Data-Driven Power Modeling and Monitoring via Hardware Performance Counter Tracking

Sergio Mazzola[a], Gabriele Ara[b], Thomas Benz[a], Björn Forsberg[c], Tommaso Cucinotta[b], Luca Benini[a,d]

[a]*Integrated Systems Laboratory (IIS), ETH Zürich, Gloriastrasse 35, Zürich, 8092, Switzerland*
[b]*Real-Time Systems Laboratory (ReTiS), Scuola Superiore Sant'Anna, Piazza Martiri della Libertà 33, Pisa, 56127, Italy*
[c]*Department of Computer Science, RISE Research Institutes of Sweden, Isafjordsgatan 22, Kista, 16440, Sweden*
[d]*Department of Electrical, Electronic, and Information Engineering (DEI), University of Bologna, Viale Risorgimento 2, Bologna, 40136, Italy*

Abstract

Energy-centric design is paramount in the current embedded computing era: use cases require increasingly high performance at an affordable power budget, often under real-time constraints. Hardware heterogeneity and parallelism help address the efficiency challenge, but greatly complicate online power consumption assessments, which are essential for dynamic hardware and software stack adaptations. We introduce a novel power modeling methodology with state-of-the-art accuracy, low overhead, and high responsiveness, whose implementation does not rely on microarchitectural details. Our methodology identifies the Performance Monitoring Counters (PMCs) with the highest linear correlation to the power consumption of each hardware sub-system, for each Dynamic Voltage and Frequency Scaling (DVFS) state. The individual, simple models are composed into a complete model that effectively describes the power consumption of the whole system, achieving high accuracy and low overhead. Our evaluation reports an average estimation error of 7.5 % for power consumption and 1.3 % for energy. We integrate these models in

Email addresses: smazzola@iis.ee.ethz.ch (Sergio Mazzola), gabriele.ara@santannapisa.it (Gabriele Ara), tbenz@iis.ee.ethz.ch (Thomas Benz), bjorn.forsberg@ri.se (Björn Forsberg), tommaso.cucinotta@santannapisa.it (Tommaso Cucinotta), lbenini@iis.ee.ethz.ch (Luca Benini)

the Linux kernel with Runmeter, an open-source, PMC-based monitoring framework. Runmeter manages PMC sampling and processing, enabling the execution of our power models at runtime. With a worst-case time overhead of only 0.7 %, Runmeter provides responsive and accurate power measurements directly in the kernel. This information can be employed for actuation policies in workload-aware DVFS and power-aware, closed-loop task scheduling.

Keywords: Power modeling, runtime power estimation, embedded systems, operating systems, Linux kernel

---

1. Introduction

Recent years have seen a dramatic evolution in the embedded and real-time computing landscape, with increasingly demanding requirements. Applications strive for ever-higher computing capabilities and energy efficiency, pushing toward heterogeneous and massively parallel computing platforms [1, 2]. However, since the end of Dennard's scaling, several *walls* have been hit, from power consumption to memory, to hardware overspecialization [3]. With the limits of current silicon technology exposed, pushing for maximum energy efficiency at runtime and in a dynamic fashion is paramount to meet the requirement for high performance within sustainable power budgets [4].

The de facto standard to boost hardware power efficiency at runtime is Dynamic Power Management (DPM), integrated even in today's simplest embedded systems in the form of Dynamic Voltage and Frequency Scaling (DVFS) and clock gating. Through DPM, different processing elements or computing islands can be independently turned off or slowed down based on the phase of the running workload. However, to exploit the full potential of the available power knobs, the software stack must also be able to perform intelligent adaptations based on power measurements. Providing such information at the level of the OS kernel, for instance, allows the *task scheduler* to perform power-aware decisions as to the assignment of computing resources to the running processes [5]. This is essential for applications characterized by real-time constraints running on embedded systems, due to critical misprediction penalties and thermal concerns [5, 6, 7].

For such full-stack, energy-aware dynamic adaptations to be effective, however, *accurate*, *fine-grained*, and *responsive* online power measurements, with negligible overhead on normal system operation, are required to close the control loop [8, 9]. An intuitive solution relies on analog power sensors. How-

ever, as discussed in Section 2.1, they unfortunately pose significant challenges in many practical scenarios [10]. As an alternative to direct power sensing, analytical and data-driven power models have been extensively researched to obtain power measurements better suited for dynamic, online adaptations of hardware and software. It is well-known that the Performance Monitoring Counter (PMC) activity effectively correlates to power consumption [11], enabling accurate, data-driven power modeling for responsive and fine-grained power gauging [8]. However, the complexity of selecting appropriate PMCs with an understanding of the underlying hardware architecture, coupled with the modeling challenges of DVFS, complicates their broader applicability in heterogeneous parallel systems with limited resources and constraining time requirements.

This paper introduces a PMC-based approach to power consumption estimation for modern, DVFS-enabled, heterogeneous systems, extending our previous work on the topic [12]. We devise a low-overhead statistical model for the power consumption of an embedded computing system composed of a host CPU and additional specialized hardware acceleration sub-systems exposing activity counters, a common practice in today's mobile and embedded platforms [13]. We decompose the system into its smaller, more easily approachable sub-systems, and build a lightweight Lookup Table (LUT) of simple power models, independently modeling every sub-system in each DVFS state. The modeling simplicity of the LUT seeks to match or outperform more complex approaches to power estimation, achieving an advantageous trade-off between estimation accuracy and evaluation overhead, together with fine granularity and high responsiveness.

As a second key contribution, we propose *Runmeter*, an architecture-agnostic integration strategy of the model within the Linux kernel that automatizes the collection of PMC samples and the online evaluation of the power model in a lightweight and responsive fashion. The approach is demonstrated with a modern, heterogeneous, DVFS-enabled target platform, the NVIDIA Jetson AGX Xavier board. Considering its CPU and GPU sub-systems, our combined, system-level power model achieves an instantaneous power Mean Absolute Percentage Error (MAPE) of 7.5 % and an overall energy estimation error of 1.3 %. On this platform, the online implementation in Linux exhibits a worst-case overhead of 0.7%, enabling the deployment of aggressive closed-loop power management strategies.

The rest of this paper is organized as follows. Section 2 frames our contributions into the context of its related work, providing background

knowledge and justifications for the approach described in this paper. Section 3 describes our statistical power modeling approach as a generalization of [12], while Section 4 illustrates the architecture of our novel power monitoring framework integrated within the Linux kernel. Finally, Section 5 describes the evaluation of our power model through offline validation and online evaluation.

2. Background & Related Work

[14, 15, 16] present a comprehensive survey of different power modeling approaches and runtime power monitors in the field of embedded and mobile devices. In the following, we mainly focus on the research works related to PMC-based power modeling and online, model-based power monitoring, comparing them with our solution. We summarize the key characteristics in Table 1, providing a comparison with our approach.

2.1. Analog Power Measurement

For power-aware dynamic adaptations to be possible, online power gauging is a requirement. To effectively leverage techniques such as DPM and power-aware task scheduling, the online power gauging must possess the following properties:

1. *accuracy*, in terms of time resolution and sensitivity, to properly feed the power control loop;

2. *responsiveness*, to promptly reflect the hardware activity profile and provide stable feedback for the control loop;

3. *fine granularity*, in terms of introspection into the power consumption of individual hardware sub-systems (i.e., *decomposability*) and task-level power budgeting.

