uopc[8:0]) (0.24), HD(slots_15.state[1:0]) (0.19), HD(slots_10.state[1:0]) (0.12), HD(slots_7.state[1:0]) (0.06), HD(slots_12.state[1:0]) (0.06), HD(slots_14.state[1:0]) (0.06)
M_ISS	HD(slots 12.state[1:0]) (0.25), HD(slots 15.state[1:0]) (0.17), HD(slots 11.state[1:0]) (0.12), HD(slots_7.state[1:0]) (0.07), HD(slots_3.state[1:0]) (0.06), HD(slots_13.state[1:0]) (0.06), HD(slots_14.state[1:0]) (0.05)
IRF	HD(io_write_ports_0_bits_data[47:40]) (0.78), HD(io_write_ports_0_bits_data[63:56]) (0.11), HD(io_write_ports_0_bits_data[23:16]) (0.03), HD(io_write_ports_1_bits_data[15:8]) (0.03)
IRR	HD(io rf_read_ports_3_data[7:0]) (0.6), HD(io_bypass_data_0[7:0]) (0.12), HD(io_rf_read_ports_1_data[7:0]) (0.05), HD(io_rf_read_ports_2_data[7:0]) (0.05), HD(io_rf_read_ports_5_data[7:0]) (0.05)
CSR	HD(io_rw_wdata[15:8]) (0.67), HD(io_decode_0_csr[11:0]) (0.15), HD(io_rw_wdata[55:48]) (0.07), HD(io_rw_wdata[7:0]) (0.06), HD(io_decode_1_csr[11:0]) (0.04)
ALU	HD(alu.io_in1[7:0]) (0.72), HD(alu.io_in2[7:0]) (0.08), HD(alu.io_in1[31:24]) (0.07), HD(imul.inPipe_bits_in1[15:8]) (0.06)
CSRX	HD(alu.io_in2[7:0]) (0.75), HD(alu.io_in2[63:56]) (0.14), HD(div.io_req_bits_in1[7:0]) (0.04)
FP	HD(fpu.dfma.fma.io_c[15:8]) (0.83), HD(fregfile.io_write_ports_0_bits_data[55:48]) (0.07), HD(fpu.dfma.fma.io_a[47:40]) (0.03)
LSU	HD(io_exe_resp_bits_addr[39:0]) (0.56), io_memresp_valid (0.1), io_memresp_bits_is_load (0.06), io_dis_ld_vals_0 (0.05), io_exe_resp_bits_uop_is_load (0.04), io_dis_st_vals_0 (0.03), HD(io_memreq_wdata[7:0]) (0.03)
ROB	io_enq_valids_0 (0.42), HD(io_enq_uops_1_uopc[8:0]) (0.26), HD(io_commit_uops_0_uopc[8:0]) (0.09), io_enq_valids_1 (0.04), HD(io_enq_uops_0_uopc[8:0]) (0.04), io_commit_valids_0 (0.04), io_wb_resps_1_valid (0.03), io_commit_valids_1 (0.03)

Figures 12 and 13 summarize the average cycle-by-cycle MAE across different folds and power models of different RI5CY and BOOM blocks, respectively. In both cases, a model that just predicts average power (Avg) performs poorly. All in all, a decision tree (DT) based power model performs consistently better than linear models as well gradient boosting and random forest based models of equivalent complexity. As also observed in prior work [14], decision tree-based data representations efficiently capture the inherent non-linear but typically discrete power behavior of design blocks. Furthermore, a deeper decision tree is capable of better capturing the non-linear power characteristics of different micro-architectural blocks in CPUs compared to a forest of shallower (random) trees. Our DT-based power models for the complete RI5CY and BOOM cores have an MAE of 2.15% and 2.86%, respectively. This is an order of magnitude better than the 15.54% and 21.83% MAE, respectively, for a model that just predicts average core power.

Among the RI5CY blocks, blocks with low power variance such as CSR can be modeled accurately. However, low power but high variance blocks such as the LS_unit (LSU) show poor accuracy in modeling with the evaluated models. Accurately modeling the power consumption of the blocks in this category requires further study and is 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 LSU is sufficient for gaining high accuracy for the core composed power model.

Among the BOOM core blocks, due to the limitation in modeling the collapsing queue behaviour of issue units and not using the physical tags as features, power models of ISS and M_ISS blocks have a higher MAE of 9.05% and 19%, respectively. By contrast, by only modeling the data interface signals, our power model for the register file can reach an accuracy of 95.16% (MAE of 4.84%). Our analysis of a decision tree based power model for LSU without data and address features shows degradation of accuracy by 11.5% (MAE of 10.125% vs. 21.625% with and without accounting for data dependencies), highlighting the significance of capturing data-dependent power characteristics in cycle-accurate power models.

