Learning-Based, Fine-Grain Power Modeling of System-Level Hardware IPs

DONGWOOK LEE and ANDREAS GERSTLAUER, University of Texas at Austin, USA

Accurate power and performance models are needed to enable rapid, early system-level analysis and optimization. There is, however, a lack of fast yet fine-grain power models of hardware components at such high levels of abstraction. In this paper, we present novel learning-based approaches for extending fast functional simulation models of accelerators and other hardware intellectual property components (IPs) with accurate cycle-, block-, and invocation-level power estimates. Our proposed power modeling approach is based on annotating functional hardware descriptions with capabilities that, depending on observability, allow capturing data-dependent resource, block, or I/O activity without a significant loss in simulation speed. We further leverage advanced machine learning techniques to synthesize abstract power models using novel decomposition techniques that reduce model complexities and increase estimation accuracy. Results of applying our approach to various industrial-strength design examples show that our power models can predict cycle-, basic block-, and invocation-level power consumption to within 10%, 9%, and 3% of a commercial gate-level power estimation tool, respectively, all while running at several order of magnitude faster speeds of 1-10Mcycles/sec. Model training and synthesis takes less than 34 minutes in all cases, including up to 30 minutes for training data and trace generation using gate-level simulations.

CCS Concepts: • **Hardware → Platform power issues**; **Modeling and parameter extraction**; **Software tools for EDA**; High-level and register-transfer level synthesis;

Additional Key Words and Phrases: Power modeling, Power estimation, Machine learning, Virtual platform, System-level design, High-level synthesis, Hardware accelerator

ACM Reference Format:
Dongwook Lee and Andreas Gerstlauer. 2010. Learning-Based, Fine-Grain Power Modeling of System-Level Hardware IPs. *ACM Trans. Web* 9, 4, Article 39 (March 2010), 25 pages. https://doi.org/0000001.0000001

1 INTRODUCTION

The continued rise in hardware and software complexities of embedded on-chip systems has necessitated raising the design process to higher levels of abstraction. At the same time, energy efficiency has become a critical design concern. Fast and accurate system-level power estimation approaches are needed to drive associated validation and optimization. Virtual platform models capable of simulating whole systems are widely employed to provide rapid feedback for design space exploration. Instead of slow co-simulation with low-level register-transfer level (RTL) or cycle-accurate models of custom hardware intellectual property components (IPs) and processors, a purely functional modeling of hardware and software behavior is typically utilized. However, the modeling gap between fast, purely functional models for integration into virtual platforms and their corresponding low-level hardware implementations makes accurate power modeling challenging.

Authors' address: Dongwook Lee; Andreas Gerstlauer, University of Texas at Austin, Austin, TX, USA.

1559-1131/2010/3-ART39 $15.00
https://doi.org/0000001.0000001

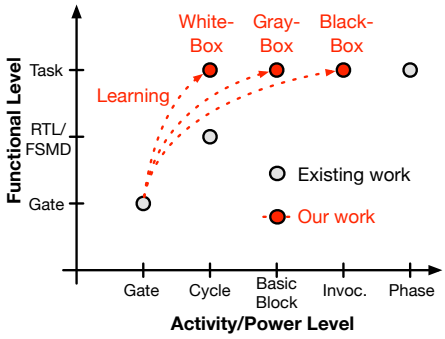


Fig. 1. Power modeling space.

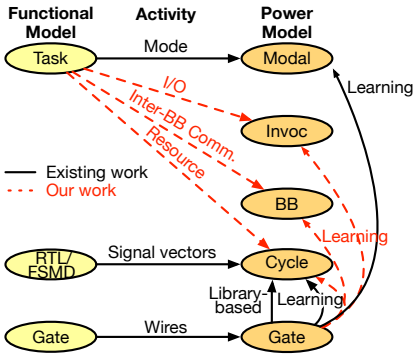


Fig. 2. Power modeling approaches.

Figure 1 classifies power modeling approaches based on the granularity and abstraction level of their functional simulation versus activity and power estimation. At the lowest level, a detailed but expensive simulation of gate-level switching activity is used to estimate gate-level power consumption. Various power modeling approaches at higher levels of abstraction have been proposed. Most previous work at the system level utilizes a fast functional C/C++ task simulation to drive state-based power estimations that only model transitions between different coarse-grain operation modes [Copty et al. 2011; Lee et al. 2006; Lorenz et al. 2014; Schürmans et al. 2015, 2013; Trabelsi et al. 2011]. Other approaches use accurate but slow activity estimation at a finer micro-architecture or RTL granularity. More recently, solutions at the intermediate representation (IR) level have emerged [Chen et al. 2007; Grüttner et al. 2014; Potlapally et al. 2001; Shao et al. 2014; Zhong et al. 2004]. However, they similarly rely on slow, fine-grain simulation of the cycle-by-cycle behavior of individually scheduled IR operations in control/dataflow graph (CDFG) or finite state machine with data (FSMD) form to obtain accurate results.

Existing approaches all estimate power at the same level of detail at which the functionality of hardware is modeled. This allows a detailed functional simulation to drive an accurate, potentially data-dependent power estimation model, but also creates a fundamental trade-off between speed and accuracy depending on the simulation granularity. In this paper, we instead propose novel approaches that enable fine-grain, data-dependent power estimation at the speed of a fast functional simulation. Instead of detailed micro-architecture or FSMD/CDFG simulation, we extend high-level functional simulations with fine-grain, dynamic power modeling capabilities. Depending on the observability of hardware internals and their mapping to high-level constructs, extended white-, gray-, or black-box models are able to capture data-dependent resource, basic block, or external I/O activity, respectively. Extracted activity data is then used to drive corresponding cycle-, block-, or invocation-level power models, where we statically synthesize data-dependent, activity based power models at three different levels from a given gate-level implementation using machine learning approaches.

In previous work, we have developed such learning-based cycle- and invocation-level power modeling approaches for white- and black-box hardware IPs, respectively [Lee et al. 2015a,b]. Our white-box approach integrates with existing high-level synthesis (HLS) tools to automatically extract resource mapping information, which is used to trace resource-level activity and drive a cycle-accurate online power-performance model during functional IR simulation. By contrast, our black-box approach utilizes only external I/O activity captured from a transaction-level simulation to drive an offline invocation-level power model. Both approaches leverage state-of-the-art machine

learning techniques to synthesize abstract power models, where we introduce specialized structural decomposition techniques to reduce model complexities and increase estimation accuracy.

In this paper, we extend our prior work and present a comprehensive and fully automated power modeling framework that provides fast yet accurate learning-based power estimation at three levels of abstraction. We introduce an intermediate gray-box approach that supports power estimation at basic block-level granularity. It utilizes less total switching activity and fewer invocations of the power model than cycle-level models, while providing a finer granularity than invocation-level models, which allows to further navigate accuracy and speed trade-offs. Furthermore, we improve our existing white- and black-box approaches to increase estimation speed.

The specific contributions of this paper are: (1) Using only limited mapping information about basic block inputs and outputs, we develop a light-weight approach for extracting block-level activity from a functional simulation (Section 4.2); (2) We propose a basic block-level power model that utilizes a novel decomposition using control flow information to reduce model complexity while improving estimation accuracy (Section 5.2); (3) We redesign white-box resource-level signal activity computation to improve estimation speed; (4) We adopt an online approach to capture invocation-level I/O activity and thus improve black-box estimation speed; (5) We present additional and larger design examples; and (6) We integrate fast and accurate hardware power models with a GEM5-based full-system simulation to demonstrate benefits of our models for system-level design, virtual platform prototyping and design space exploration (Section 6.6).

The rest of the paper is organized as follows: following a discussion of related work, Section III introduces an overview of our proposed methodology, while Section IV and Section V elaborate on each step in more detail. Section VI shows experimental results of applying the flow to a set of industrial-strength design examples. Finally, Section VII concludes the paper with a summary.

2 RELATED WORK

Figure 2 shows a more detailed taxonomy and overview of existing power modeling work in relation to our approach. Traditional accurate power models are constructed by coupling gate-level simulations with gate library power models. To generate higher-level timing and energy models of custom hardware accelerators and processors, library- or learning-based approaches can be utilized. In a library-based approach, an overall model is assembled from pre-characterized component data [Bogliolo et al. 2000; Chen et al. 2007; Grüttner et al. 2014; Gupta and Najm 2000; Hsieh et al. 1996; Potlapally et al. 2001; Ravi et al. 2003; Shao et al. 2014; Zhong et al. 2004]. This enables rapid exploration but does not accurately account for all glue logic and implementation-level optimizations in a combined architecture. In learning-based approaches, a low-level implementation is simulated in a sampling fashion to derive a regression-based model for a complete processor or each macro-block [Hsu et al. 2011; Park et al. 2007, 2011; Sunwoo et al. 2010; Wu 2015]. Such approaches can accurately reflect the behavior of the final implementation. In the best reported cases, state-of-the-art RTL power models achieve up to 95% cycle accuracy compared to gate-level simulations. However, speed is fundamentally bounded by the abstraction level at which functional simulation is performed. Accurate fine-grain and data-dependent approaches require simulation at the RTL or micro-architecture level to extract internal signal information driving the generated models, which is typically too slow to be integrated into virtual platforms at the system level.

At the system level, component models are often only functionally equivalent ones, where necessary internal architectural information for fine-grain modeling is not available, especially in case of pre-designed IPs. The limited observability of such high-level, black-box models restricts power estimation to a coarse-grain state-based approach, where the projection of either given, documented states [Lee et al. 2006; Trabelsi et al. 2011], or state information estimated from external transaction events [Copty et al. 2011; Lorenz et al. 2014; Schürmans et al. 2015, 2013] only supports

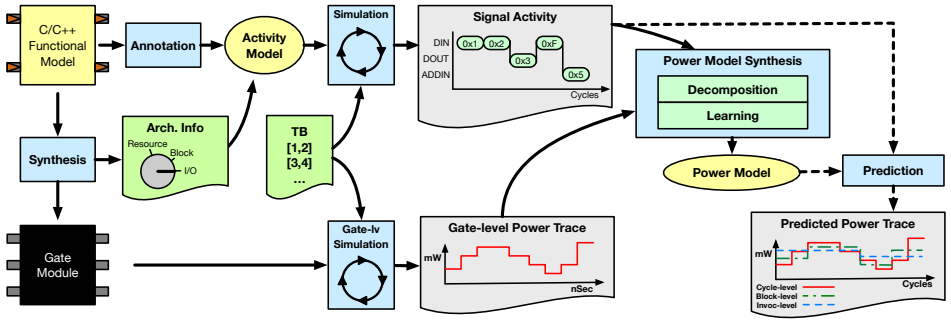


Fig. 3. Power modeling flow.

capturing average power transitions between different operating modes, such as read and write modes in memories or buses. To take into account data-dependent effects in power estimation of black-box components, a corresponding extension of coarse-grain state-based models was recently proposed [Lorenz et al. 2014]. Depending on the hardware being modeled, invocation-by-invocation errors of 1%-17% at 4,000x speedup compared to gate-level simulations are reported. In this approach, cycle-level input switching activity information is utilized to refine states in which significant data-dependent power variations are observed. This requires augmenting state-based models with the ability to capture cycle-by-cycle activity, which introduces a significant overhead in the simulation. Furthermore, a simple linear regression is inherently limited in accuracy.