Several off-the-shelf system-on-a-chip (SoC) platforms come equipped with built-in power sensors, although not integrated on the same die of the SoC due to technological reasons. Hence, they can rarely provide the level of introspection of individual hardware sub-systems. For the same reason, off-chip parasitics pollute their measurements with longer transients, impacting the responsiveness of their measurements [17]. Their latency is further affected by the communication channel with the host, usually implemented by a serial protocol such as I2C, which does not match the speed of the digital

domain. Additionally, due to their physical size and deployment costs, analog gauges often suffer from limited scalability in large-scale or densely integrated systems [10].

Although unsuitable for reliable, power-driven actuation policies, built-in analog sensors do not require external equipment for current and voltage measurements, and they can be programmatically and reliably driven. Therefore, they prove useful to build accurate, fine-grained, and responsive PMC-based power models through the approach showcased in Section 3. This is demonstrated in Section 5.4.

2.2. PMC-Based Power Modeling

PMC-based statistical power models have been a hot research topic for the last 20 years, spanning all computing domains from embedded computing devices at the edge to data centers in the cloud. Being typically accessible via memory-mapped registers, PMCs are cheap to use, and their readings are fast and reliable. As part of the digital domain, PMC activity promptly reflects the current state of the hardware resources, exposing desirable responsiveness properties. PMCs also provide a high degree of introspection into individual hardware sub-systems [18], resulting in highly decomposable PMC-based power models [8].

However, power estimation through PMCs raises several challenges. Modern computer architectures, even in the embedded domain, expose hundreds of countable performance events [19, 20]. Hence, the parameter selection for a robust statistical power model often requires considerable knowledge of the underlying hardware. Growing parallelism and heterogeneity, together with the frequent lack of open documentation, further amplify the challenge. A careful choice of the model parameters is necessary for several additional factors: first, Performance Monitor Units (PMUs) can simultaneously track only a limited number or combinations of performance counters [21, 20, 22]. Second, the amount of model predictors directly impacts evaluation overhead, which must be small for practical actuation strategies and minimal interference with the system's regular operation and time constraints. DVFS determines an additional layer of modeling complexity, as hardware behavior at varying frequencies has to be considered. To the best of our knowledge, the approach we propose in Sections 3 and 4 is the first one to holistically address all the mentioned challenges.

Table 1: Comparison between representative works in the literature of PMC-based power modeling. Note that accuracy metrics, such as the power MAPE, are platform- and application-dependent. Due to differences in the experimental set-ups, we report the MAPE to indicate whether each approach delivers the required accuracy for the target application, rather than as a comparison across different methodologies.

	Heterogeneity	Generality	Automation	Min architectural knowledge	Lightweight model	DVFS support	Decomposability	Runtime monitoring	Power MAPE
Walker et al. (2016) [23]	CPU only	ARM cores	✓	✗	✓	✓	✗	✗	3-4%
Yoon et al. (2017) [13]	✓	Android	≈	≈	✓	✓	✓	✓	5.1%
Wang et al. (2019) [24]	iGPU only	✗	✗	✗	✓	✗	✗	✗	3%
Mammeri et al. (2019) [25]	✓	mobile	≈	≈	✗	✗	✗	✗	4.5%
Tarafdar et al. (2023) [26]	✗	data centers	?	✗	✓	?	✗	✓	4.7%
This work	✓	✓	✓	✓	✓	✓	✓	✓	CPU 3-4.4% GPU 6-8%

2.2.1. Bottom-up and top-down modeling

Bertran et al. [27] identify two families of PMC-based power models, depending on their construction: *bottom-up* and *top-down*. *Bottom-up* approaches rely on extensive knowledge of the underlying architecture to estimate the power consumption of individual hardware sub-systems. Although the pioneering works of this field fall into this category [18, 8], their results highlight the limited applicability of bottom-up power models, which are closely tied to a reference architecture. Recent research further confirms this limitation [28].

Top-down approaches target simple, low-overhead, and more generally

applicable models, often black-boxing the platform internals. Over the years, this approach has been refined from the usage of few, manually selected PMCs [11] to the employment of more elaborate procedures for PMC selection and support for parallel [29, 30] and heterogeneous [31] platforms. However, no past research investigates a combination of accurate and low-overhead models addressing DVFS without requiring expert architectural knowledge.

In the context of CPU power modeling specifically targeting mobile and embedded platforms, Walker et al. [23] employ a systematic technique for PMC selection and train power models for the ARM A7 and A15 embedded processors. However, only one trained weight is used to estimate the power consumption at any DVFS state. As no information on the employed DVFS states is available, it is not possible to assess whether such a modeling choice can avoid large inaccuracies due to the limited number of parameters.

Yoon et al. [13] propose a power model for mobile SoCs solely based on the utilization metrics provided by the Android kernel, which abstracts the model deployment from the specific architecture. However, to construct the model, the authors individually characterize each sub-system by leveraging in-depth architectural knowledge. Moreover, the single utilization parameter often fails to grasp the different phases of the running workload, which results in a larger estimation error with workloads showing higher variability. Yoon et al. also implement an online monitor deploying their power model, which reports an overhead of up to 4.5 % of CPU time at a 1 Hz sampling frequency. In comparison, Runmeter achieves up to 0.7 % overhead with a sampling period of 10 Hz.

Top-down power modeling approaches are also applied to GPUs. Wang et al. [24] analyze the power consumption of an AMD Integrated GPU, carefully studying its architecture and selecting the best PMCs to build a linear power model. While they achieve a MAPE below 3 %, the model is manually fine-tuned for a single system, requiring its expert architectural knowledge, and DVFS is not taken into account. Recent works also resort to deep learning for creating accurate black-box power models: Mammeri et al. [25] train an Artificial Neural Network (ANN) with several manually chosen CPU and GPU PMCs, achieving a MAPE of 4.5 %. However, neural networks generally require a number of multiply-accumulate operations two orders of magnitude higher than for a linear model, representing a non-negligible runtime overhead. Potentially long training time, risk of overfitting, deployment challenges, and lack of decomposability are additional drawbacks of this approach.

Tarafdar et al. [26] also propose several power modeling techniques based

on multi-variable linear regression, Support Vector Regression, and ANN. While their approach is statistically sound, they conceive it as a solution for data centers. Therefore, no fine-grained power information about the computing platform is made available. Moreover, the model parameter selection happens a priori and is not correlated to the modeled platform. Their best power model, which features an evaluation overhead in the order of the microseconds, shows an average power estimation error of at most 4.7%.

2.2.2. Contributions

Our proposed model shares its decomposability and responsiveness with bottom-up approaches but resorts to top-down modeling for individual subsystems: we trade a lower per-component introspection for a systematic modeling procedure requiring very little architectural knowledge and minimal human intervention. In addition, we carefully address the platform heterogeneity and DVFS capabilities by introducing a LUT-based approach that employs individual, low-overhead linear models for each sub-system and for each DVFS state. With the aim to enable real-time, in-kernel power estimation on embedded and edge systems, simplicity, determinism, and overhead are critical. While techniques such as quantized deep learning could address inference overhead, they open up challenges in training, deployment, and reproducibility in constrained OS environments. On the other hand, the simplicity of linear models makes them particularly suitable for deployment in embedded systems, possibly with real-time constraints, and as part of an Operating System (OS) kernel, thanks to their low overhead when evaluated at runtime.