Figures 14 and 15 further detail the 10-fold cross-validation results of DT-based models for the different sub-blocks of the RI5CY and BOOM core, respectively. RI5CY results show some variation

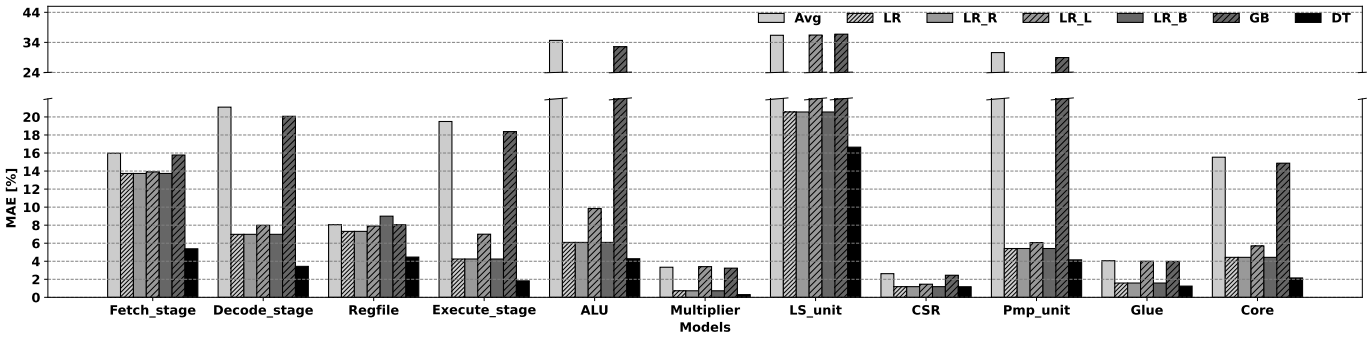


Fig. 12. Cross-validation accuracy of RI5CY block and whole core models.

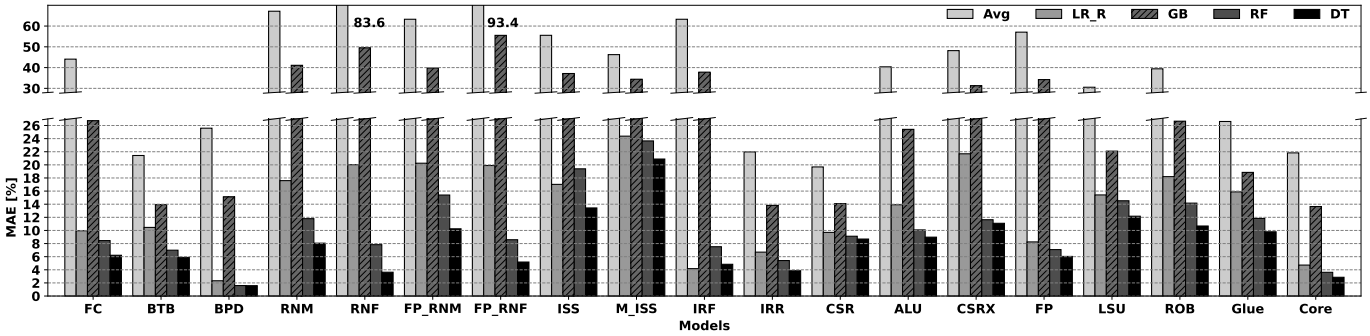


Fig. 13. Cross-validation accuracy of BOOM block and whole core models.

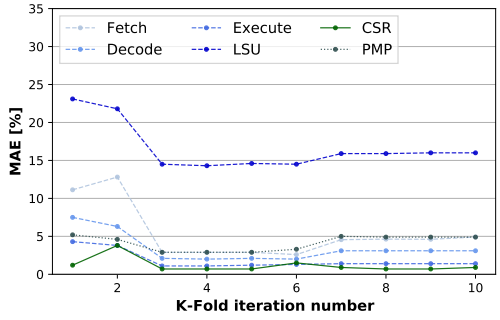


Fig. 14. RI5CY 10-fold cross-validation results.

TABLE 8
Predicted BOOM power statistics.

Benchmark	***CoreMark***			***FFT***		
Block	Avg. Power [mW]	MAE [%]	Avg. Error [%]	Avg. Power [mW]	MAE [%]	Avg. Error [%]
Fetch controller	11.36	10.86	4.48	13.64	10.65	4.49
Branch Targ. Buff.	11.96	7.18	2.17	14.83	7.33	2.18
Branch Predictor	22.4	2.38	2.38	29.34	1.2	2.42
Decode unit - 0	0.51	8.27	0.21	0.54	8.65	0.24
Decode unit - 1	0.52	9.15	0.58	0.52	9.12	0.57
Rename Maptable	3.13	8.51	0.61	3.42	8.58	0.63
Rename Freelist	2.25	3.88	0.19	2.36	5.97	1.25
FP Maptable	2.55	11.44	1.48	2.73	11.31	1.37
FP Freelist	0.63	5.97	1.25	0.68	5.78	1.42
Issue unit	6.19	17.92	6.11	9.03	18.89	7.83
Mem issue unit	3.26	22.84	9.46	4.14	23.45	9.67
Iregister file	10.9	9.34	2.2	15.3	9.22	2.22
Iregister read	3.91	7.42	1.19	4.54	7.67	1.23
CSR	0.91	9.1	3.25	0.94	9.18	3.26
ALU	6.96	15.15	3.78	17.73	15.38	3.85
CSR Exe Unit	1.34	24.66	4.6	1.66	25	5
FP Pipeline	9.27	6.14	0.47	15.32	6.24	0.87
LSU	7.72	16.27	1.32	7.92	12.23	3.12
ROB	5.08	13.24	0.09	5.66	13.44	0.56
Core (composed)	117.05	14.14	12.14	163.18	14.98	13.38
Core (w/ Glue logic)	136.01	3.59	0.23	179.13	3.48	1.14