By contrast, we aim to drive fine-grained, data-dependent power models directly from high-level C/C++ functional simulations. Our approach supports both library-based and learning-based methods, where our focus is on the learning-based generation of lightweight implementation-level representations of complete hardware processors. We propose cycle-, basic block-, and invocation-level power models combined with extraction of resource, inter-basic block communication, and external I/O activity from high-level functional simulations. Compared to [Lorenz et al. 2014], our invocation-level approach only requires capturing I/O activity and instead uses advanced learning methods to improve accuracy.

A key concern in learning-based methods is managing model complexities without sacrificing accuracy. Existing approaches rely on sampling a subset of key signals or state variables that are identified either manually or in a trial-and-error process [Hsu et al. 2011; Sunwoo et al. 2010]. Other approaches decompose the full power model into several parts based on manual decisions [Park et al. 2007]. This requires detailed architectural knowledge or designer insight, which is often not available, especially for black-box IPs. By contrast, we automatically decompose a full power model into several simpler models based on cycle-, block- or purely I/O-specific information, which results in better accuracy while reducing learning and estimation overhead.

For software running on processors, so-called source-level or host-compiled modeling approaches have recently emerged as an alternative to micro-architecture or instruction-set simulation. In such approaches, a source or IR model of the application is statically back-annotated with timing and energy estimates extracted from low-level simulations [Grüttner et al. 2014; Zhao et al. 2017]. The static back-annotation is typically performed at the basic block level, which is only able to capture control-dependent power behavior. Our proposed approach is motivated by host-compiled software models, but also aimed at accurately capturing data-dependent power effects. Instead of back-annotating static per block estimates, we annotate the functional simulation with dynamic, data-dependent cycle-, block- or invocation-level power models. Our approach supports offline or online prediction as part of high-level functional and activity simulations.

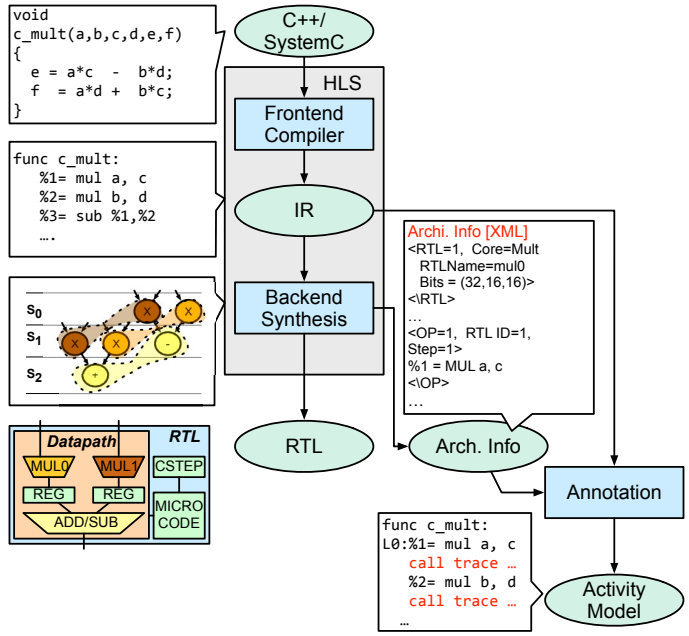


Fig. 4. Activity model generation flow.

3 POWER MODELING FLOW

Figure 3 shows an overview of our proposed power modeling flow. The inputs to the flow are a high-level functional simulation model of a hardware component, its corresponding gate-level implementation, and optional micro-architecture mapping information. Depending on internal observability, architecture information can consist of a complete mapping of high-level operations into RTL states and resources (white-box), the mapping of basic block inputs and outputs to resources and ports (gray-box), or only limited information about the mapping of external I/O (e.g. in case of black-box IPs). Micro-architecture information can be manually provided or automatically extracted during synthesis. In our flow, we integrate with existing, commercial HLS tools to provide a fully automated power model generation for custom hardware synthesized by HLS.

Using white-, gray-, or black-box micro-architecture mapping information, we annotate the high-level functional simulation code with the ability to capture activity traces of individual resource, basic block, or external input and output value transitions. In a training phase, the given gate-level model and the generated activity model are then simulated with the same input vectors. Power synthesis utilizes resource, block, or I/O activity traces from the high-level simulation together with cycle-level power traces from gate-level estimation to learn a power model. Instead of building a single power model, the synthesis flow decomposes power estimation into multiple models and individually trains them. Each decomposed power model is further simplified using a feature selection to reduce the amount of switching information that needs to be collected. In the process, the activity model is also simplified by removing unnecessary signal tracing not utilized after feature selection. In the prediction phase, the synthesized power models are then used to estimate data-dependent cycle-, block-, or invocation-level power traces from corresponding resource, basic block, or external I/O activity captured in high-level simulations.

4 ACTIVITY MODEL GENERATION

The annotation process refines a high-level C/C++ hardware functional model into an activity model. Depending on available architecture information, the refined activity model supports three different levels of switching activity tracing: individual resources, blocks, or only external I/Os. In the following, we first describe the general annotation process followed by each specific level of activity tracing and modeling.

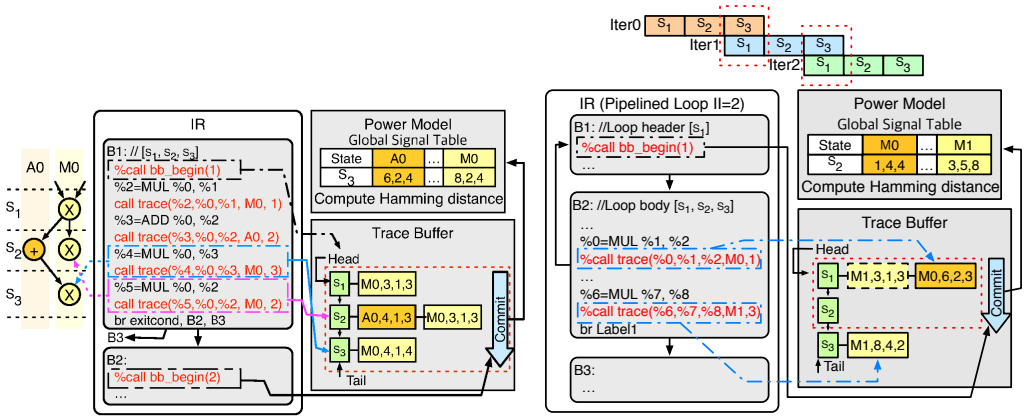
Figure 4 shows an overview of our activity model generation flow, accompanied by representative models and code snippets at various stages. In our framework, we synthesize a given functional hardware model down to an RTL description using a standard HLS process. In the process, we extract the IR of the design generated by the HLS frontend. Working at the IR level allows us to accurately reflect source-level optimizations, such as bit width reductions that affect tracking of internal signals in the synthesized RTL datapath. At the same time, the IR is extracted in C/C++ form before back-end synthesis in the HLS tool, i.e. it remains at a fast functional level. The IR code is further synthesized into an RTL implementation by the HLS tool. In this process, we automatically extract architecture information in the form of an extensible markup language (XML) file that stores mapping information between the IR and the synthesized RTL implementation. Mapping information can be automatically generated during HLS as in our case or optionally manually provided. Depending on observability, it captures the mapping of IR operations to RTL control steps and datapath resources, the mapping of basic block inputs and outputs to resources and ports, or the mapping of functional interfaces to external I/O ports. The annotation process then automatically inserts corresponding signal *trace*() functions to generate an activity model that allows capturing cycle-by-cycle switching activity of individual datapath resources, basic block-by-basic block activity of block input and outputs, or invocation-by-invocation activity of external I/O during functional IR simulation.

4.1 Resource-Level Activity Computation

In a white-box case, we support capturing cycle-accurate activity of RTL datapath resources, such as adders and multipliers, during high-level functional simulation by back-annotating abstract micro-architecture information into the IR. We assume that micro-architecture mapping information is provided, where we can extract an FSMD-level description from the HLS tool. The extracted architecture information includes each IR operation node's resource scheduling, binding, and bit width information. Based on this information, the annotation process inserts *trace*() functions that store the operands and results of each IR operation together with the scheduled control state and bound resource ID to compute switching activity. We capture the flow of data and associated switching activity by tracing IR operands and results. To map data activity into signal transitions of actual hardware resources, we include resource scheduling and binding information in the captured traces. In addition, bit width information is annotated to extract the actual number of bits utilized in hardware. This information is then used to track cycle-by-cycle activity of each resource while taking into account resource sharing and other back-end synthesis optimizations.

The hardware implementation generally exploits operation-level parallelism and scheduling flexibility to maximize performance under given resource or timing constraints. As a result, operators in the IR are not necessarily simulated in the same order in which they execute in the final hardware. Figure 5(a) and Figure 5(b) show such intra- and inter-block level out of order execution scenarios, respectively.

In order to rearrange out-of-order execution traces captured in the IR simulation into in-order traces for hardware estimation, we perform an online reordering of traced information using annotated scheduling and binding information. As shown in Figure 5(a), the execution order of two



(a) Intra basic block (b) Inter basic block

Fig. 5. Operation-level signal trace rearrangement.

operators in the same basic block can be reversed in the hardware implementation if there is no dependency between the operations. To rearrange the trace, we utilize a global signal table and a trace reordering buffer. The global table tracks signal values of all hardware resources in the most recent cycle. The trace buffer temporally stores and reorders signal updates associated with the current basic block. It consists of control state tags and corresponding signal trace lists. Each entry in the signal trace lists contains the utilized resource ID together with operands and result of the operation. At the beginning of each basic block, an additional function is annotated to initialize the buffer and insert state tags corresponding to the block's control states. Within the block, each call to the *trace*() function then attaches a new entry to the signal trace linked list corresponding to the annotated control state. At the end of the current and beginning of the next basic block, all signals updated in each control steps are sequentially committed to the global signal table, the head of the buffer is moved to the tail, all current control step and trace lists are discarded, and new control state tags assigned to the next block are inserted. In this process, the Hamming distances of all signals toggling in each control step are computed, and this switching activity information is committed to either a tracing file or the final power model. In addition, for performance estimation, a global cycle counter is increased by the number of cycles spent on the block.