2.3. Online Model-based Power Monitoring

Various tools have been proposed for online, model-based estimation of power consumption through PMC sampling. Commonly, runtime monitoring is implemented by simply sampling the PMCs with a fixed periodicity [32, 33].

One of the most popular open-source tools for online PMC sampling is PMCTrack, developed by Saez et al. [34]. PMCTrack can monitor per-system, per-CPU, and per-process PMCs directly within the Linux kernel. However, targeting general-purpose use cases, many aspects of its implementation are not suited for real-time tasks, that pose different requirements from those of `SCHED_OTHER` tasks. As an example, PMCTrack may delay the generation of a PMC sample related to a task until it is scheduled for execution by the task scheduler. Such a delay is detrimental to the responsiveness of a

power-monitoring tool. Other tools based on PMCTrack modify its source to generate samples at every context switch, albeit their target use-case again differs from that of real-time tasks [35], making them unsuitable for our study.

On the other hand, with Runmeter, we provide a responsive and reliable mechanism to monitor the evolution of PMCs in use cases that include real-time tasks. As discussed in Section 4, we achieve this by implementing a moving sampling window for PMC collection, featuring fully configurable time resolution and sensitivity. This results in responsive power readings that do not sacrifice estimation accuracy, which depends on the configurable window size [33]. By deploying our low-overhead power models in Runmeter, we allow the collection of online accurate power measurements with minimal overhead and sub-system-level introspection at the granularity of individual tasks.

3. Data-Driven Power Modeling

This section describes our automated, data-driven approach to the training of a DVFS-aware, low-overhead power estimation model for heterogeneous, embedded platforms. Given a generic target platform composed of one or multiple sub-systems, we provide a systematic approach to its characterization (i.e., model parameter selection) based on extensive profiling of the exposed PMCs, rather than on microarchitectural details. This results in an accurate, responsive, and low-overhead power model for the entire platform and its individual sub-systems, at the desired operating frequencies.

3.1. The systematic, data-driven methodology

For the purpose of this section, we consider a generic computing platform composed of a set D of individual sub-systems d. Each sub-system d has a set F_d of possible DVFS states, each one characterized by an operating frequency f_d. We define $D^* \subseteq D$ as the subset of sub-systems that we target for power modeling. Furthermore, for each $d \in D^*$, we define $F_d^* \subseteq F_d$ as the subset of d's DVFS states that we consider. Both D^* and F_d^* are user-defined parameters that might vary based on the use-case. For each sub-system d, up to N_d distinct performance events can be tracked at the same time, i.e., d features N_d PMCs. In the following, we refer to an individual performance event exposed by the sub-system d and tracked by its i-th PMC as x_i, with $i = 1 \,..\, N_d$. The set X_{d,f_d} of all x_i of a sub-system d, operating at f_d, represents the set of input independent variables, or the *predictors*, of our models.

We propose a methodology to heuristically select the best X_{d,f_d} set for each sub-system/frequency in terms of overhead and accuracy, subsequently training its related weights W_{d,f_d}. The individually generated power models $P_d(X_{d,f_d}, W_{d,f_d})$ are simple, linear models that we compose into a Lookup Table (LUT), effectively grasping the different platform behaviors at varying operating frequencies.

$$LUT[d, f_d] = P_d(X_{d,f_d}, W_{d,f_d}) \quad \text{for } d \in D^*,\ f_d \in F_d^* \tag{1}$$

The overall system power consumption P_{tot} is computed by *reduction sum* of the LUT along the sub-system dimension d, after fixing a f_d for each sub-system.

$$P_{tot} = \sum_{d \in D^*} LUT[d, f_d] = \sum_{d \in D^*} P_d(X_{d,f_d}, W_{d,f_d}) \tag{2}$$

The power consumption of a digital system has a non-linear dependency on its operating voltage and frequency. Our LUT-based approach allows us to break it down into the individual contributions of each sub-system, linearizing their power models. This greatly simplifies the model generation and evaluation, which favors its deployment within embedded, real-time systems and its applicability to different platforms [13]. However, our decomposition is based on the assumption of independence among power consumption of different sub-systems. Such an assumption is justified by the limited accuracy increase of the non-linear model at the cost of a much higher overhead [36].

3.2. Analytical Model Building & Benchmarks Selection

As a first step toward the construction of a system-level LUT, we associate the generic sub-systems d with the expression P_d of a power model based on generic performance event information (Figure 1, ❶). Thanks to the LUT-based approach, the frequency f_d is factored out. Therefore, the individual power models are reduced to linear combinations of the PMC samples and the trained weights.

$$P_d(X_{d,f_d}, W_{d,f_d}) = L_d + \sum_{i=1}^{N_i} \left(\frac{1}{T} \cdot x_i \right) \cdot w_i \tag{3}$$

$$\text{for } d \in D^*,\ f_d \in F_d^* \text{ and } \tfrac{L_d}{w_i} \in W_{d,f_d},\ x_i \in X_{d,f_d}$$

The weight L_d is used to capture the constant component of the power consumption, i.e., the leakage, while the PMC-dependent terms vary based on

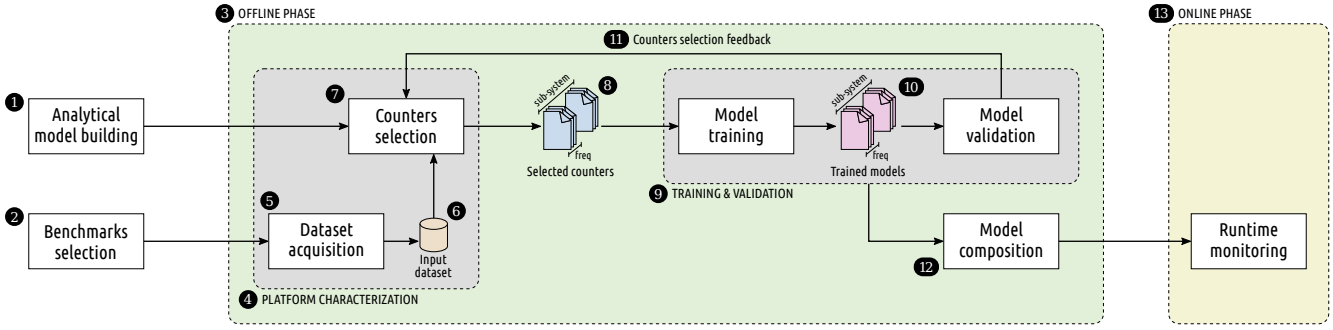


Figure 1: Scheme of the proposed data-driven, automatic power modeling approach for DVFS-enabled heterogeneous platforms.

the hardware activity, modeling the dynamic power. The factor $1/T$ normalizes the raw PMC samples x_i with respect to the sampling period T of the dataset traces: training the model on PMC rates rather than absolute values addresses sampling jitter and simplifies time-rescaling for model evaluation at arbitrary time resolutions.