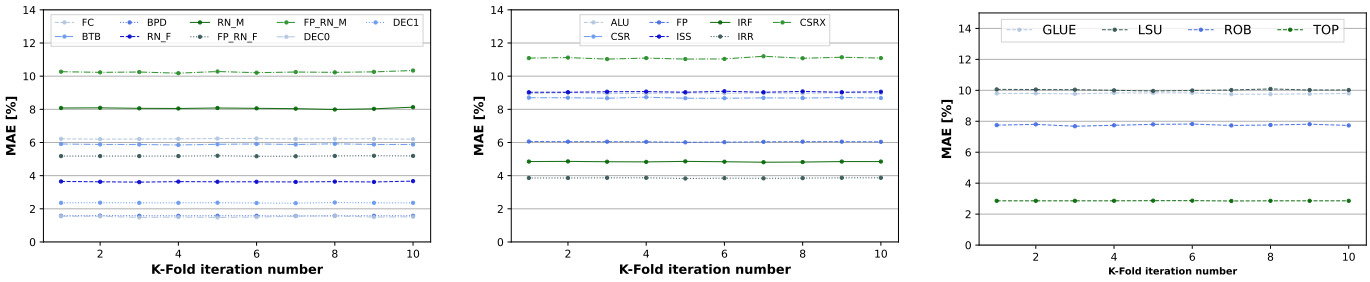
with and sensitivity to training data. By contrast, BOOM results are stable across training folds. This is due to the overall faster learning rate of the hierarchically composed BOOM power model (see Section 5.3).

5.2 Testing Accuracy

We further validate our DT-based BOOM power models by training them on the riscv-test micro-benchmarks and evaluating the trained model on the CoreMark and FFT benchmark tests representing real-world applications.

Table 8 summarizes accuracy for different micro-architectural blocks and the whole core in terms of cycle-by-cycle MAE ($m = 1$) as well as absolute error in predicting average power over the whole trace ($m = n$). The table lists hierarchically composed core power both with and without accounting for glue logic. Our hierarchically composed core models can predict cycle-by-cycle and average power with less than 3.6% and 1.2% error, respectively. By contrast, a composed model without accounting for glue logic only achieves a 14% MAE and 12% average error. Power models for the ALU and CSRX blocks have degraded accuracy when evaluated on an unseen workload, which we attribute to the training sets' incomplete coverage in the data input space. Preliminary results obtained by selectively including around 50 cycles from the CoreMark test set in the training set show an accuracy improvement of up to 2% for these blocks. The study of optimal training sets for all the CPU blocks with good coverage of the signal activity space is left for future work.

Figure 16 further detail the histograms of cycle-by-cycle baseline versus predicted power as well as absolute prediction errors of the whole core when executing CoreMark and FFT benchmarks. . As shown, the corresponding baseline and predicted power histograms have similar distributions with small differences in the number of occurrences. Error histograms further put that in context, showing



(a) Frontend and Rename (b) Issue and execute (c) LSU, ROB and Core

Fig. 15. BOOM 10-fold cross-validation results.

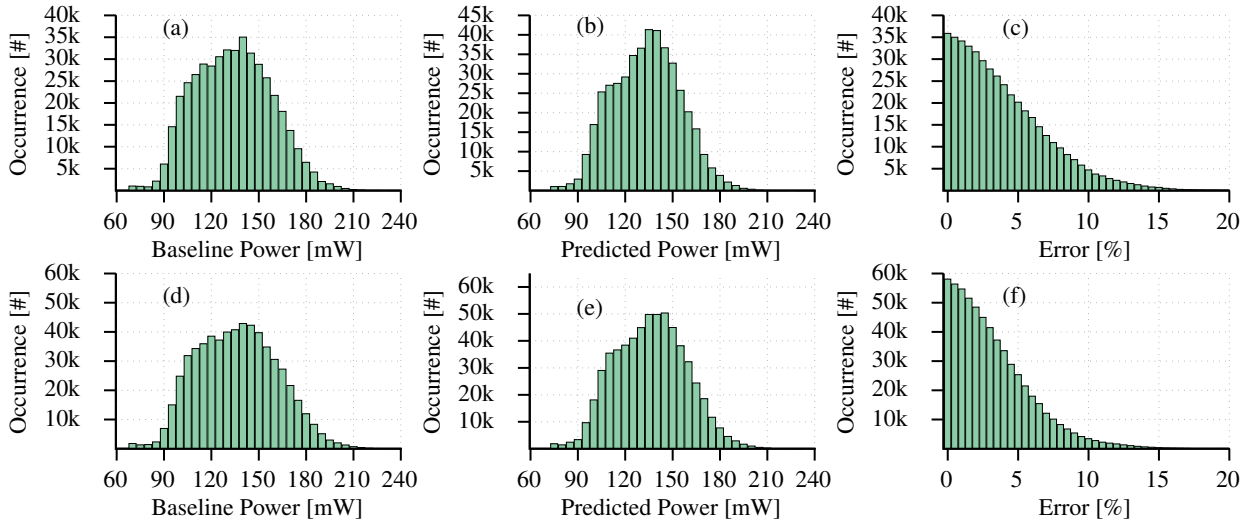


Fig. 16. BOOM core power and error histograms for (a)/(b)/(c) CoreMark and (d)/(e)/(f) FFT benchmarks.