As shown in Figure 5(b), the execution of basic blocks can be overlapped in the case of pipelined hardware loops. This results in some operators in the second iteration to be executed before the last operator in the first iteration. In [Lee et al. 2015a], we had introduced an additional intermediate pipeline buffer that retains signal traces of previous iterations to emulate the pipeline structure. To improve simulation speed, we now account for such pipeline effects by instead controlling the head and tail management in the trace buffer itself. When first entering the header block of a pipelined loop, control state tags for a single iteration of the loop body block are inserted into the buffer. If a pipelined execution is detected, the trace buffer is not committed during execution of the loop body. Instead, during execution of the header block at the start of each new loop iteration, only the completed control steps, i.e. entries corresponding to the loop initiation interval (II) are committed, and the head is moved and entries are discarded accordingly. Remaining entries are retained and their state tags relabeled to overlap with the start of the next iteration. Finally, new control state tags for the bottom part of the loop body are inserted into the trace buffer. With each such iteration, new traces will be added to the remaining buffer contents, which will contain uncommitted signal

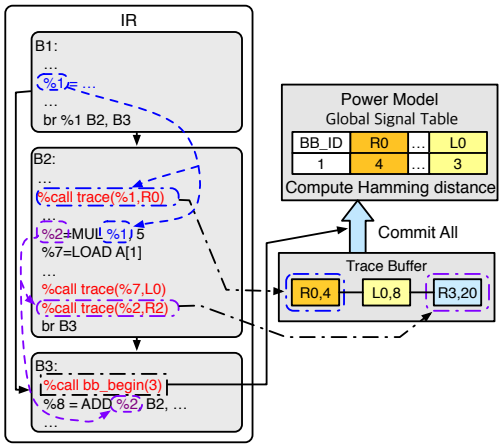


Fig. 6. Basic block-level signal trace rearrangement.

data from previous iterations. After the end of execution of a loop, all remaining entries in the buffer are committed. Loop information (II as well as IDs of all loop header and body blocks) is automatically extracted from the HLS tool together with other scheduling and binding information. Overall, this approach allows us to accurately trace the signal transitions of hardware resources without the need for a slow lockstep pipeline simulation.

4.2 Block-Level Activity Computation

Internal signal switching activity estimation is a key for data-dependent power modeling. Resource-level tracing provides cycle-accurate switching activity of each datapath component, but requires extending the functional model to capture cycle-specific activities, resulting in simulation overhead. Moreover, Hamming distance and switching activity computation for whole resources is typically the most significant bottleneck for power estimation, and it is often much slower than actual functional simulation [Pedram 1997]. Instead of computing cycle-by-cycle switching activity for all resources, we propose a basic block-level model that only utilizes inter-basic block communication, i.e. inputs and outputs of basic blocks for activity and power estimation. This reduces the total amount of signal traces and switching activity that need to be collected and computed, which results in faster estimation speed.

In order to track inter-basic block communication activity in each block, we trace all input variables that are updated in a previous block but read in the current block, all output variables that are written in the current and read in a subsequent block, as well as all block-internal memory accesses. We extract the mapping of each input and output variable and each array access in the basic blocks to corresponding registers and memory ports in the hardware while taking into account register and memory port sharing. The annotation process inserts *trace*() function calls to store the inter-basic block communication traces along with the mapping IDs. Input variables are traced at the beginning of each basic block while output variables and memory access are traced at the end of each block.

Shared memory accesses can be flexibly scheduled during RTL synthesis to maximize hardware performance if there is no dependency between the operations. To compute accurate memory port activity, a reordering is therefore also required, but cycle-accurate reordering is not necessary. Instead of using a reordering buffer, the annotation process statically reorders the shared memory accesses by inserting the trace functions in access order.

Figure 6 shows the block-level activity computation process. Basic block inputs and outputs are collected in the trace buffer at run-time. At a beginning of the basic block, the trace functions attach basic block inputs to the signal trace linked list, along with annotated resource information. At the end of each block, memory accesses and outputs are also attached to the linked list. At the beginning of the next basic block, the whole list of captured traces is committed into the global signal table to compute inter-basic block activity during a single basic block execution. For cycle-accurate performance estimation, the execution time of each basic block is also extracted from the HLS tools and annotated in a similar manner as for resource-level activity tracing.

4.3 External I/O Activity Computation

If no internal architecture information is available, we support black-box power estimation utilizing only external I/O activity captured for each function invocation. This is the case for pre-designed hardware IPs, where only functional simulation models without detailed architecture descriptions are provided together with pre-synthesized gate-level implementations. This approach can also further reduce tracing and computation overhead, but without internal timing information, only supports power modeling at invocation-level granularity.

We assume that I/O interface mapping information between system-level transactions and IP data ports is given. In black-box models, communication interfaces are approximately modeled, and the detailed computation architecture is fully abstracted out. The models are often also purely functional, where no timing information is available. However, even a high-level functional model has interfaces that map to corresponding external I/O ports. In general, we can find such mapping information in documents or test benches for gate level simulation.

Required architecture information only consists of external I/O port mapping, bit width and control port/register information. Designers can manually describe the architecture file to utilize our automated annotation flow or manually insert trace functions into the source code. Both approaches can be seamlessly integrated into the automated power model synthesis process without further manual interventions.

Mapping information and external I/O data are passed into annotated *trace*() calls, which are inserted at the beginning and end of each function. To compute I/O activity, we utilize a similar mechanism as at the basic block-level, but we commit signal traces into the global signal table only at the end of each function invocation. In addition to external data I/O activity, important control registers or control ports are also traced. We assume that such control dependencies are available to model functional or performance behavior. The activities of control signals, such as mode selections, do not by themselves affect power consumption. However, their value is utilized to estimate operating mode dependent power variations.

5 POWER MODEL SYNTHESIS

After collecting switching activity traces from the activity model using simulations of a training set, power models are synthesized in a one-time offline learning process. In the following, we describe cycle-, block- and invocation-level power models associated with resource, basic block and external I/O activity including proposed power model synthesis processes utilizing state-of-the-art machine learning techniques.

5.1 Cycle-Level Power Model

Previous approaches for power estimation at the gate, RTL or micro-architecture level mostly choose a linear function to model the relation between the internal signal switching activity and power consumption of a hardware component. Given the internal and external signal switching

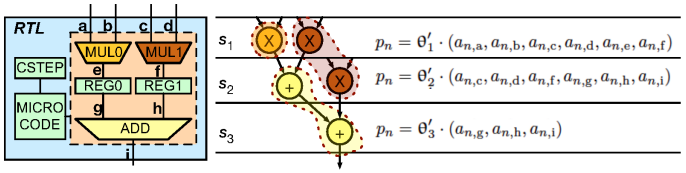


Fig. 7. Example of power model decomposition.

activity column vector $\mathbf{a}(t)$ at time t, power consumption $p(t)$ can be modeled as

$$p(t) = \theta \cdot \mathbf{a}(t), \tag{1}$$

where θ denotes a coefficient row vector. To simplify the model, we assume that related pins, e.g. of buses are grouped, and Hamming distances within a group are utilized as an alternative to individual bit-wise switching activity. With this assumption, power behavior of complex arithmetic units is generally not linear [Bogliolo et al. 2000], but without loss of generality, we use a linear model for the following model derivations.

Ignoring glitching or asynchronous activities, we can convert the continuous power function into a discrete cycle-level model. In general, average power consumption p_n in cycle n can be modeled as

$$p_n = \frac{1}{T} \int_{(n-1)T}^{nT} p(t)\, dt = \theta \cdot \mathbf{a}(nT) = \theta \cdot \mathbf{a}_n = P_{\mathrm{CS}}(\mathbf{a}_n), \tag{2}$$

where $\mathbf{a}_n$ is a discrete activity vector. We utilize resource-level activity vectors captured during functional tracing to drive a single cycle-level power model $P_{\mathrm{CS}}(\mathbf{a}_n)$.

The complexity of the power model in (2) is directly proportional to the dimension of the activity vector, where high dimensionality may create generalization errors in learning processes. To avoid over-fitting, feature sampling, which reduces model dimensions by selecting a key subset of signals, can be utilized, but this may result in a loss of accuracy. As an alternative to traditional feature selection, we introduce a structural model decomposition that uses architectural information to reduce unnecessary signals while improving accuracy.

In white-box models, hardware can be described in FSMD form. Given a finite set of FSMD states S, where the state executed in cycle n is defined as s_n, the power consumption in a given cycle n is dependent on resource utilization in FSMD state s_n. Further, given a finite set of hardware resources R, a resource scheduling and binding function can be defined as $m : S \times R \rightarrow \{0, 1\}$. For instance, $m(r, s) = 1$ indicates that resource r is utilized in the state s. With such mapping information, we can formulate the power consumption in a given cycle n in the following manner:

$$p_n = \sum_{r \in R} m(r, s_n)\, \theta_r \cdot \mathbf{a}_{n,r} = \sum_{r \in R} \theta'_{s_n,r} \cdot \mathbf{a}_{n,r}, \tag{3}$$

where θ_r and $\mathbf{a}_{n,r}$ denote the coefficient and switching activity subvectors corresponding to resource r, respectively. In this formulation, the coefficient factors vectors $\theta'_{s_n,r} = m(r, s_n)\, \theta_r$ are not only resource-, but also state-dependent. Coefficients are masked and zeroed depending on resources utilized in each state. Crucially, such a re-formulation also allows unmasked entries to vary in order to be able to account for any power consumption of resources as well as connected control and glue logic being dependent on the control state.

With this, we can decompose equation (3) into separate and independent cycle-level power models P_{CD,s_n} for each control state s_n. Decomposed models P_{CD,s_n} are separately and independently trained from corresponding state-specific training sets. In the process, we can further exploit mapping

information $m(r, s)$ to identify and remove unnecessary signals $\mathbf{a}_{n,r}$ corresponding to unused resources r and thus masked activity in a particular state s_n:

$$p_n = P_{\mathrm{CD},s_n}(\mathbf{a'}_{s_n,n}) = \theta'_{s_n} \cdot \mathbf{a}'_{s_n,n}, \tag{4}$$

where $\mathbf{a}'_{s_n,n} = (\mathbf{a}_{n,r} | r : m(r, s_n) \neq 0)$ is a subvector composed of activity of those signals used in the state s_n. We illustrate this with the help of a small example. Figure 7 shows a hardware micro-architecture in which three resources are allocated (*MUL0*, *MUL1* and *ADD*). The power consumption of the complete hardware processor can be estimated using a single cycle-level model (P_{CS}) from (2) using all switching vectors connecting to all resources. By contrast, the decomposed power model (P_{CD}) of a given control state instead utilizes the much smaller subset of signals connecting the resources scheduled in the given state only. For example, the power consumption of state S_3 can be estimated with three signals instead of all nine switching vectors. As such, a power model decomposition based on structural micro-architecture information is able to reduce the complexity of the model with little to no information loss. At the same time, it also allows for state-dependent variations in coefficients θ'_s that can account for differences in power consumption of resources and other shared logic.