The model generation is also based on the careful choice of a representative set of workloads for the platform (Figure 1, ❷). In the first place, complete coverage of all targeted sub-systems is required to address the platform heterogeneity fully. Secondly, for each sub-system, the workloads should be diverse enough to induce a broad range of behaviors for a dataset-independent result. The selected benchmarks are employed to build a dataset for platform characterization, model training, and model validation, containing the activity traces of the workloads, i.e., the PMC samples.

3.3. Platform Characterization

Given the P_d mathematical model for each sub-system d, we define *platform characterization* the process of model parameter selection (Figure 1, ❹). In other words, with the platform characterization, we define the set X_{d,f_d} of performance events, which will be used to model the dynamic power of each sub-system d at each frequency f_d.

Individually for each sub-system d and frequency f_d, we perform a one-time correlation analysis between all of its local PMCs and the sub-system power consumption, looking for the X_{d,f_d} that achieves the most convenient trade-off in terms of model accuracy and estimation overhead under the constraints imposed by the PMU limitations. Note that different frequencies of the same sub-system d might be assigned with different PMCs, which effectively models

DVFS with simpler, linear models. This characterization process is meant as an automatic, data-driven alternative to methodologies requiring expert microarchitectural knowledge, such as manual PMC selection and analytical power modeling [37]. Given the sub-system d, its characterization involves the following steps:

1. for each DVFS state $f_d \in F_d^*$, we profile *all* performance events exposed by d while tracing d's power (Figure 1, ❺); as time-correlated PMC and power measures are only required for post-mortem traces, simple synchronization techniques can be used depending on the nature of the power sensor [31, 23];

2. we normalize the PMC samples with respect to the sampling periods to overcome sampling jitter;

3. we compute a Linear Least Squares (LLS) regression of each event's activity trace over its related power measurements, for each f_d; we discard events with a p-value above *0.05* as not reliable for a linear correlation;

4. individually for each f_d, we sort the remaining events by their Pearson Correlation Coefficient and select the best-scoring ones that can be profiled simultaneously (Figure 1, ❼ and ❽).

To compose X_{d,f_d}, it is usually enough to select the desired number of best-scoring performance events. A higher number of model parameters, within the limit of overfitting, usually corresponds to higher model accuracy, but also larger evaluation latency. The optimal number of PMCs with respect to model accuracy can be defined by iteratively considering the estimation error results from the model evaluation step (Figure 1, ⓫). On the other hand, this step has to consider the limitations of the platform's PMU, which come in two forms:

- PMUs have a limited number of PMCs to track events. For example, typical embedded ARM CPUs feature up to four or six individual counters, that can be mapped to freely selectable or fixed performance events. High-performance CPUs, like the *Hisilicon Kunpeng 920* processor, can track up to twelve events per core domain [21].

- some performance events are mutually exclusive, i.e., *incompatible*.

Incompatible events can be tracked in a time-sharing fashion through *counter multiplexing*. However, such an approach increases tracking overhead and introduces interpolation errors, decreasing estimation accuracy [21]. Targeting low-overhead, real-time model estimation, we opt for a lower-complexity solution that avoids PMC multiplexing. To this end, we devise a PMU-aware heuristic identifying the subset of compatible counters that provide the highest power estimation accuracy while complying with PMU constraints (Figure 1, **11**) [12].

3.4. Training, Validation, and System-level Model

With the sets of counters X_{d,f_d} defined during platform characterization, we compose the LUT of Equation (1) by individually training the linear power model $P_d(X_{d,f_d}, W_{d,f_d})$ of each sub-system $d \in D^*$ for each $f_d \in F_d^*$ (Figure 1, **9**). The output of each training step is a set of weights W_{d,f_d} (Figure 1, **10**). To train each individual $P_d(X_{d,f_d}, W_{d,f_d})$, we perform a Non-Negative Least Squares (NNLS) linear regression of the PMCs rates over the power measurements, obtaining the set of non-negative weights W_{d,f_d}. Compared to unconstrained LLS, non-negative weights are physically meaningful and prove to be robust to multicollinearity, which makes our simple models less prone to overfitting. We subsequently validate each individual $P_d(X_{d,f_d}, W_{d,f_d})$.

After individual training and validation, we combine all the individual sub-system models (Figure 1, **10**) into the system-level power model (Figure 1, **12**) defined by Equation (2). Under the reasonable assumption of power consumption independence among sub-systems [36], such an approach relieves us from profiling all possible combinations of sub-systems' frequencies. This simplifies and accelerates the platform characterization, ultimately generating a simple linear model that is more robust to overfitting, accurate, lightweight, and decomposable.

The complete model can finally be used online (Figure 1, **13**) to monitor the instantaneous power consumption of the entire system and its sub-systems. To do so, it is enough to keep the weights W_{d,f_d} available at runtime for each $(d,\ f_d)$ combination of interest. Due to our modeling methodology, this has a negligible memory footprint. After acquiring the PMC samples for the set of model parameters X_{d,f_d}, the power model can be efficiently evaluated with the small number of multiply-accumulate operations required by its linear expression. Our proposed framework for the online deployment of our power model is the object of the next section.

4. Online Monitoring and Kernel Support

The target of our work is to enable power awareness in crucial functionalities, such as task scheduling and resource allocation, when running with real-time applications on resource-constrained devices. The modeling approach discussed in Section 3 results in a complete system-level power model that can provide accurate and introspective power estimates with minimal overhead. An online monitoring framework that integrates the proposed model is also essential to our target. For this, we propose Runmeter, an online monitoring framework integrated in the Linux kernel[1].

Runmeter supports the runtime estimation of system- and task-level metrics, including power and energy consumption, through PMC tracking. As such, it provides the infrastructure to flexibly collect PMC samples with minimal overhead and evaluate the model presented in Section 3, exposing its estimates to the Linux scheduler. The modular design of Runmeter abstracts its implementation from the specifics of the hardware architecture. Only a minimal subset of its components must be re-implemented to support different target platforms, sub-systems, and tracked metrics. As a case study, we leverage the framework to implement support for online power estimation and monitoring of the CPU sub-system.

4.1. Runmeter Kernel Module

Once loaded into the kernel, the Runmeter Kernel Module hooks to strategic callbacks to trace and collect running statistics on a selection of the available PMCs, according to the result of the platform characterization (Section 3.3). Since a different set of counters $X_{\text{CPU},f_{\text{CPU}}}$ can be selected to model the evolution of the platform depending on the DVFS state f_{CPU}, the module selects the correct PMCs to track according to the model LUT (Equation (1)). The module also subscribes to the CPU frequency governor (`CPUFreq`) to be notified of each change of frequency so that it can dynamically reconfigure the set of tracked PMCs for each CPU core. When tracking is enabled, the kernel module generates a new PMC sample on each CPU core whenever one of the following events occurs:

- a context switch, in which case a new sample is always generated, or

[1] Runmeter Framework is an open source project; its homepage is at `https://gitlab.retis.santannapisa.it/ampere/runmeter`. Here, users can find all the necessary tools to build and deploy the framework on supported platforms.

- a user-configurable number of scheduler ticks since the last sample was produced on that core.

The first trigger allows Runmeter to collect PMC statistic on a per-task basis and derive power estimates with task-level granularity. The second trigger, on the other hand, provides an upper bound to the inter-arrival time between two consecutive PMC samples. This guarantees that tasks hogging the CPU do not interfere with the monitoring. Since this bound is expressed in terms of scheduler ticks, its granularity depends on the `CONFIG_HZ` Linux kernel option. The dependence on scheduler ticks also prevents unwanted activation of Runmeter during deep idle states.