that the majority of the error values are smaller than 5%.

Our models can be used to predict not only cycle-accurate but also the average power of different micro-architectural blocks and the whole core at larger temporal granularities. Average power at coarser sampling rates is often sufficient for many applications such as temperature or hotspot analysis, where accuracy of power prediction is significantly higher when averaging. Figure 17 shows the MAE of the predicted versus baseline average power of different micro-architectural blocks and the whole core for the CoreMark and FFT benchmarks using different averaging and prediction granularities m. An interval size of $m = 1$ represents cycle-by-cycle power estimation. As shown, MAE decreases exponentially with increasing prediction granularity, where our models can efficiently predict average power with a very small error of less than 1% MAE on average already for a modest interval size of $m = 200$k cycles.

Finally, Figure 18 shows a comparison of gate-level baseline versus predicted power traces of CoreMark and FFT benchmarks averaged at a granularity of $m = 100$ cycles. Furthermore, the figures show a zoomed-in view of cycle-by-cycle power for a region around the cycle with peak power consumption. As shown, for both benchmarks, predicted power tracks baseline power accurately. Specifically, our model can predict the exact cycle and hence instruction sequence that triggers peak power with a small error of 4% and 2% in predicting the peak power for the peak of CoreMark and FFT, respectively. Cycle-accurate and peak power estimation is critical for many processor design aspects such as voltage noise or IR drop analysis. Such information has traditionally not been available at higher abstraction levels with existing power models.

All in all, our hierarchically composed core-level power model can predict the power of the evaluated segment of the CoreMark and the FFT workload to within 3.6% cycle-by-cycle MAE of a gate-level power estimate. In addition, our cycle-accurate power models can give highly accurate power estimates (more than 99% accuracy) when used to predict average power at coarser granularity or across the complete workload segment.

5.3 Training and Prediction Overhead

The main learning overhead is the time required for reference gate-level simulations and cycle-accurate power analysis. Figures 19 and 20 show the training curve and learning rate of the DT-based power model for different blocks of the RI5CY and BOOM cores, respectively. Our RI5CY models are able to learn power behavior with less than 300K cycle-level samples and hence instructions needing to be simulated. By contrast, the hierarchical breakdown into smaller sub-block models each with simple behavioral and feature complexity further improves learning rate of the BOOM power models by a factor of 10x even though the BOOM core in total is much larger than the RI5CY one. Note that the behavior of the training curves depends on the diversity of samples in the training set. The first 20k samples of the BOOM training set represent setup and program loading code. They are nearly identical and thus do not improve accuracy. In all the cases, the BOOM models provide high accuracy power estimates once trained with fewer than 30k cycle-level samples and instructions. These are in contrast to the 2.2M samples and 20h of time required

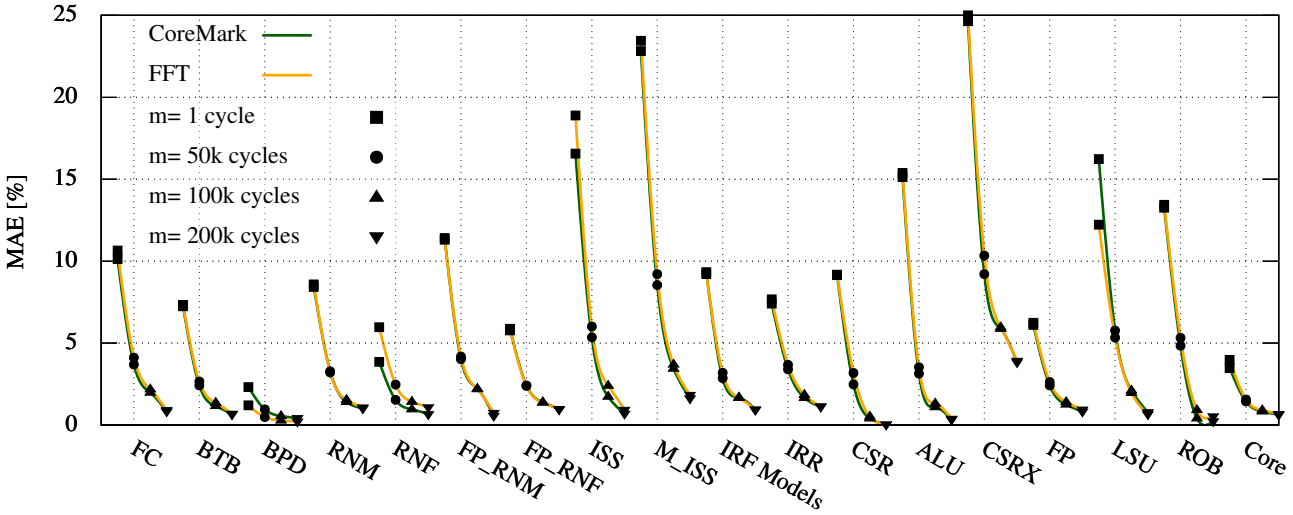


Fig. 17. BOOM block and whole core accuracy for different prediction granularities m.