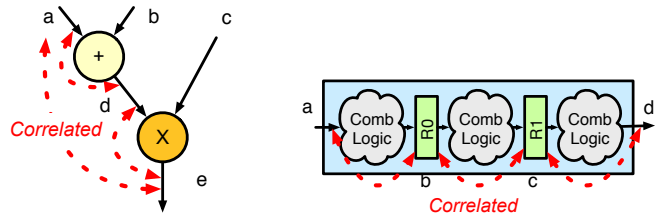
Decomposition based on the FSMD information still has limitations in handling states with high resource utilization, such as pipelined states with many scheduled operators. Moreover, decomposition still requires all signals to be traced across states, which decreases simulation speed. To further reduce feature sets, we additionally leverage a decision tree approach from machine learning [Ratanamahatana and Gunopulos 2003]. Additional feature selection further reduces complexity and improves estimation latency.

5.2 Block-Level Power Model

Instead of internal resource-level activity, our block-level power model only utilizes switching activity of sampled basic block inputs and outputs for power estimation. Given the mapping of block inputs and outputs to registers or ports, internal signal activity in such an approach is indirectly observed from switching activity of input and output signals. Internal signal activity for pipelined and multi-stage hardware architectures in the current cycle can thereby be approximated from future and past switching activities of output and input registers/ports, respectively. We leverage the fact that internal switching activities are highly correlated with input and output activities. Figure 8(a) and Figure 8(b) show such correlations in combinational logic and multi-stage architecture implementations, respectively.

The input and output switching activities of combinational arithmetic operators are linearly correlated [Bogliolo et al. 2000]. Hence, the input switching activity of an operator can be modeled as a linear function of the input switching activity of the driving ancestor. For example, the power consumption of the dataflow graph in Figure 8(a) can be formulated as $p_n = \sum_{v=\mathrm{a}\ldots\mathrm{e}} \theta_v \cdot a_{n,v}$. Using such a linear input-output relationship, we can simplify this equation to $p_n = \sum_{v=\mathrm{a,b,c,e}} \theta''_v \cdot a_{n,v}$.

For pipelined or multi-stage architectures, input activity and activity of the first pipeline stage register are also linearly correlated. Similarly, activity of the second stage is linearly correlated to activity in the first stage. We can therefore approximately estimate internal switching activities throughout the pipeline from the input activity history. However, the activity of registers far away from the input are weakly correlated or not correlated at all. Instead, they are more likely to be correlated to activity at the outputs of the pipeline. Hence, to handle deeply pipelined logic and improve accuracy, we also consider future output activities for prediction. For a given pipeline of



(a) Combinational logic. (b) Multi-stage architecture.

Fig. 8. I/O switching activity correlation.

depth d, we can derive an I/O-based cycle-level power model P_{CI} as

$$\begin{aligned} p_n &= P_{\text{CI}}(\mathbf{a}_{n-d+1,I}, ..., \mathbf{a}_{n,I}, \mathbf{a}_{n,O}, ..., \mathbf{a}_{n+d-1,O}) \\ &= \textstyle\sum_{i=0}^{d-1} \boldsymbol{\theta}''_i \cdot (\mathbf{a}_{n-i,I}, \mathbf{a}_{n+d-i-1,O}), \end{aligned} \tag{5}$$

where $\mathbf{a}_{n,I}$ and $\mathbf{a}_{n,O}$ denote the input and output activity vectors, and $\boldsymbol{\theta}''_i$ denotes coefficient vectors corresponding to pipeline stage i. For example, the power consumption of the pipelined hardware implementation in Figure 8(b) can be computed as $p_n = \sum_{v=\text{a,b,c,d}} \theta_v \cdot a_{n,v}$. Using I/O history and future, we can instead re-formulate power consumption as $p_n = \sum_{i=0}^{2} \boldsymbol{\theta}''_i \cdot (a_{n-i,a}, a_{n+2-i,\text{d}})$. The power consumption of the micro architecture in Figure 7 can similarly be formulated as $p_n = \sum_{i=0}^{1} \boldsymbol{\theta}''_i \cdot (a_{n-i,\text{a}}, a_{n-i,\text{b}}, a_{n-i,\text{c}}, a_{n-i,\text{d}}, a_{n+1-i,\text{i}})$ using activity history of primary I/O ports 'a', 'b', 'c', 'd', and 'i'. Note that this model applies to power estimation in all cycles/states S_n, $n = 1...3$, i.e. the stage-wise decomposition here is different from the state-wise in (4).

A block-level power model can then be formulated to estimate an average power consumption per basic block using switching activity of basic block inputs and outputs. Given a set of basic blocks B, where the m-th executed basic block is defined as b_m, the average power consumption $\bar{p}_m$ of basic block b_m can be formulated from (5) as

$$\bar{p}_m = \frac{1}{\bar{L}_m} \sum_{n=n_m}^{n_m+\bar{L}_m-1} \sum_{i=0}^{d-1} \boldsymbol{\theta}''_i \cdot (\mathbf{a}_{n-i,I}, \mathbf{a}_{n+d-i-1,O}), \tag{6}$$

where n_m and $\bar{L}_m$ denote the start cycle time and execution cycles of the m-th basic block, respectively. To simplify the equation, we can remove the summations over the pipeline and execution cycles by introducing a new coefficient vector $\bar{\boldsymbol{\theta}}$ and thus define a single block-level power model P_{BS} in the following manner:

$$\bar{p}_m = \frac{1}{\bar{L}} \bar{\boldsymbol{\theta}} \cdot \bar{\mathbf{a}}_m = P_{\text{BS}}(\bar{\mathbf{a}}_m), \tag{7}$$

where $\bar{L} = \max_m \bar{L}_m$ denotes the maximum execution cycles over all basic blocks and $\bar{\mathbf{a}}_m = (\mathbf{a}_{n_m+j,I}, \mathbf{a}_{n_m+j+d-1,O} | \, 1-\text{d} \leqq j < \bar{L})$ denotes a concatenation of all the activity of input and output ports that can flow through the pipeline for the length of the block. For blocks with length $\bar{L}_m < \bar{L}$, we append zero pad vectors ($\mathbf{a}_{n_m+j,I} = \mathbf{a}_{n_m+j+d-1,O} = \vec{\mathbf{0}}$, $\bar{L}_m < j < \bar{L}$) to keep the same dimension for all $\bar{\mathbf{a}}_m$.

In block-level gray-box activity models, we can only observe inputs and outputs of each basic block, not the actual cycle-by-cycle activity of all primary input and output registers or ports. For example, in Figure 7, 'a', 'b', 'c', 'd', and 'i' are primary I/O ports of the whole hardware. Assuming that a basic block starts and ends with S_1 and S_3, block inputs are 'a', 'b', 'c', and 'd' in state S_1, and

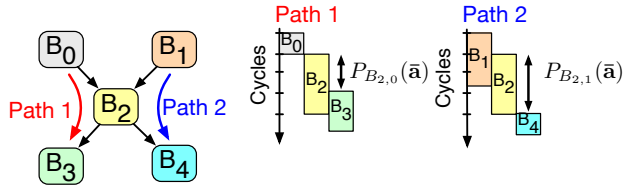


Fig. 9. Basic block-level decomposition for multi-path control flow.

'c' and 'd' again in state S_2. Block output is 'i' once in S_3. We capture only these values, which are represented as different variables in the activity model code. Similarly, assuming the block starts with S_2, block inputs would be 'c', 'd', 'g', and 'h'. In this case, instead of using history of primary input ports for estimating activity of internal registers *REG*0 and *REG*1, we directly capture and trace block inputs 'g' and 'h'. In all cases, we can therefore only use actual inputs and outputs of the current basic block to estimate the power consumption, where feature selection is implicitly applied to remove unnecessary signals not utilized in the block, thus reducing model complexity. With this, we can further decompose equation (7) into separate and independently learned block-specific power models P_{BD,b_m} for each basic block b_m in the following way:

$$\bar{p}_m = P_{\mathrm{BD},b_m}(\bar{\mathbf{a}}'_{m,b_m}) = \frac{1}{\bar{L}_m}\bar{\theta}'_{b_m} \cdot \bar{\mathbf{a}}'_{m,b_m} \tag{8}$$

where $\bar{\mathbf{a}}'_{m,b_m}$ and $\bar{\theta}'_{b_m}$ denote the block-level activity vector and corresponding coefficient vector for basic block b_m, respectively.

In case of pipelined or speculative scheduling, executions of successive basic blocks can overlap. Since we can not separate power consumption of overlapped blocks during training, we need to account for such periods by attributing power contributions of previous blocks that are still executing to the model of a current block. We redefine the execution length $\bar{L}_m$ of a characterized block b_m as the cycle difference between the start of its first operation and the start of the first operation of the next block. In other words, a block is defined to end when the next block starts. In addition, we extend the activity vector of a block by including activity vectors of all overlapping blocks. Such extended activity vectors may increase the complexity of the model, but only a small part of the transaction activities contribute to the power consumption in any given cycle, which results in many of the elements of the feature vector being zero or small. To prune away such uncorrelated features, we apply an additional feature selection for each decomposed model.

Finally, overlapped execution of blocks can also depend on control flow. Figure 9 shows the control flow graph of five basic blocks with two different paths (Path 1 and Path 2). Depending on the taken path, the length and overlapping of block B_2 varies. To account for such variations, we extract possible execution paths during training and build different power models $P_{\mathrm{BD},b_{m,k}}$ for each possible path k through a block b_m.