Proper selection of this upper bound is key to ensuring the desired responsiveness when monitoring CPU counters. The basic PMC sampling mechanism provides the accumulated value of the event counter since its last reading. In this case, a small sampling period negatively impacts the information collected by the PMCs: since each sample is tied to a single task, it is difficult to derive any meaningful data about the overall platform status from it. On the other hand, with a large sampling period, the read-out value is updated less frequently, which is detrimental for actuation policies requiring high responsiveness [33].

As a trade-off, we devise a *moving-window* approach that decouples the PMC sampling period from their observation window. The moving window allows us to obtain PMC statistics accumulated over an arbitrarily long window and updated at an arbitrarily fine time granularity. To implement the moving-window approach, we instantiate a *window buffer* for each PMC. Each buffer stores a user-configurable number of the most recent PMC samples. The value of each PMC over the whole window is tracked by summing up all samples in the buffer. This information is updated each time the window moves forward (i.e., a new sample is available). We refer to this value as *synthetic PMC sample*. Such a moving buffer also serves the purpose of aggregating the per-task PMC samples to obtain core-level metrics.

Consuming synthetic samples provides more meaningful PMC data for the metrics to be estimated. This comes at the cost of the additional processing of each PMC's window. However, as shown in Section 5, this additional processing introduces negligible overhead in practice. Moreover, the aggregation of samples at the core level is required regardless of the moving-window mechanism, making the relative cost of this step minimal.

4.2. In-Kernel CPU Power Model

Synthetic PMC samples provide visibility over a time defined by the window size, but are updated at a rate defined by the PMC sampling period. Components like an online CPU power (or energy) monitor are required to re-evaluate their estimates at each update of the synthetic samples. A high degree of responsiveness in the monitoring, useful for robust actuation, can be achieved when the model evaluation time can keep up with the stream of synthetic samples.

The model we present in Section 3 retains high accuracy despite its low computational complexity, which empowers it to actually keep up with the stream of samples. As a case study, we deploy it in the Linux kernel through the infrastructure provided by the Runmeter Framework. The CPU power monitor in Runmeter implements, for each CPU DVFS state, the following model, extending Equation (3) to support multiple CPU cores:

$$P_{\text{CPU}} = L_{\text{CPU}} + \underbrace{\sum_{i=1}^{\#\text{cores}} \sum_{j=1}^{N_{\text{CPU}}} \left(\frac{1}{T'} \cdot x_{ij} \right) \cdot w_{ij}}_{= \frac{1}{T'} \sum_{i=1}^{\#\text{cores}} \sum_{j=1}^{N_{\text{CPU}}} x_{ij} \cdot w_{ij}} \tag{4}$$

The weights L_{CPU} and w_{ij} are fractional values, but the usage of floating-point arithmetic within the Linux kernel is problematic and expensive. For this reason, we use fixed-point arithmetic to implement the in-kernel power model, supported by the negligible loss of dynamic range and precision that we evaluate in Section 5.4.

The factor $1/T'$ normalizes the value of each synthetic sample with respect to the width of the user-configured observation window T'. T' might indeed differ from the sampling period T of the model training dataset (Equation (3)). Thanks to the linearity of our models, we can perform the normalization by factoring out $1/T'$ and operating only one multiplication after the summation. This achieves arbitrary time-rescaling of the model with negligible overhead.

4.3. Runmeter and PMCTrack

Runmeter Framework's kernel components are based on the implementation of PMCTrack [34], albeit some fundamental differences branch away from the original implementation due to the specific requirements of our use

case. This section clarifies the similarities and the key differences between the two tools, motivating our design choice.

Runmeter's current implementation exploits a kernel patch to insert callbacks to its PMC sampling mechanism, detailed in Section 4.1. We implement the rest of the Runmeter Framework as a dynamically loadable kernel module that hooks to such entry points. The Runmeter Kernel Patch is equivalent to the one provided by PMCTrack. PMCTrack additionally provides a mechanism to dynamically inject entry points for the kernel module without a kernel patch [38]. Such a mechanism is based on *dynamic ftrace*, which, from Linux kernel v5.9 on, provides a stable interface to inject the hooks required by PMCTrack. As reported in Section 5, our target platform relies on the Linux kernel v4.9, which therefore requires a patch to support the Runmeter kernel module. Nevertheless, Runmeter can seamlessly leverage the very same mechanism for a patch-less implementation with more recent kernel versions.

On the other hand, PMCTrack and Runmeter Framework substantially differ in their respective kernel modules. PMCTrack's users can register *monitoring modules* as consumers of PMCTrack-managed PMC samples [34]. However, PMCTrack's limited tracing modes prevent us from implementing Runmeter as a monitoring module for PMCTrack. In particular, PMCTrack implements three different tracing modes: 1. an *event-based mode*, that generates PMC samples when one of the counters reaches a configured threshold, and 2. two different variants of a *timer-based mode*, called time-based sampling (TBS), that generate samples periodically. Due to the use case targeted by Runmeter, in this work we focus on a timer-based sampling approach.

The first TBS mode provided by PMCTrack is *per-task tracing*. Per-task tracing generates new PMC samples periodically, based on a callback executed by the scheduler tick (as in Runmeter), or when tasks are selected for execution, rather than switched out. In the latter case, the PMC sample generation is delayed until the task is selected again. Furthermore, each traced task has to be configured individually, which makes it challenging to trace all the tasks running on the system. While this enables PMCTrack to profile different performance events for each task, it represents a drawback for Runmeter, whose event selection is dictated exclusively by the operating frequency. This limitation is common to many tracing tools [35]. Runmeter, on the other hand, targets a rapid swap among different sets of PMCs whenever the operating frequency changes, to align to the power model's LUT (Equation (1)) determined during the platform characterization without

looping through all the individually traced tasks.

PMCTrack also implements a system-wide TBS mode, where a kernel timer dictates the sampling of PMCs on a per-core basis [34]. This mode provides a predictable sampling period suitable for real-time power estimation, but it generates PMC samples on a per-core basis. This prevents monitoring modules from collecting task-level metrics. In contrast, Runmeter's hybrid sampling strategy generates PMC samples on a per-task basis, subsequently aggregating them into per-core samples, granting a predictable periodicity in the PMC collection. Furthermore, being driven by scheduler ticks rather than a kernel timer, Runmeter periodic activation would not wake up a core in a deep idle state, polluting power measurements, as opposed to PMCTrack system-wide TBS mode.

Motivated by our requirements, we leverage PMCTrack's abstraction of the low-level PMC sampling and execution hooks while re-implementing the PMC collection and delivery to consumers, tailoring it to our specific requirements.

5. Evaluation

In this section, we evaluate the holistic power modeling approach discussed in Section 3, and its in-kernel implementation within the Runmeter online monitoring framework described in Section 4.