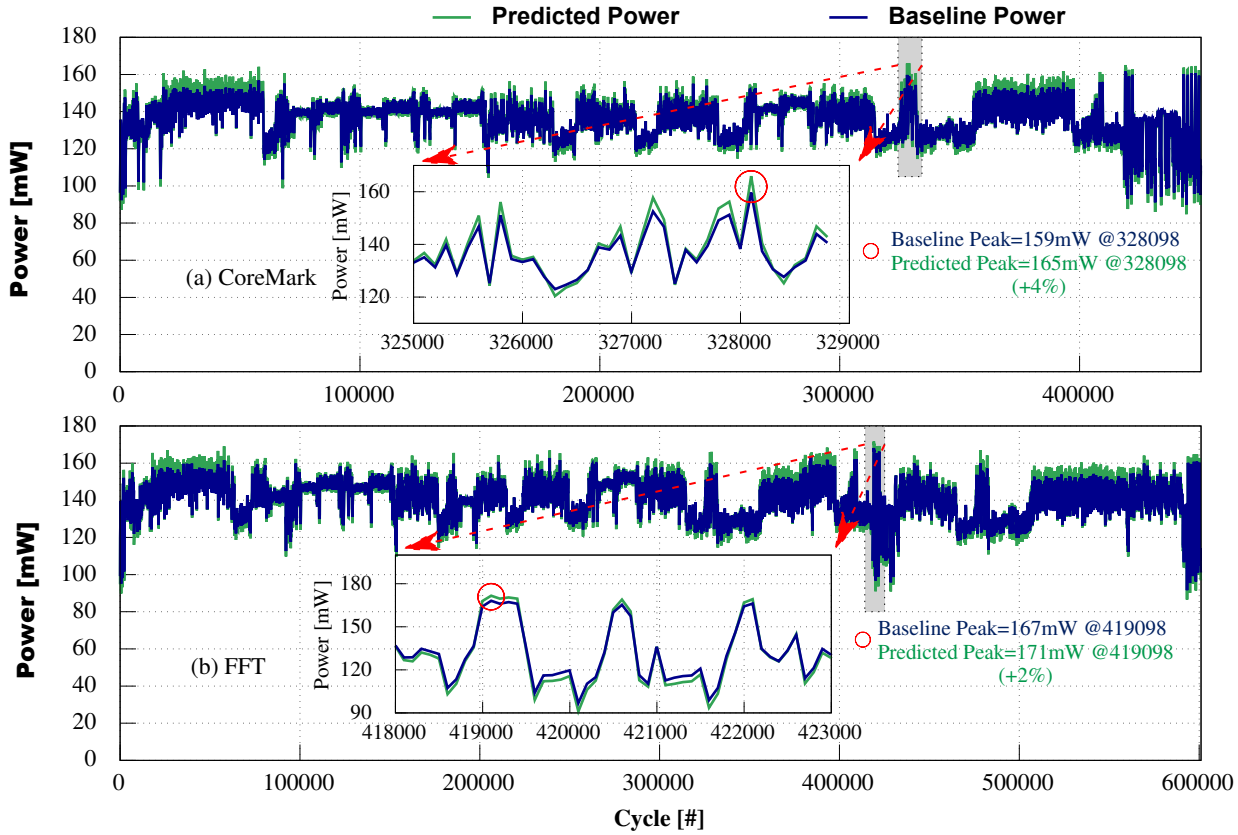


Fig. 18. Predicted and baseline BOOM core power traces for (a) CoreMark and (b) FFT.

to train the DNNs in [7] to provide comparable accuracy for an in-order RISC-V core. Our hierarchical composition and feature engineering methodology enables us to use simple models with low overhead requiring only a small number of time-consuming gate-level simulations for accurate training. In addition, in all cases, decision tree models could be synthesized in less than 5 minutes on an Intel Xeon 8160 workstation running at 2.1GHz. Furthermore, a hierarchical decomposition of models will require only the models of those blocks that are modified to be re-trained when performing micro-architectural studies.

Model evaluations are fast, performing predictions at a rate of 4 Mcycles/s or 3 minutes per block on average, where block models can be evaluated in parallel. As such, prediction is dominated by simulation times to collect activity information. Our Verilator simulations of CoreMark and FFT on the BOOM core require 7-9 hours running at 18 cycles/s, but when coupled with typical cycle-accurate micro-architecture simulators, speeds in the kcycles/s range can be expected. By comparison, a full-featured RTL simulation of the BOOM core requires 13-17 hours running at 10 cycles/s, and an accurate gate-level reference power estimation a total of 40-65 hours running at 2-3 cycles/s.