5.3 Invocation-Level Power Model

For black-box cases, we propose an invocation-level power model that estimates an power consumption per invocation using switching activity of external I/O and control signals only. Given a per-invocation execution latency $\bar{\bar{L}}_l$ and assuming that the l-th invocation starts in cycle n_l, we can formulate an invocation-level power model P_{IC} that itself is not learned, but instead computes the average power $\bar{\bar{p}}_l$ of invocation l by averaging cycle-by-cycle power obtained from a single

I/O-based cycle-level power model P_{IC} according to (5) over the length of the invocation $\bar{\bar{L}}_l$:

$$\begin{aligned}\bar{\bar{p}}_l &= \tfrac{1}{\bar{\bar{L}}_l} \textstyle\sum_{n=n_l}^{n_l+\bar{\bar{L}}_l-1} P_{\mathrm{CI}}(\ldots, \mathbf{a}_{n,I}, \mathbf{a}_{n,O}, \ldots) \\ &= P_{\mathrm{IC},\bar{\bar{L}}_l}(\mathbf{A}_{I,l}, \mathbf{A}_{O,l}).\end{aligned} \tag{9}$$

Here, $\mathbf{A}_{I,l}$ and $\mathbf{A}_{O,l}$ denote an external input and output activity matrix composed of $\bar{\bar{L}}_l$ input and output activity column vectors $\mathbf{a}_{n,I}$ and $\mathbf{a}_{n,O}$, $n_l \leqq n \leqq n_l + \bar{\bar{L}}_l - 1$, respectively. In this formulation, we assume that invocations do not overlap, and we enforce the following initial condition on the input and output activity vectors: $\mathbf{a}_{n,I} = \mathbf{a}_{n,O} = \vec{\mathbf{0}}$ for all $n < n_l$ or $n > n_l + \bar{\bar{L}}_l$.

In this model, re-arrangement of transactions and cycle-by-cycle I/O tracking is required to compute cycle-level switching activity on external input and output ports, which introduces a significant computation overhead. However, by fully expanding equation (9) following (5), it can be seen that invocation-level power does not actually depend on the order of activity information. Furthermore, if there is no transition in cycle n for input or output ports, the corresponding elements in external activity matrices $\mathbf{A}_{I,l}$ or $\mathbf{A}_{O,l}$ will be zero and terms will be masked. This indicates that we can formulate a single invocation-level power model $P_{\mathrm{IS},\bar{\bar{L}}_l}$ by finding the contributed and reordered coefficients $\bar{\bar{\boldsymbol{\theta}}}$ purely from transaction-level activity vectors $\bar{\bar{\mathbf{a}}}$:

$$\bar{\bar{p}}_l = \frac{1}{\bar{\bar{L}}_l} \bar{\bar{\boldsymbol{\theta}}} \cdot \bar{\bar{\mathbf{a}}} = P_{\mathrm{IS},\bar{\bar{L}}_l}(\bar{\bar{\mathbf{a}}}). \tag{10}$$

We create multiple such power models, one for each possible invocation latency $\bar{\bar{L}}_l$.

Transaction-level activity vectors are computed using Hamming distances over transaction data traces, where $\bar{\bar{\mathbf{a}}}$ is a concatenated vector composed over all transactions in an invocation, which does not require cycle-level rearrangement or cycle-by-cycle activity computation. However, the worst-case dimension of $\bar{\bar{\mathbf{a}}}$ is the product of the total number of external ports and execution cycles $\bar{\bar{L}}_l$, which may create generalization errors in learning processes.

To address such issues, we previously decomposed cycle- and block-level power models into separate and independent models for each state or block. However, this is not possible in black-box models, where control flow or state composition as well as scheduling and binding information is not available. However, since the current state is a function of the cycle n, we can indirectly capture the state based on n and an additional control vector $\mathbf{c}$. We thereby assume that control signals $\mathbf{c}$, if any, determine the IP operating mode on a per invocation basis, but remain constant over one invocation. With this, we can decompose the power model into separate and independently learned models $P_{\mathrm{ID},n}(\mathbf{c}, \bar{\bar{\mathbf{a}}})$ for each cycle n. In the process, we convert equation (10) into an ensemble of decoupled multiple regressions as follows:

$$\begin{aligned}\bar{\bar{p}}_l = P_{\mathrm{IE},\bar{\bar{L}}_l}(\mathbf{c}, \bar{\bar{\mathbf{a}}}) &= \tfrac{1}{\bar{\bar{L}}_l} \textstyle\sum_{n=n_l}^{n_l+\bar{\bar{L}}_l-1} P_{\mathrm{ID},n}(\mathbf{c}, \bar{\bar{\mathbf{a}}}), \\ P_{\mathrm{ID},n}(\mathbf{c}, \bar{\bar{\mathbf{a}}}) &= \bar{\bar{\boldsymbol{\theta}}}'_n \cdot (\mathbf{c}, \bar{\bar{\mathbf{a}}}),\end{aligned} \tag{11}$$

where $\bar{\bar{\boldsymbol{\theta}}}'_n$ denotes a decomposed coefficient vector and $(\mathbf{c}, \bar{\bar{\mathbf{a}}})$ the concatenation of control inputs and transaction activity. In (11), the dimension of each model $P_{\mathrm{ID},n}$ is the same as the single power model $P_{\mathrm{IS},\bar{\bar{L}}_l}$ from (10), i.e. the decomposed models use the complete transaction activity $\bar{\bar{\mathbf{a}}}$ at their input. However, only a small part of the transaction activities actually contribute to the power consumption in any given cycle. We leverage a decision tree based feature selection for each decomposed model to remove such unimportant features and reduce model complexity. As a result, the uncertainty of the individual cycle-models is improved and there is less chance to run into

generalization errors. Note that (11) is similar to (9), but (9) uses a single, uniform instead of separate and independent models for each cycle. Overall, the total number of models to learn is increased. However, each decomposed model uses the same input vectors, which enables parallel learning and prediction without additional overhead. Note that models could be further decomposed along control inputs. However, as the control space is exponential in the number of control signals, this would result in significant learning overhead.

The decomposition in (11) represents a form of ensemble learning. Ensemble learning is known to achieve better accuracy by utilizing the diversity over multiple learning models [Bishop 2007]. Traditional ensemble learning introduces diversity by dividing the training set, training each model with the partitioned training set, and then predicting the target value as the average over the prediction values of each model. By contrast, we introduce diversity by decomposing the model into separate cycle models. Compared to the single invocation model in (10), if errors of each decomposed cycle model are uncorrelated, the error of the ensemble model can be reduced by a factor of $\bar{L}$ [Lee et al. 2015b]. In general, each decomposed model predicts a different cycle power, which implies that individual cycle errors will not be highly correlated. Moreover, we utilize non-linear learning approaches to prevent correlations between models, as will be discussed in the following section. As such, we can expect that the ensemble model always provides better accuracy than the single invocation one. Note that an averaged cycle-level model P_{IC} also estimates cycle-by-cycle power behavior using a single model instead of multiple decomposed ones. However, since each error is generated from the same model, errors are highly correlated. As a result, we can also expect that the ensemble model shows better prediction accuracy than a single cycle-level model.

5.4 Model Selection and Training

Each power model is trained from given power and activity traces. The activity traces collected from activity model simulation contain the start cycle times, execution cycle times, decomposed model IDs, and corresponding switching vectors. Power traces contain actual power measurements from an equivalent gate-level simulation for the same set of training inputs. Activity and power traces are partitioned into model IDs, and each power model is then trained with the corresponding partitioned traces. Synthesized power models are thereby able to compute data-dependent power consumption estimates from the captured activity traces.

In general, a least squares linear regression over a set of training vectors has been widely employed to find the coefficient of power models. However, as mentioned previously, power behavior of complex arithmetic units is generally not correlated linearly to Hamming distances of inputs and outputs [Bogliolo et al. 2000]. Moreover, control dependencies may have a non-linear correlation with power consumption in the invocation-level case. A decomposition along control inputs could potentially linearize them, but this increases learning complexity. Furthermore, linear regressions provide not enough diversity, which increases the error in the final ensemble model [Gashler et al. 2008].

By contrast, depending on hardware functionality, input data statistics and complexity of models, a non-linear machine learning model can represent the power consumption behavior better than a typical linear least squares model while also providing more diversity, but this comes at the expense of estimation overhead. We thus evaluate various linear as well as non-linear regression models as part of our experiments.

For online power estimation, calls to a regression model library are inserted as part of the annotations in the activity model. At the start of hardware simulation, synthesized power model parameters, coefficients, and data structures are loaded into regression models. As part of this process, unnecessary signal tracing calls inserted during the activity annotation process are removed

Table 1. Benchmark summary

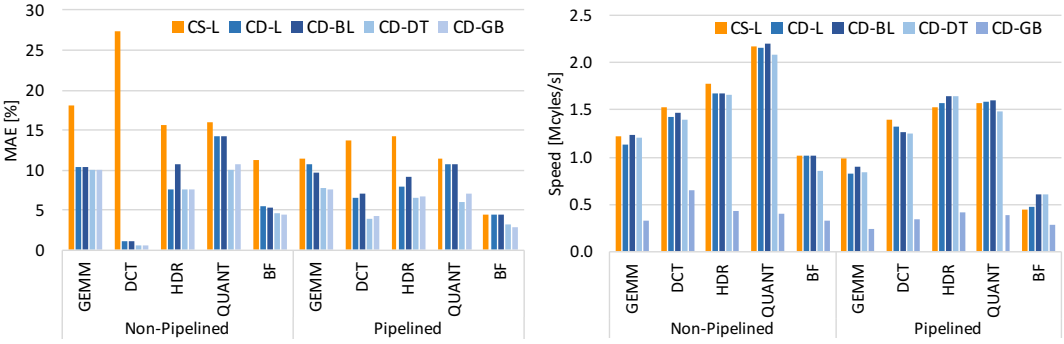
	Pipe	Basic Blocks	States	Cycles per Invocation	Gates	Traced IR Op.	Traced Block I/O	Traced Ext. I/O	Train Invoc.	Test Invoc.	Total Test Cycles	Avg. Power
GEMM	No	10	6	734	703	11	4	3	2,000	5,000	3,670,000	0.36mW
	Yes	6	4	436	964	20	4	3	2,000	5,000	2,180,000	0.72mW
DCT	No	6	23	179	7,007	139	32	8	3,000	10,800	1,933,200	0.67mW
	Yes	6	12	94	6,309	127	32	8	3,000	10,800	1,015,200	2.05mW
HDR	No	13	18	995	4,883	70	24	12	988	1,200	1,194,000	0.81mW
	Yes	10	20	825	7,887	104	52	12	988	1,200	990,000	1.07mW
QUANT	No	6	6	194	1,032	7	10	4	3,600	12,288	7,150,452	0.24mW
	Yes	6	4	68	1,035	8	10	4	3,600	12,288	2,506,752	0.43mW
BF	No	4	12	135	19,364	159	7	4	2,460	24,000	3,240,000	3.65mW
	Yes	4	4	25	22,228	169	7	4	2,460	24,000	600,000	4.08mW

to improve simulation speed. At run-time, the power model then estimates the power consumption of the hardware implementation from the dynamically computed switching activities.