5.1. Experimental methodology

The target platform for our experiments is an NVIDIA Jetson AGX Xavier, powered by the Xavier SoC [19]. It is a highly parallel and heterogeneous SoC provided with an 8-core 64-bit ARMv8.2 CPU, a 512-core NVIDIA Volta GPU, and several additional accelerators for deep-learning, computer vision, and video encoding/decoding. With many DVFS states available for its sub-systems, this platform represents a challenging state-of-the-art target to validate our approach. In particular, the single CPU island on the platform can be clocked at 29 different discrete frequencies between 115 MHz and 2.3 GHz, while the GPU has 14 available DVFS states between 115 MHz and 1.4 GHz. For the evaluation of our power modeling approach, we target the CPU and GPU sub-systems,

$$D^* = \{CPU,\ GPU\}$$

considering the following DVFS states:

$$F^*_{\text{CPU}} = \{730\,\text{MHz},\ 1.2\,\text{GHz},\ 2.3\,\text{GHz}\}$$
$$F^*_{GPU} = F_{GPU} = \{\text{all 14 from } 115\,\text{MHz to } 1.4\,\text{GHz}\}$$

To build the input dataset, we profile several workloads based on the considerations of Section 3.2. For the CPU, we employ the OpenMP benchmarks from the Rodinia 3.1 heterogeneous benchmark suite [39] in several multi-thread configurations, in addition to further synthetic benchmarks targeting static stress test of common compute- and memory-bound patterns, such as `memcpy` [12]. For the GPU, we employ the CUDA benchmarks from Rodinia. To average out possible interference in our measurements, such as unpredictable OS activity, each workload is profiled 3 times.

PMC samples are acquired in a continuous, periodical mode with a sampling period of 100 ms. During each sampling period, power measures of the CPU and GPU sub-systems are also acquired from the INA3221 built-in power monitors [40]. This grants the time correlation needed for an effective correlation analysis and training [41]. We find that collecting more than one sample per 100 ms does not capture any additional information due to the electrical inertia of the built-in current sensors.

As discussed in Section 2, typically, built-in power monitors are not robust tools for online, power-aware actuation policies. This is mainly due to their speed, coarse granularity, and low resolution, which for the Xavier is limited to about 200 mW. However, they are helpful for building datasets to achieve higher introspection, time granularity, and responsiveness enabled by PMC-based power models, as proved in Section 5.

5.2. Offline Platform Characterization and Modeling

This section discusses the result of the power model generation (Section 3) of the individual CPU and GPU sub-systems for the NVIDIA Jetson AGX Xavier case study.

5.2.1. Sub-system Characterization

For the CPU sub-system, the results of the platform characterization suggest that the power consumption of the cores is highly correlated, depending on the selected DVFS state, with the number of cycles during which the cores are not power-gated, the number of retired instructions, the floating point activity, and various cache-related events. The ARM PMU always exposes

the CPU active cycle counter. For the remaining power model parameters, we consider the three best counters for each frequency, as the maximum allowed by the PMU. From our experiments, all selected performance events are compatible with each other.

For the GPU sub-system, our results expose multiple incompatibilities among the performance events that best correlate with the power profile. To be able to simultaneously track the best model parameters at runtime, we adopt the PMU-aware heuristic described in Section 3.3, to identify the viable set of compatible performance events. We conclude that events related to L2 cache utilization and warp execution best correlate with the power consumption of the GPU. Additionally, through the validation step, we find that a number of eight PMCs per frequency is the optimal trade-off between model evaluation time and the power estimate accuracy.

5.2.2. Sub-system Modeling and Validation

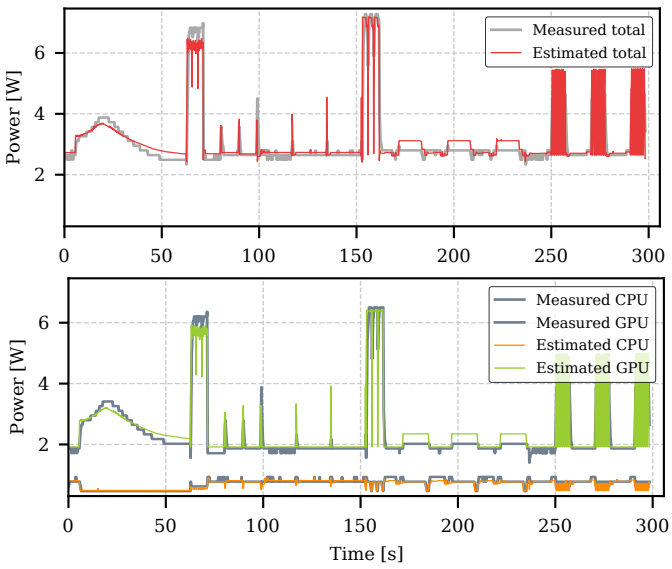


Figure 2: Instantaneous power estimate over the validation set for the system-level power model (on the left) and its breakdown into the individual sub-system estimates (on the right), with $f_{\mathrm{CPU}} = 1.2\,\mathrm{GHz}$, $f_{\mathrm{GPU}} = 830\,\mathrm{MHz}$.

For the CPU, we adopt a NNLS regression to individually train a linear model based on Equation (1) for each frequency. We employ four independent variables per core, i.e., the three configurable PMCs for each frequency and

the cycle counter. Out of our input dataset, we use a random selection of 70 % of the total data for training and the remaining 30 % for validation. In terms of instantaneous power accuracy, the model achieves a Mean Absolute Percentage Error (MAPE) between 3 % and 4.4 % based on the frequency, with a standard deviation of approximately 5 %. When employed to estimate the energy over the entire validation set, our model achieves a maximum error of 4 %, delivering an equal or higher accuracy as reported by the previous state-of-the-art work. It shall be noted that direct comparisons on the same hardware would require adapting and updating the previous works to the platform targeted by our experiments. These adaptations could determine accuracy degradations, whose attribution to limitations of the original methodologies could be debatable. On the other hand, as stated in Table 1, comparing the results achieved on the original targets serves as an indication that the proposed automatic and data-driven methodology delivers results within the expected level of accuracy for relevant applications, e.g., power-aware task scheduling.

For the GPU, we likewise train the Equation (1) for each of the 14 GPU frequencies with a NNLS linear regression. We use a 70% and 30% ratio for the training and validation set. Comparing the instantaneous power consumption estimation with the data measured on the real platform, we obtain a MAPE between 6 % and 8 %, depending on the frequency. The standard deviation over all frequencies is approximately 8 %. The maximum energy estimation error over the full validation set is 5.5 % over all frequencies, with an average of 2.2 %.

5.3. Combined Model Evaluation

After building, training, and validating the CPU and GPU power models individually, we combine them to obtain a system-level power model for every possible combination of $f_{\text{CPU}} \in F^*_{\text{CPU}}$ and $f_{\text{GPU}} \in F^*_{\text{GPU}}$, corresponding to the LUT of Equation (1). Figure 2 shows how our decomposable power model can effectively track the instantaneous power consumption of the system over time. The achieved instantaneous power MAPE of the final, combined model has an average of 8.6 % over all CPU and GPU frequency combinations. Regarding energy, the model reaches an average estimation error of 2.5 %.

Our results highlight that the estimation error of the combined model is higher when f_{CPU} and f_{GPU} diverge from each other. In particular, when f_{GPU} is very low compared to f_{CPU}, the CPU may stall waiting for the offloaded computation. Our power model is not capable of capturing such

behavior, which depends on the interaction among different sub-systems, due to our assumption of sub-system independence (cf. Section 3). On the other hand, a real use case where such a scenario occurs is highly unlikely due to its inefficiency. As a consequence, restricting our evaluation to real scenarios, our assumption of sub-system independence is still valid: by considering only reasonably close CPU and GPU frequencies, in particular $f_{\mathrm{GPU}} > 600\,\mathrm{MHz}$, we report an instantaneous power MAPE of 7.5 % and energy estimation error of 1.3 %, with a maximum of 3.1 %.