6 Summary, Conclusions, and Future Work

In this paper, we presented a hierarchical power modeling approach that supports the development of simple, yet accurate power models for CPUs and their internal components at micro-architecture levels

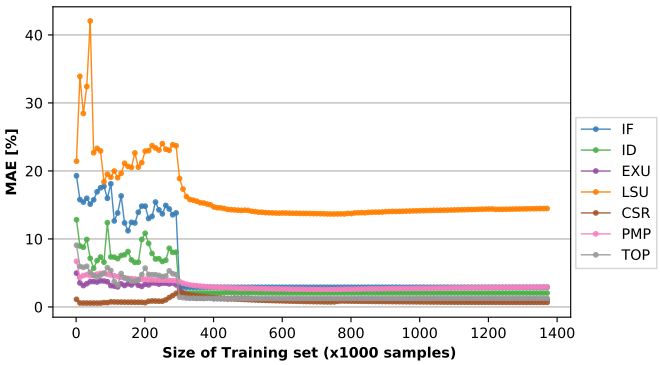


Fig. 19. Learning curve for RI5CY blocks.

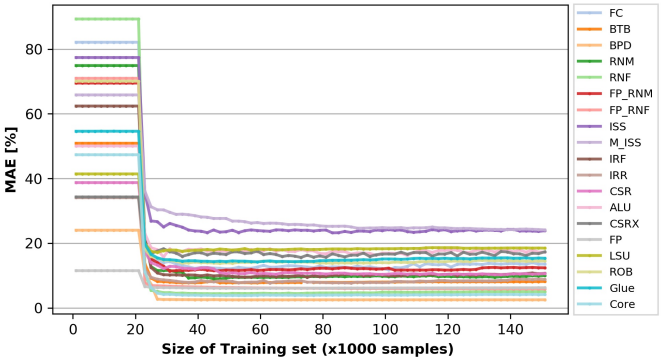


Fig. 20. Learning curves for BOOM core blocks.

of abstractions. We presented a methodology for feature selection and feature engineering that enables using low complexity learning formulations to accurately model common micro-architectural sub-blocks in CPUs. Our core power model, synthesized by integrating these sub-block models, provides cycle-accurate power estimates at sub-block granularity with low training overhead using only features that are extracted from micro-architecture simulations. Results show that a decision tree-based hierarchically composed model, built using our approach, can predict cycle-by-cycle power consumption with less than 2.2% and 2.9% error rate for RI5CY core and BOOM cores, respectively. Resultant BOOM core power models trained on simple micro-benchmarks can further predict cycle-by-cycle power consumption of unseen real-world workloads to within 3.6% of a gate-level power estimate. In future work, we plan to investigate extension to other system components such as inter-connect and approaches for scaling of learnt models to predict across changes in technology or synthesis parameters without the need for re-training. In the future work, we plan to study power models for more complex CISC and multi-core processors including un-core components, models that are parametrized by configuration settings (such as ROB sizes) to increase flexibility, as well as models that only use averaged activity information to predict power at coarser temporal granularity.

Acknowledgments

The authors would like to thank Paul Genssler and Austin Vas from the Chair of Semiconductor Test and Reliability (STAR) for their valuable help with paper revisions. This work was supported in part by the Samsung GRO Program and NSF grant CCF-1763848.

References

[1] Ansys, "PowerArtist," https://www.ansys.com/products/semiconductors/ansys-powerartist.

[2] Mentor, "PowerPro RTL Low-Power," https://www.mentor.com/hls-lp/powerpro-rtl-low-power/.

[3] A. Bogliolo, L. Benini, and G. De Micheli, "Regression-based RTL power modeling," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 5, no. 3, Feb. 2000.

[4] D. Sunwoo, G. Y. Wu, N. A. Patil, and D. Chiou, "PrEsto: An FPGA-accelerated power estimation methodology for complex systems," in *International Conference on Field Programmable Logic and Applications (FPL)*, Aug. 2010.

[5] J. Yang, L. Ma, K. Zhao, Y. Cai, and T.-F. Ngai, "Early stage real-time SoC power estimation using RTL instrumentation," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, Jan. 2015.

[6] Z. Xie, X. Xu, M. Walker, J. Knebel, K. Palaniswamy, N. Hebert, J. Hu, H. Yang, Y. Chen, and S. Das, "APOLLO: An automated power modeling framework for runtime power introspection in high-volume commercial microprocessors," in *International Symposium on Microarchitecture (MICRO)*, Oct. 2021.

[7] Y. Zhou, H. Ren, Y. Zhang, B. Keller, B. Khailany, and Z. Zhang, "PRIMAL: Power inference using machine learning," in *Design Automation Conference (DAC)*, Jun. 2019.

[8] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "The McPAT framework for multicore and manycore architectures: Simultaneously modeling power, area, and timing," *ACM Transaction on Architecture Code Optimization (TACO)*, vol. 10, no. 1, Apr. 2013.