6 EXPERIMENTAL RESULTS

We have implemented a fully automated realization of our power modeling flow. We integrated our flow with the Xilinx Vivado HLS engine utilizing the LLVM compiler framework [Lattner and Adve 2004] for automatic activity annotation, prediction insertion and IP model generation. Power model synthesis utilizes the scikit-learn [Pedregosa et al. 2011] machine learning library for Python. For fast online prediction, we natively implemented C++ based power estimation models to reduce Python binding overhead. We have released our learning-based system-level power modeling tool set in open-source form at [Lee 2017].

We applied our flow to generate models for pipelined and non-pipelined hardware designs of a 6x6 general matrix multiplication (GEMM), a 2D discrete cosine transform (DCT), a JPEG quantizer (QUANT), a weight computation block of a high dynamic range (HDR) imaging application [Mertens et al. 2007] and a bilateral filter (BF) [Paris et al. 2009]. The quantizer has two control inputs for choosing a quantization table and the image scaling quality. All hardware designs were synthesized using Synopsys Design Compiler with the Nangate 45nm Open Cell Library [Nangate 2017] at 200Mhz clock frequency. Gate-level power was estimated using Synopsys PrimeTime PX with VCD files generated from full gate-level simulation. All experiments were performed on a quad-core Intel i7 workstation running at 3.5 GHz. To learn each power model, we used training sets generated from different random seeds or images. To generate test vectors, the GEMM design was simulated with 5000 random test matrices. A 640x320, a 512x512, and a 200x100 24-bit RGB image are used to generate DCT, QUANT, and HDR test vectors, respectively. The test vectors for the BF design are sampled from a 600x402 gray image. Three different quality factors and two different table setting are utilized to generate the test set for the QUANT design. Table 1 summarizes benchmarks and synthesis results including number of basic blocks and states in each design, execution cycles per invocation, the number of signals traced and utilized by each activity and power model, the size of training and test sets, and the average power consumption of each test set simulation.



(a) Cycle-by-cycle power accuracy.

(b) Estimation speed of models.

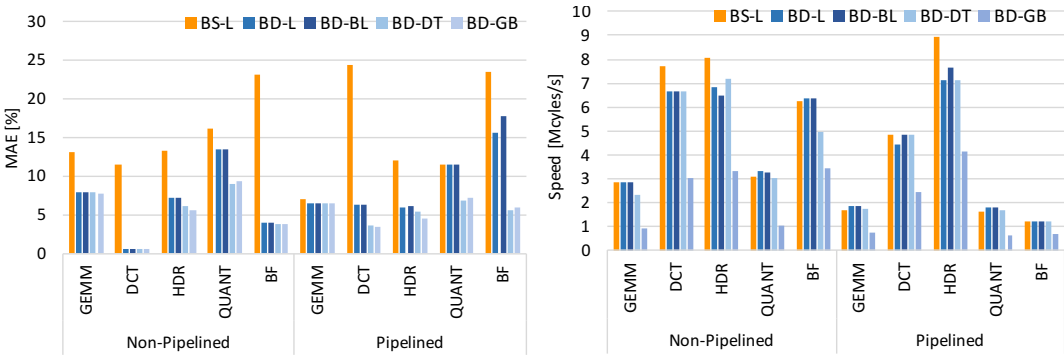
Fig. 10. Comparison of cycle-level model accuracy and speed.

6.1 Cycle-Level Power Estimation Results

Figure 10 shows accuracy and speed of proposed decomposed cycle-level power models (CD) as compared to a single cycle-level power model (CS) across various benchmarks. We measured data-dependent cycle-by-cycle mean absolute error (MAE) of values predicted by each model compared to gate-level simulations, normalized against average power over the full simulation. We compare both power models utilizing a least squares linear regression (CS-L and CD-L) against a decomposed model using a decision tree regression (CD-DT), a linear Bayes ridged regression (CD-BL), or a gradient boosting regression composed of multiple decision trees (CD-GB). In all cases, we applied a decision tree based feature selection to remove uncorrelated features and then unused signals.

We can observe that, in all cases, linear decomposed models (CD-L) show on average 1.8x better accuracy than single cycle-level models with least squares regression (CS-L). The proposed structural decomposition technique results in up to 26% less MAE, which indicates that decomposition is a key factor in improving model accuracy. Significant accuracy improvements are observed in the non-pipelined DCT case. In the non-pipelined DCT, there is a substantial power variation across states. It is generally hard to capture such state-dependent trends in a single cycle model. Compared to simpler designs (QUANT, GEMM), higher accuracy improvements are observed in complex hardware implementations (HDR, DCT, BF), which indicates that decomposition is more effective in large designs. Among all models, decomposed power models utilizing decision tree (CD-DT) or gradient boosting (CD-GB) regression show better accuracy than others. Linear models (CD-BL, CD-L) show the worst results in all cases, with up to 4.8% higher errors, where Bayesian models (CD-BL) generally perform similar or worse than standard least squares regressors.

Speed (Figure 10(b)) generally depends on the complexity versus execution cycles of the design. Single models are slightly faster than decomposed ones on average. In the single models, more activity features are treated as correlated and thus removed during feature selection, which results in significantly less accuracy but better speed. Models using gradient boosting regression (CD-GB) are on average 3.6x slower than others. Gradient boosting needs to call multiple subcomponent models, which generally introduces much larger prediction overhead. The decision tree model (CD-DT) is thereby 3.4x faster than a gradient boosting (CD-GB) one at similar accuracy. Least squares models (CD–L) are on average slightly faster, but decision tree models (CD-DT) provide on average 1.3x better accuracy. Overall, when comparing different regression methods and models, results show that a decomposed power model utilizing a decision tree regression (CD-DT) provides the best trade-off between accuracy and speed. The CD-DT model achieves on average 1.3Mcycles/sec



(a) Basic block-by-basic block power accuracy.

(b) Estimation speed of models.

Fig. 11. Comparison of block-level model accuracy and speed.

at 94% accuracy. Compared to our previous work [Lee et al. 2015a] that did not include overhead for writing results to disk, when similarly excluding file I/O, decision tree models (CD-DT) are on average 2.4x faster at the same accuracy. Such speed gains are achieved using an improved resource-level activity computation (Section 4.1) and adopting a C++ instead of a Python based power estimation model.

6.2 Block-Level Power Estimation Results

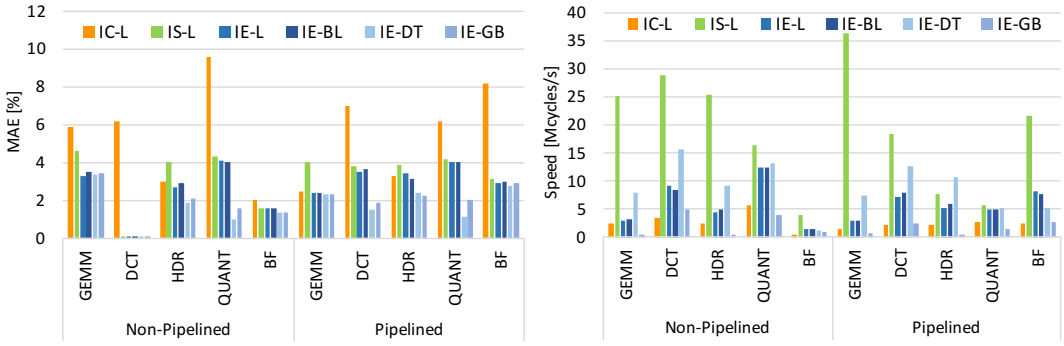
Figure 11 shows model accuracy and speed of proposed block-level decomposed power models (BD) as compared to the single basic block models (BS) across various benchmarks. To evaluate block-by-block MAE, we convert the gate-level cycle-by-cycle trace by assigning the average power dissipation to corresponding blocks. We compare both power models utilizing a least squares linear regression (BS-L, BD-L) against a decomposed model using a decision tree (BD-DT), a linear Bayes ridged (BD-BL), or a gradient boosting (BD-GB) regression. Decision tree based feature selection is applied in all cases.

The decomposed model using least squares regression (BD-L) shows up to 19% higher accuracy than a single model (BS-L). In case of pipelined QUANT and GEMM, accuracy is not improved since one single loop body block takes up most of the execution time. Non-linear regression models (BD-DT and BD-GB) again show better accuracy than the linear ones (BD-L, BD-BL), with up to 12.1% lower errors.

Figure 11(b) compares speed across various benchmarks. Here, decomposed models show faster estimation speed than single ones, since the latter require the union of all possible block inputs and outputs to be provided for each block. As before, the decision tree model (BD-DT) provides the best balance. It is on average almost as fast as linear models, and 2x faster than a gradient boosting (BD-GB) one at similar accuracy. Overall, block-level models provide similar accuracy than cycle-level estimates at significantly improved speed. The BD-DT model achieves on average 4.1Mcycles/sec at 94.5% accuracy.

6.3 Invocation-Level Power Estimation Results

Figure 12 compares model accuracy and speed of proposed invocation-level ensemble models (IE) as compared to averaged single cycle-level (IC) and single invocation-level power models (IS) across various benchmarks. We measured data-dependent invocation-by-invocation MAE of values predicted by each model compared to gate-level simulations. We compare all power models utilizing a least squares regression (IC-L, IS-L, IE-L) against an ensemble model using a linear Bayes ridged



(a) Invocation-by-invocation power accuracy.

(b) Estimation speed of models.

Fig. 12. Comparison of invocation-level model accuracy and speed.

(IE-BL), a decision tree (IE-DT), or a gradient boosting (IE-GB) regression. Decision tree based feature selection is applied in all cases.

The ensemble model using a least squares regression (IE-L) shows up to 6.2% and 1.4% lower errors compared to the single cycle- and invocation-level models (IC-L and IS-L), respectively. The non-pipelined DCT hardware shows large power variations in each cycle, but almost constant power consumption for each invocation. Since errors of the cycle-level model (IC-L) generated from the same, single cycle model are highly correlated, the error is not significantly reduced by averaging over invocations. The single invocation-level model (IS-L) shows better accuracy on average, but higher errors in complex cases utilizing long transaction activity vectors (GEMM, HDR). By contrast, the ensemble estimation (IE-L) utilizing decomposed cycle models does not suffer from such correlation and complexity problems. Among ensemble models, linear regressions (IE-L and IE-BL) again show the worst accuracy. Non-linear models (IE-DT, IE-GB) show up to 3.3% additional accuracy improvement. The accuracy improvements in QUANT benchmarks are bigger than in other benchmarks due to the non-linear correlation between control inputs and power consumption. Overall, IE-DT and IE-GB estimate invocation-level power dissipation to within 3.3% MAE compared to gate-level power results.