This accuracy must be interpreted relative to the precision of our INA3221 reference sensor, which itself has a power resolution of 200 mW (around 3 % of the measurements in our experiments) [40]. Beyond these quantitative results, it is important to acknowledge further practical sources of uncertainty that inherently limit the achievable accuracy of PMC-based models, along with possible mitigations. Because we train our models directly on the target device, fabrication-time process variations and normal supply-voltage fluctuations are inherently captured. Likewise, temperature-dependent leakage is reflected in our dataset, and we observe negligible drift across the temperature swings imposed during the intensive platform characterization and model training. Finally, longer-term aging effects, which develop over months or years, can be corrected via periodic retraining using our lightweight, automatic approach without requiring a full recharacterization.

5.4. CPU Power Monitoring with Runmeter

To evaluate our online monitoring framework, Runmeter, we integrate it into the kernel of the Linux distribution running on the NVIDIA Jetson AGX Xavier. Then, through Runmeter, we implement the power monitor discussed in Section 4 with support for the CPU sub-system, collecting PMC samples with 10 Hz sampling period. In all experiments, the target platform uses a patched version of the NVIDIA Jetson Linux kernel that includes the entry points for the Runmeter module. The most recent version of said kernel at the time of our evaluation is v4.9.253[2].

In this section, we discuss the impact of the fixed-point implementation of our power model, which is necessary for the in-kernel implementation. Subsequently, we validate the power estimations resulting from the online monitoring and evaluate its overhead.

[2] `https://gitlab.retis.santannapisa.it/ampere/runmeter/kernel-jetson`

5.4.1. Fixed-point Approximation Error

In this section, we evaluate the approximation error introduced by our fixed-point implementation of the power model described in Section 4, necessary to integrate it as part of a Linux kernel module. For the fixed-point implementation, we use 64-bit integers, assigning the 29 less significant bits to the fractional part. To analyze the approximation error, we collect the data published by the Runmeter Kernel Module and feed them to a user-space C++ checker procedure. The checker evaluates the model through the floating-point and the fixed-point implementations, measuring the deviation. For this evaluation, we use the same validation set discussed in Section 5.2.

Figure 3 shows the distribution of such a deviation. From our extensive evaluation, the maximum absolute approximation error is about 17 mW; the mean error, however, is only of about 0.17 mW. The maximum percentage error is always below 0.8 % of the power consumption estimated using floating-point arithmetic, with a mean error of about 0.015 %. Given the negligible magnitude of the error introduced by the fixed-point implementation, we conclude that this approximation does not impact the accuracy of the model in any meaningful way.

5.4.2. Online Power Estimation Accuracy

Employing the fixed-point implementation of the CPU power model validated in the previous section, we integrate the power monitor in the Runmeter Framework. We then log the online estimates computed at runtime by the power model to later perform post-mortem analysis. Therefore, differently from what analyzed so far, the power estimates reported in this section are computed directly at runtime as soon as new PMC samples are available. The profiled workloads are the same as the validation set discussed in Section 5.2.

Figure 4 shows the Absolute Percentage Error (APE) distribution of the energy estimation provided by the in-kernel model when compared against the value collected from the onboard analog sensor. The maximum APE registered over all our experiments is around 29 %, the error at 90th percentile is around 20.8 %, and the MAPE is around 9 %.

According to our experiments, the majority of the estimation error accounted for during such an evaluation is to be attributed to very specific time frames when the phase of the workload abruptly changes. Such behavior is visible in the example CPU power profiles depicted in Figure 5. On sharp changes in the system activity, inducing rapid switches in the power consumption, the power estimated by the PMC-based power model has faster rising and

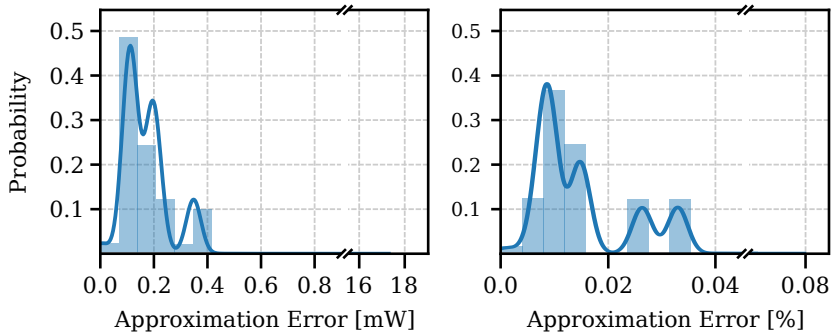


Figure 3: Distribution of approximation error between floating-point and fixed-point implementations of the CPU power model. The distribution is shown in terms of both absolute power approximation error and percentage error.

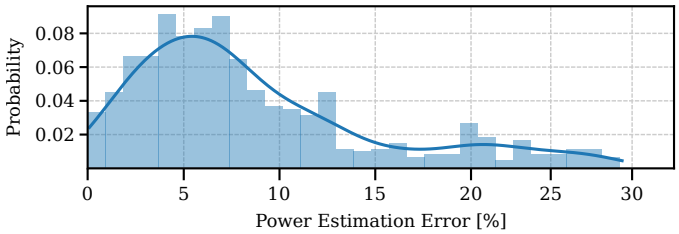
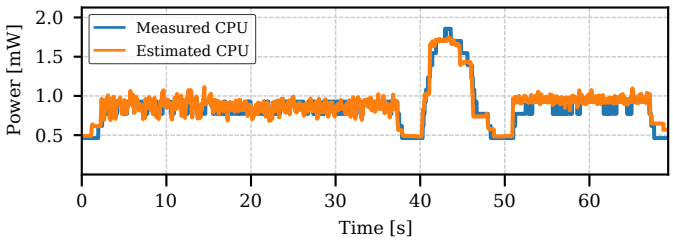


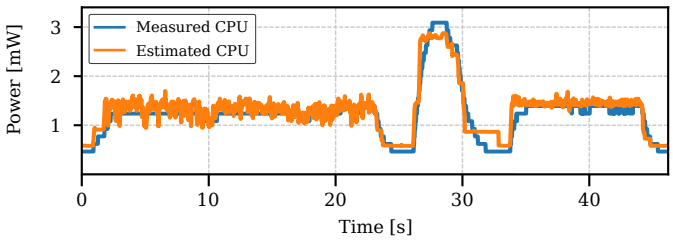
Figure 4: Distribution of the APE of the online energy estimates over the duration of each benchmark.

falling edges than the power measured by the analog sensor. This is especially visible at higher CPU frequencies, where the inertia of the analog current sensors has an increasingly worse impact on the latency of the measurements. On the other hand, PMCs are embedded in the digital domain, and their values instantly reflect the dynamic behavior of the monitored workloads. Nevertheless, our power modeling approach makes use of the onboard power sensor to build the input dataset for training and validation. While this makes the procedure automatic, easier, and less error-prone, it creates an unavoidable discrepancy between our estimates and the ground truth during transients, related to the high responsiveness of the PMC-based model.