[9] S. L. Xi, H. Jacobson, P. Bose, G.-Y. Wei, and D. Brooks, "Quantifying sources of error in McPAT and potential impacts on architectural studies," in *International Symposium on High Performance Computer Architecture (HPCA)*, Feb. 2015.

[10] W. Lee, Y. Kim, J. H. Ryoo, D. Sunwoo, A. Gerstlauer, and L. K. John, "PowerTrain: A learning-based calibration of McPAT power models," in *IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, Jul. 2015.

[11] M. LeBeane, J. H. Ryoo, R. Panda, and L. K. John, "Watt Watcher: Fine-grained power estimation for emerging workloads," in *International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, Oct. 2015.

[12] Y.-H. Park, S. Pasricha, F. J. Kurdahi, and N. Dutt, "A multi-granularity power modeling methodology for embedded processors," *IEEE Transactions on Very Large Scale Integration Systems (TVLSI)*, vol. 19, no. 4, Apr. 2011.

[13] D. Brooks, P. Bose, V. Srinivasan, M. K. Gschwind, P. G. Emma, and M. G. Rosenfield, "New methodology for early-stage, microarchitecture-level power-performance analysis of microprocessors," *IBM Journal of Research and Development*, vol. 47, no. 5.6, Sep. 2003.

[14] D. Lee and A. Gerstlauer, "Learning-based, fine-grain power modeling of system-level hardware IPs," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 23, no. 3, Feb. 2018.

[15] A. K. Ananda Kumar and A. Gerstlauer, "Learning-based CPU power modeling," in *ACM/IEEE Workshop on Machine Learning for CAD (MLCAD)*, Sep. 2019.

[16] "Learning-based architecture-level CPU power modeling (LACPo)," https://github.com/SLAM-Lab/LACPo.

[17] Synopsys, "PrimeTime," https://www.synopsys.com/implementation-and-signoff/signoff/primetime.html.

[18] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A framework for architectural-level power analysis and optimizations," in *International Symposium on Computer Architecture (ISCA)*, 2000.

[19] H. Jacobson, A. Buyuktosunoglu, P. Bose, E. Acar, and R. Eickemeyer, "Abstraction and microarchitecture scaling in early-stage power modeling," in *International Symposium on High Performance Computer Architecture (HPCA)*, Feb. 2011.

[20] S. Van den Steen, S. De Pestel, M. Mechri, S. Eyerman, T. Carlson, D. Black-Schaffer, E. Hagersten, and L. Eeckhout, "Micro-architecture independent analytical processor performance and power modeling," in *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2015, pp. 32–41.

[21] S. Van den Steen, S. Eyerman, S. De Pestel, M. Mechri, T. E. Carlson, D. Black-Schaffer, E. Hagersten, and L. Eeckhout, "Analytical processor performance and power modeling using micro-architecture independent characteristics," *IEEE Transactions on Computers*, vol. 65, no. 12, pp. 3537–3551, 2016.

[22] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The Gem5 simulator," *ACM SIGARCH Computer Architecture News (CAN)*, Aug. 2011.

[23] J. Zhai, C. Bai, B. Zhu, Y. Cai, Q. Zhou, and B. Yu, "McPAT-Calib: A RISC-V BOOM microarchitecture power modeling framework," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, pp. 1–1, 2022.

[24] D. Kim, J. Zhao, J. Bachrach, and K. Asanović, "Simmani: Runtime power modeling for arbitrary RTL with automatic signal selection," in *International Symposium on Microarchitecture (MICRO)*, Oct. 2019.

[25] W. L. Bircher and L. K. John, "Complete system power estimation: A trickle-down approach based on performance events," in *International Symposium on Performance Analysis of Systems Software (ISPASS)*, Apr. 2007.

[26] T. Li and L. K. John, "Run-time modeling and estimation of operating system power consumption," *SIGMETRICS Perform. Eval. Rev.*, vol. 31, no. 1, Jun. 2003.

[27] X. Zheng, L. K. John, and A. Gerstlauer, "LACross: Learning-based analytical cross-platform performance and power prediction," *International Journal of Parallel Programming*, vol. 45, no. 6, Dec. 2017.

[28] M. Sagi, N. A. V. Doan, M. Rapp, T. Wild, J. Henkel, and A. Herkersdorf, "A lightweight nonlinear methodology to accurately model multicore processor power," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, pp. 3152–3164, 2020.

[29] Y. Zhang, H. Ren, and B. Khailany, "GRANNITE: Graph neural network inference for transferable power estimation," in *Design Automation Conference (DAC)*, 2020.

[30] Verilator, https://www.veripool.org/wiki/verilator.

[31] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, "CACTI 6.0: A tool to model large caches," HP Laboratories, Tech. Rep. HPL-2009-85, 2009.

[32] A. Patel, F. Afram, S. Chen, and K. Ghose, "MARSS: a full system simulator for multicore x86 CPUs," in *Design Automation Conference (DAC)*, Jun. 2011.