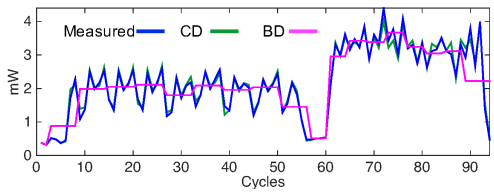
When comparing speed (Figure 12(b)), due to its simplicity, the single invocation model (IS-L) is on average significantly faster than others. Both the ensemble and cycle-level power models (IE-L and IC-L) estimate at a cycle-by-cycle level, but the ensemble models (IE-L) are on average 5x faster due to light-weigth activity computation and parallelized cycle-level prediction. Among ensemble models, decision tree (IE-DT) models are the fastest. The dimension of activity features for the invocation-level model is much higher than resource- and block-level activity features. With such high-dimensional feature vectors, decision tree regressions can be faster than linear ones. Overall, when comparing different regression methods and models, results show that IE-DT provides the best trade-off between accuracy and speed. The IE-DT model achieves on average 8.8Mcycles/sec at 98% accuracy. By adopting an online I/O activity computation, it is on average 24x faster than our prior offline estimation in [Lee et al. 2015b].

6.4 Overall Speed and Accuracy Comparison

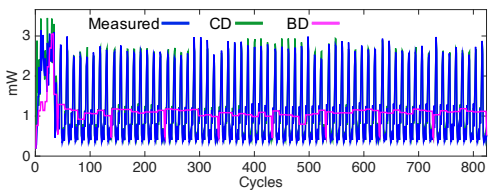
Table 2 summarizes accuracy and speed of models across benchmarks. We compare the accuracy of cycle-level decomposed models (CD), basic block-level decomposed models (BD), and invocation-level ensemble models (IE) utilizing decision tree regression in all cases. In addition to average

Table 2. Summary of modeling accuracy and estimation speed

	Pipe	MAE						Average Error			Speed [Cycles/Sec]					
		Cycle	Basic Block		Invocation											
		CD	CD	BD	CD	BD	IE	CD	BD	IE	C Code	CD	BD	IE	RTL	Gate
GEMM	No	10.1%	7.9%	7.8%	3.1%	3.0%	3.3%	0.4%	0.5%	0.5%	220M	1.20M	2.34M	7.92M	51K	0.61K
	Yes	7.9%	6.5%	6.5%	2.2%	2.2%	2.3%	0.1%	0.1%	0.1%	130M	0.84M	1.70M	7.27M	35K	0.36K
DCT	No	0.6%	1.3%	0.5%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	32M	1.40M	7.44M	15.59M	16K	0.41K
	Yes	3.9%	2.5%	3.6%	1.1%	1.4%	1.6%	0.5%	0.2%	0.2%	17M	1.25M	4.83M	12.69M	5.9K	0.19K
HDR	No	7.6%	3.2%	6.1%	2.0%	1.9%	1.8%	0.9%	0.7%	0.7%	32M	1.66M	7.61M	9.17M	13K	0.28K
	Yes	6.6%	3.1%	5.4%	2.4%	1.9%	2.4%	1.0%	1.4%	1.9%	27M	1.65M	8.94M	10.73M	11K	0.20K
QUANT	No	10.0%	10.0%	9.0%	3.6%	3.0%	1.0%	0.1%	0.9%	0.2%	48M	2.08M	2.91M	13.00M	19K	1.80K
	Yes	6.0%	6.0%	6.8%	1.7%	3.0%	1.1%	0.4%	0.7%	0.8%	17M	1.48M	1.64M	5.11M	9.3K	1.52K
BF	No	4.6%	2.7%	3.8%	0.7%	1.3%	1.4%	0.5%	0.5%	0.7%	64.80M	0.85M	4.98M	1.28M	19K	1.80K
	Yes	3.2%	4.3%	5.6%	1.5%	4.5%	2.8%	0.9%	3.9%	1.8%	17M	1.48M	1.64M	5.11M	9.3K	1.52K
Avg.	-	6.0%	4.8%	5.5%	1.8%	2.2%	1.8%	0.5%	0.9%	0.7%	60M	1.30M	4.08M	8.79M	17K	0.58K

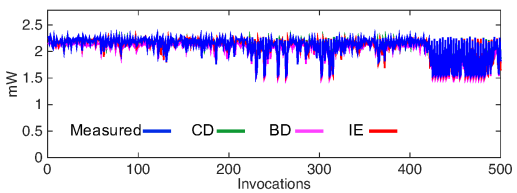


(a) DCT simulation.

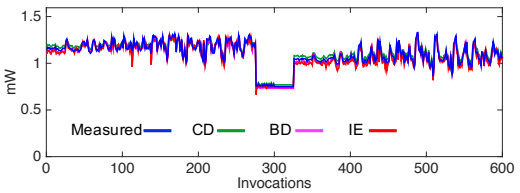


(b) HDR simulation.

Fig. 13. Cycle-by-cycle power traces for a single invocation.



(a) DCT simulation.



(b) HDR simulation.

Fig. 14. Invocation-by-invocation power traces.

errors across the whole simulation, we measure the data-dependent cycle-by-cycle, basic block-by-basic block, and invocation-by-invocation MAE predicted by each relevant model compared to gate-level simulations. We compute the basic block-by-basic block, and invocation-by-invocation errors of the cycle- and basic block-level models, respectively, by averaging power models over blocks and invocations. The cycle-level model (CD) shows better block-level accuracy than the basic block-level model (BD), similar invocation-level accuracy to the ensemble model (IE), and the best average error across the whole simulation since it utilizes the largest amount of tracing. The BD model utilizes the smallest number of decomposed models, which results in the worst average prediction accuracy among all models. The ensemble approach (IE) utilizes the largest number of models, which enables a better invocation-level accuracy than others. Across all benchmarks,

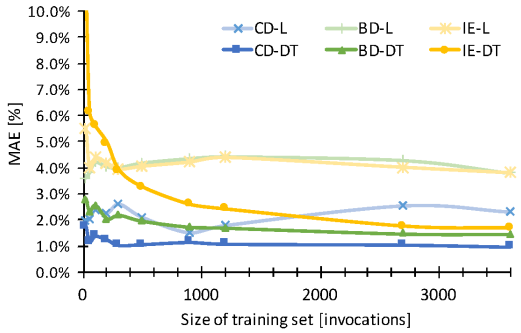


Fig. 15. Learning overhead vs. model accuracy for pipelined DCT.

average errors of cycle-, block- and invocation-level models across the whole simulation are below 1%, 2%, and 2%, respectively.

We also compare simulation speed of generated models to pure source-level, RTL, or gate-level simulation. The block-level model (BD) is on average 3x faster than the cycle-level one (CD). This is due to its reduced activity features and smaller number of power model function calls. Both IE and CD utilize cycle-level estimation models, but the invocation-level model is on average 7x faster. The light-weight activity computation and parallelized internal component power estimation of the I/O-based ensemble approach improves simulation throughput significantly. Overall, compared to a pure source-level simulation, the cycle-, block- and invocation-level models are on average 45x, 15x and 7x slower. However, they are about 77x, 239x and 516x faster than RTL power simulation, and 2,200x, 7,100x, and 15,300x faster than gate-level estimation.

Figure 13 and 14 show cycle-by-cycle and invocation-by-invocation profiles of estimated versus measured power waveforms for the pipelined DCT and HDR designs, respectively. Note that the cycle-level trace of the block-level model shows the averaged power at block granularity. As the profiles show, our proposed models can accurately track power behavior within each invocation, as well as data-dependent effects across different invocations of the same design.

6.5 Learning Overhead

The major learning overhead is collecting gate-level simulation results to construct the training vectors. Depending on the trace length and design complexity, we were able to generate gate-level power traces for training within 6 to 30 minutes. The learning times of cycle-, block-, and invocation-level models are proportional to the number of decomposed models, i.e. states, basic blocks, and execution cycles per invocation, respectively. As mentioned previously, the invocation-level model supports parallel learning, which results in comparable learning speed to cycle-level models. The synthesis time of block-level models is the shortest with 30 to 90 seconds. Synthesis of cycle- and invocation-level models is on average three times slower than block-level models, taking 30 to 200 seconds. Overall, we were able to synthesize power models in each case within 34 minutes including trace generation.

Figure 15 further details the learning overhead and accuracy at different modeling levels for the pipelined DCT benchmark. By increasing the size of training sets, we explore trade-offs between learning overhead and final accuracy of trained models. We measure accuracy as invocation-by-invocation MAE of CD, BD and IE models utilizing either decision tree (-DT) or least squares linear regression (-L). In all cases, as discussed before, models utilizing decision tree regression always show better results than simple linear ones. Models with linear regression suffer from overfitting

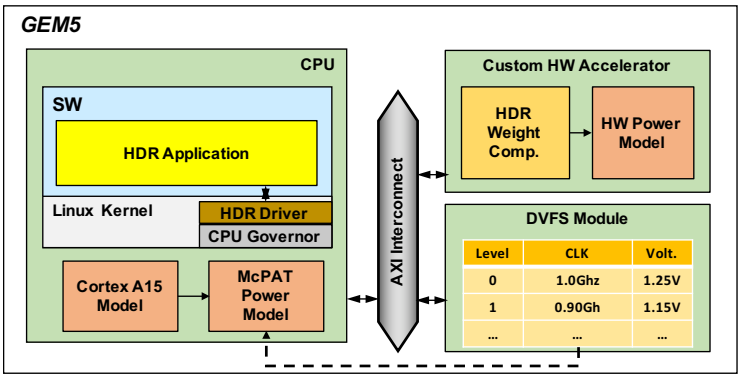


Fig. 16. GEM5 simulation of HDR application with integrated hardware accelerator model.

trends, which indicates that IP power behavior is inherently non-linear. We can observe that the CD-DT model provides the best accuracy for the same size of the training set, reaching more than 99% accuracy for a training set with 300 vectors. The IE-DT model shows the worst learning efficiency, reaching 96% accuracy with the same amount of training. For the same training size, models based on more detailed and fine-grain estimation generally show better accuracy than more coarse-grain ones. Combined with opposing trends in estimation speed, this establishes a trade-off between modeling level, accuracy, training efficiency and speed.

6.6 Integration with Full-System Virtual Platform Simulation

In order to demonstrate benefits of our models for virtual platform prototyping and system-level design space exploration, we integrate the generated power and performance model for the HDR weight computation accelerator into a full-system simulation of a complete HDR application. For online power and performance estimation, we integrate a C++ model of the HDR weight computation block annotated with our cycle-accurate white-box power model into the GEM5 [Binkert et al. 2011] system simulator. We combine the weight computation accelerator with a cycle-accurate simulation of the remaining HDR application executing on an ARM Cortex-A15 CPU running a Busybox Linux kernel version 2.6.38. We manually design the necessary software drivers and hardware wrappers to provide the communication interface between the HDR application on the CPU and the generated hardware accelerator model. Figure 16 shows an overview of the final system model integrating the HDR accelerator.