[33] D. Burger and T. M. Austin, "The SimpleScalar tool set, version 2.0," *ACM SIGARCH computer architecture news (CAN)*, Jun. 1997.

[34] D. Lee, L. K. John, and A. Gerstlauer, "Dynamic power and performance back-annotation for fast and accurate functional hardware simulation," in *Design, Automation and Test in Europe (DATE)*, Mar. 2015.

[35] D. Lee, T. Kim, K. Han, Y. Hoskote, L. K. John, and A. Gerstlauer, "Learning-based power modeling of system-level black-box IPs," in *International Conference on Computer-Aided Design (ICCAD)*, Nov. 2015.

[36] M. Donno, A. Ivaldi, L. Benini, and E. Macii, "Clock-tree power optimization based on RTL clock-gating," in *Design Automation Conference (DAC)*, Jun. 2003.

[37] D. Rossi, F. Conti, A. Marongiu, A. Pullini, I. Loi, M. Gautschi, G. Tagliavini, A. Capotondi, P. Flatresse, and L. Benini, "PULP: A parallel ultra low power platform for next generation IoT applications," in *IEEE Hot Chips 27 Symposium (HCS)*, Aug. 2015.

[38] C. Celio, P.-F. Chiu, B. Nikolic, D. A. Patterson, and K. Asanović, "BOOM v2: an open-source out-of-order RISC-V core," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2017-157, Sep. 2017.

[39] Silvaco, "Silvaco FreePDK45 Open Cell Library," http://www.silvaco.com.

[40] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research (JMLR)*, vol. 12, 2011.

[41] PULP platform, "Pulpino test suite," https://github.com/pulp-platform/pulpino/tree/master/sw/apps/bench.

[42] RISC-V Tests, https://github.com/riscv/riscv-tests, RISCV Foundation.

[43] S. Gal-On and M. Levy, "Exploring CoreMark — a benchmark maximizing simplicity and efficacy," *The Embedded Microprocessor Benchmark Consortium (EEMBC)*, 2012.

[44] "Risc-v emulators for parallax propeller and propeller 2," https://github.com/totalspectrum/riscvemu, Tech. Rep., 4 2021.

[45] A. K. Ananda Kumar and A. Gerstlauer, "Learning-based architecture-level power modeling of CPUs," Electrical and Computer Engineering, The University of Texas at Austin, Tech. Rep. UT-CERC-20-01, 2020.

Ajay Krishna Ananda Kumar received the B.E. degree in electronics and communication engineering from Anna University, Chennai, India, in 2015 and the M.S. degree in electrical engineering from the University of Texas at Austin, TX, USA in 2020. From 2015 to 2018, he was an SoC Design Engineer at Qualcomm, India. He was a graduate research assistant at The University of Texas at Austin from 2019 to 2020 where he worked on the application of Machine Learning methods for building accurate power models of CPUs to enable power aware micro-architectural design space exploration and focused power optimizations. His research interests include energy-efficient circuit design and CPU/GPU micro-architectural optimizations for power and performance.

Sami Al-Salamin received his B.Sc. degree in computer systems engineering and M.Sc. degree in informatics (first rank) from Palestine Polytechnic University, Hebron, Palestine in 2005 and 2012, respectively. He received his Ph.D. degree in engineering in 2021 from Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany. He is with Palestine Polytechnic University (PPU) since 2005. He is now with Hyperstone®. His main research interests are emerging technologies, low power design, reliability of embedded systems, and machine learning. He holds HiPEAC Paper Award. ORCID 0000-0002-1044-7231

Hussam Amrouch (S'11-M'15) is a Junior Professor heading the Chair of Semiconductor Test and Reliability (STAR) within the Computer Science, Electrical Engineering Faculty at the University of Stuttgart. He received his Ph.D. degree with distinction (Summa cum laude) from the Karlsruhe Institute of Technology in 2015. He was recently appointed as as an Editor in the Nature Portfolio for the Nature Scientific Reports journal. He holds eight HiPEAC Paper Awards and three best paper nominations at top EDA conferences: DAC'16, DAC'17 and DATE'17 for his work on reliability. He has 170+ publications (including 70 journals) in multidisciplinary research areas, starting from semiconductor physics to circuit design all the way up to CAD and computer architecture. He has delivered 10+ tutorial talks in major EDA conferences like DAC, DATE, ICCAD and 27 invited talks (including 2 Keynotes) in several international conferences, universities, and leading EDA companies such as Synopsys.

Andreas Gerstlauer (SM'11) is a Professor of Electrical and Computer Engineering (ECE) at The University of Texas at Austin. He received his Ph.D. degree in Information and Computer Science (ICS) from the University of California, Irvine (UCI) in 2004. Prior to joining UT Austin in 2008, he was an Assistant Researcher in the Center for Embedded Computer Systems (CECS) at UCI. His research interests cover system-level design and embedded systems. He is co-author on 3 books and over 150 publications. His work was recognized with several best paper awards and nominations from major conferences such as DAC, DATE and HOST. He is recipient of a Humboldt Research Fellowship and serves or has served as an Editor for ACM TECS and TODAES journals as well as General or Program Chair for major international conferences such as ESWEEK.