To enable processor power estimation, we modify GEM5 to integrate a McPAT [Li et al. 2013] based power model for the CPU as well as DVFS capabilities based on [Spiliopoulos et al. 2013]. In an offline process, we first extract power coefficients from McPAT for the given Cortex A15 processor description. Using these coefficients, we then integrate a fast online power model that estimates CPU power consumption using statistics collected from the GEM5 simulator together with DVFS state information sampled every 5ms. The CPU governor in the Linux OS running on the platform thereby controls processor voltage and frequency between 1GHz and 310MHz through a DVFS module model. The hardware accelerator is simulated to run at a fixed 200MHz.

We demonstrate system-level architecture design space exploration for the HDR application using three different system configurations: a floating-point software-only design (Float), a fixed-point software-only design (Fixed), and fixed-point hardware-accelerated system (FixAcc). Figure 17 shows the simulated full-system power traces, total HDR execution time and total energy consumption for each configuration. Full-system simulations run at speeds of 29Kcycles/sec (Float),

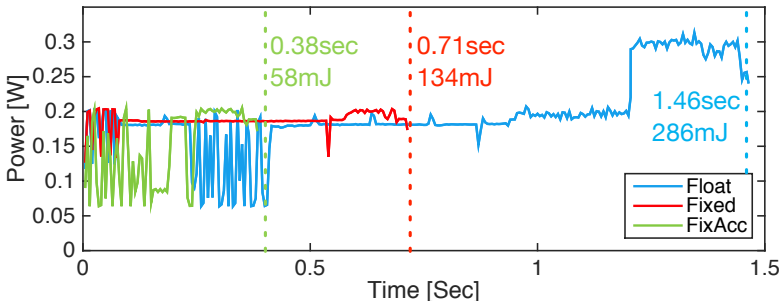


Fig. 17. Simulated traces of total system power for HDR application.

25Kcycles/sec (Fixed) and 26Kcycles/sec (FixAcc) in terms of fixed 1GHz system reference clock cycles simulated per second. This demonstrates that overall simulation speed is limited by CPU simulations. Even our most detailed, cycle-accurate hardware model introduces only negligible simulation overhead and is, in fact, faster than the cycle-accurate CPU model.

7 SUMMARY AND CONCLUSIONS

In this paper, we presented a novel machine learning-based approach for extending fast high-level functional models of hardware IPs with data-dependent accurate power estimates. Our power modeling approach is fully automated by integrating with commercial HLS tools for custom hardware synthesized by HLS. Depending on hardware observability, an automated annotation flow first generates activity models, which allow capturing data-dependent hardware activity without detailed micro-architecture simulation. Our power model synthesis flow then leverages state-of-the-art machine learning techniques to synthesize power models at different granularities. We propose novel model decompositions that reduce model complexities and increase estimation accuracy. We have evaluated our flow on several industry-strength benchmark designs. Results show that our proposed power models are able to achieve orders of magnitude speedup compared to gate-level or RTL power simulation, all while estimating cycle-, block- and invocation-level power to within 10%, 9% and 3% of a commercial gate-level estimation tool. Such fast and accurate power and performance models allow integration with virtual platform or full-system simulators to support system-level architecture exploration.

ACKNOWLEDGMENTS

This work has been partially supported by Intel and NSF grant CCF-1218483.

REFERENCES

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. 2011. The Gem5 Simulator. *ACM SIGARCH Computer Architecture News (CAN)* 39, 2 (Aug. 2011).

C. Bishop. 2007. *Pattern Recognition and Machine Learning*. Springer.

A. Bogliolo, L. Benini, and G. De Micheli. 2000. Regression-based RTL Power Modeling. *ACM Transation on Design Automation Electronic System (TODAES)* 5, 3 (Jul. 2000).

D. Chen, J. Cong, Y. Fan, and Z. Zhang. 2007. High-Level Power Estimation and Low-Power Design Space Exploration for FPGAs. In *Asia and South Pacific Design Automation Conference (ASP-DAC)*.

E. Copty, G. Kamhi, and S. Novakovsky. 2011. Transaction level statistical analysis for efficient micro-architectural power and performance studies. In *Design Automation Conference (DAC)*.

M. Gashler, C. Giraud-Carrier, and T. Martinez. 2008. Decision Tree Ensemble: Small Heterogeneous Is Better Than Large Homogeneous. In *Seventh International Conference on Machine Learning and Applications (ICML)*.

K. Grüttner, P. A. Hartmann, T. Fandrey, K. Hylla, D. Lorenz, S. Stattelmann, B. Sander, O. Bringmann, W. Nebel, and W. Rosenstiel. 2014. An ESL timing amp; power estimation and simulation framework for heterogeneous SoCs. In *International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*.

S. Gupta and F. N. Najm. 2000. Power modeling for high-level power estimation. *IEEE Transactions on Very Large Scale Integration Systems (TVLSI)* 8, 1 (Feb. 2000).

C.-T. Hsieh, Q. Wu, C.-S. Ding, and M. Pedram. 1996. Statistical sampling and regression analysis for RT-Level power evaluation. In *International Conference on Computer Aided Design (ICCAD)*.

C. W. Hsu, J. L. Liao, S. C. Fang, C. C. Weng, S. Y. Huang, W. T. Hsieh, and J. C. Yeh. 2011. PowerDepot: Integrating IP-based power modeling with ESL power analysis for multi-core SoC designs. In *Design Automation Conference (DAC)*.

C. Lattner and V. Adve. 2004. LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation. In *International Symposium on Code Generation and Optimization (CGO)*.

D. Lee. 2017. Learning-Based IP Power Modeling (LIPPo). https://github.com/SLAM-Lab/LIPPo. (2017).

D. Lee, L. K. John, and A. Gerstlauer. 2015a. Dynamic Power and Performance Back-annotation for Fast and Accurate Functional Hardware Simulation. In *Design, Automation & Test in Europe (DATE)*.

D. Lee, T. Kim, Y. Hoskote, L. K. John, and A. Gerstlauer. 2015b. Learning-Based Power Modeling of System-Level Black-Box IPs. In *International Conference on Computer-Aided Design (ICCAD)*.

I. Lee, H. Kim, P. Yang, S. Yoo, E.-Y. Chung, K.-M. Choi, J.-T. Kong, and S.-K. Eo. 2006. PowerViP: SoC power estimation framework at transaction level. In *Asia and South Pacific Conference on Design Automation (ASP-DAC)*.

S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi. 2013. The McPAT Framework for Multicore and Manycore Architectures: Simultaneously Modeling Power, Area, and Timing. *ACM Transaction on Architecture Code Optimization (TACO)* 10, 1, Article 5 (Apr. 2013).

D. Lorenz, K. Grüettner, and W. Nebel. 2014. Data- and State-Dependent Power Characterisation and Simulation of Black-Box RTL IP Components at System Level. In *Euromicro Conference on Digital System Design (DSD)*.

T. Mertens, J. Kautz, and F. V. Reeth. 2007. Exposure fusion. In *Pacific Conference on Computer Graphics and Applications (PG)*.

Nangate. 2017. Open Cell Library. http://www.nangate.com. (2017).

S. Paris, P. Kornprobst, J. Tumblin, and F. Durand. 2009. Bilateral Filtering: Theory and Applications. *Foundations and Trends® in Computer Graphics and Vision (CGV)* 4, 1 (2009).

Y. H. Park, S. Pasricha, F. J. Kurdahi, and N. Dutt. 2007. System level power estimation methodology with H.264 decoder prediction IP case study. In *International Conference on Computer Design (ICCD)*.

Y. H. Park, S. Pasricha, F. J. Kurdahi, and N. Dutt. 2011. A Multi-Granularity Power Modeling Methodology for Embedded Processors. *IEEE Transactions on Very Large Scale Integration Systems (TVLSI)* 19, 4 (Apr. 2011).

M. Pedram. 1997. Advanced power estimation techniques. In *Low power design in deep submicron electronics*, Wolfgang Nebel and Jean Mermet (Eds.). Kluwer Academic Publishers.

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. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research (JMLR)* 12 (2011).

N. R. Potlapally, A. Raghunathan, G. Lakshminarayana, M. S. Hsiao, and S. T. Chakradhar. 2001. Accurate power macro-modeling techniques for complex RTL circuits. In *International Conference on VLSI Design (ICVD)*.

C. A. Ratanamahatana and D. Gunopulos. 2003. Scaling up the Naive Bayesian Classifier: Using Decision Trees for Feature Selection. *Applied Artificial Intelligence (AAI)* 17 (2003).

S. Ravi, A. Raghunathan, and S. Chakradhar. 2003. Efficient RTL power estimation for large designs. In *International Conference on VLSI Design (ICVD)*.

S. Schürmans, G. Onnebrink, R. Leupers, G. Ascheid, and X. Chen. 2015. ESL power estimation using virtual platforms with black box processor models. In *International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*.

S. Schürmans, D. Zhang, D. Auras, R. Leupers, G. Ascheid, Xiaotao Chen, and Lun Wang. 2013. Creation of ESL power models for communication architectures using automatic calibration. In *Design Automation Conference (DAC)*.

Y. S. Shao, B. Reagen, G.-Y. Wei, and D. Brooks. 2014. Aladdin: A Pre-RTL, Power-performance Accelerator Simulator Enabling Large Design Space Exploration of Customized Architectures. In *International Symposium on Computer Architecuture (ISCA)*.

V. Spiliopoulos, A. Bagdia, A. Hansson, P. Aldworth, and S. Kaxiras. 2013. Introducing DVFS-Management in a Full-System Simulator. In *International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOT)*.

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

C. Trabelsi, R. B. Atitallah, S. Meftali, J.-L. Dekeyser, A. Je, H. I. Inria, and A. Jemai. 2011. A model-driven approach for hybrid power estimation in embedded systems design. *EURASIP Journal on Embedded Systems (EURASIP JES)* 1 (Mar. 2011).

G. Wu. 2015. *Performance, Power, and Confidence Modeling of Digital Designs.* Ph.D. Dissertation. University of Texas at Austin.

Z. Zhao, A. Gerstlauer, and L. K. John. 2017. Source-Level Performance, Energy, Reliability, Power and Thermal (PERPT) Simulation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 36, 2 (Feb. 2017), 299–312.

L. Zhong, S. Ravi, A. Raghunathan, and N. K. Jha. 2004. Power estimation for cycle-accurate functional descriptions of hardware. In *International Conference on Computer Aided Design (ICCAD)*.