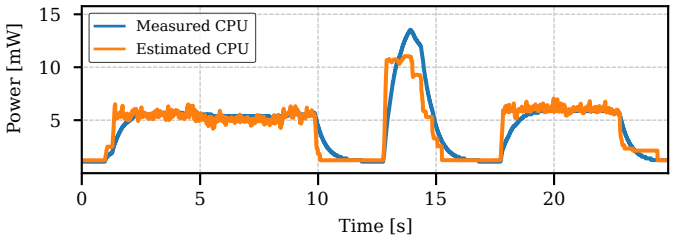
The problem emerges, then, of how to assess the reliability of our power estimates during transients if the model has been trained with a built-in analog sensor. As a solution, during the training phase, we deliberately bias the training set toward workloads with more stable activity: this means that the power model is trained, on average, with power values matching the actual



(a) 730 MHz



(b) 1.2 GHz



(c) 2.3 GHz

Figure 5: Comparison of the instantaneous CPU power consumption measurement provided by the onboard INA3221 sensor and the estimation computed at runtime by the in-kernel power model. Each plot represents the same sequential execution of several workloads over time at different frequencies.

consumption of the platform. Thanks to the linearity of the power models, such a solution decouples the trained weights from the low sensibility and time granularity of the input power data used for training. Once trained, the power model can scale and interpolate those values according to the PMC samples collected at runtime, providing faster power estimation and higher responsiveness.

5.4.3. Monitoring Overhead

Runmeter is integrated with callbacks triggered at specific times during the Linux kernel execution. This imposes a certain processing overhead mainly

due to PMC data collection and manipulation, including model estimation. To measure it, we profile the execution of the Runmeter Kernel Module callbacks. We perform these measurements in various working conditions, ranging from an "idle" state to the execution of multiple parallel applications from the set of benchmarks described in [12]. We use the same frequencies employed for the CPU model evaluation.

The maximum overhead is reported when many applications execute concurrently on the system, as the number of invocations of Runmeter's callbacks increases with the number of context switches performed by the system. In the worst-case condition of intense context switching, the time spent executing all of the framework's callbacks never exceeds 7 ms per second (i.e., 0.7 % overhead). Moreover, the execution of all framework's callbacks significantly speeds up when increasing the CPU frequency, reducing to less than 2 ms per second in the worst case (0.2 % overhead) when operating at 2.3 GHz. In idle conditions, the overhead of the framework at 2.3 GHz is always less than 0.4 ms per second (0.04 %).

6. Conclusions and Future Work

With this work, we propose a systematic, data-driven approach to DVFS-aware statistical power modeling of heterogeneous computing systems, whose implementation is decoupled from the target platform's microarchitectural details. We individually model each sub-system through its local PMCs, autonomously selecting the best ones to represent its power consumption. The sub-system models are later composed in a LUT-based system-level power model, able to grasp the complex behaviors of DVFS-enabled hardware using simple, linear expressions. This approach achieves a novel combination of automated model construction, low-overhead evaluation, high accuracy, responsiveness, and decomposability, proving itself suitable for real-time applications running on mobile and embedded systems.

To demonstrate the applicability of our power model, we propose Runmeter, a flexible framework for PMC monitoring and power model evaluation from within the Linux kernel. Runmeter is a substantial improvement over existing mechanisms based on PMC tracking, as it focuses on minimizing the response time between PMC observation and model evaluation, enhancing the responsiveness of power estimates with negligible overhead.

The validation of our power modeling approach on the state-of-the-art NVIDIA Jetson AGX Xavier embedded platform results in power and energy

estimation accuracies aligned with or higher than reported by previous state-of-the-art work. By integrating Runmeter in the Linux kernel of the same platform, we also prove the viability of our modeling and monitoring approach for online power tracking, a key prerequisite to implementing robust power-aware control loops in DPM and power-aware task scheduling.

While our methodology is designed to be independent of microarchitectural details, future work will explore its validation across multiple hardware platforms to further reinforce its general applicability. The automatically identified PMC set can also serve the development of an initial model, which can subsequently be refined with expert architectural knowledge when tighter accuracy constraints are required for a given application. Exploring different target platforms will also serve to demonstrate additional capabilities of our power models, such as leveraging *deep idle states*. These states are indeed unlikely to be reached in a platform with a single CPU frequency island, such as the AGX Xavier. Additionally, while our approach achieves excellent results without resorting to counter multiplexing, optimized event grouping techniques can enable more flexible parameter selection, potentially resulting in more accurate models [21]. However, time-sharing inherently introduces PMC tracking overhead and interpolation errors, requiring a careful trade-off assessment, particularly in real-time applications.

As far as our contribution to the Linux kernel is concerned, our results pave the way to bring the benefits of our modeling approach to the Linux real-time task scheduler `SCHED_DEADLINE` and the CPU frequency governor through the Runmeter framework. This aims to improve the effectiveness and correctness of energy-aware real-time task scheduling within Linux. Further directions of work also include going beyond the estimation of the current hardware status through predictive models. As of now, the Linux kernel contains very simple linear models for estimating the power consumed at each frequency, which are used to make decisions when selecting the appropriate operating frequency for the CPU. Models based on online PMC data, like that collected by Runmeter, may prove to be more effective from an energy-saving perspective while maintaining a very low overhead.

Acknowledgement

This work has received funding from the European Commission through the EU H2020 research project AMPERE (A Model-driven development

framework for highly Parallel and EneRgy-Efficient computation supporting multi-criteria optimization) under grant agreement no. 871669.

Sergio Mazzola and Gabriele Ara contributed equally to this work.

References

[1] T. Cucinotta, A. Amory, G. Ara, F. Paladino, M. D. Natale, Multi-criteria optimization of real-time DAGs on heterogeneous platforms under P-EDF, ACM Transactions on Embedded Computing Systems 23 (2024) 1–35.

[2] S. Alcaide, L. Kosmidis, C. Hernandez, J. Abella, Software-only based diverse redundancy for ASIL-D automotive applications on embedded HPC platforms, in: IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), 2020, pp. 1–4.

[3] A. Fuchs, D. Wentzlaff, The accelerator wall: Limits of chip specialization, in: 2019 IEEE International Symposium on High Performance Computer Architecture (HPCA), 2019, pp. 1–14.

[4] M. Duranton, K. De Bosschere, B. Coppens, C. Gamrat, T. Hoberg, H. Munk, C. Roderick, T. Vardanega, O. Zendra, HiPEAC vision 2021: high performance embedded architecture and compilation (2021).

[5] A. Mascitti, T. Cucinotta, M. Marinoni, L. Abeni, Dynamic partitioned scheduling of real-time tasks on ARM big.LITTLE architectures, Journal of Systems and Software 173 (2021) 110886.

[6] A. Balsini, L. Pannocchi, T. Cucinotta, Modeling and simulation of power consumption and execution times for real-time tasks on embedded heterogeneous architectures, ACM SIGBED Review 16 (2019) 51–56.

[7] G. Ara, T. Cucinotta, A. Mascitti, Simulating execution time and power consumption of real-time tasks on embedded platforms, in: Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing, ACM, 2022.

[8] R. Bertran, M. Gonzalez, X. Martorell, N. Navarro, E. Ayguade, A systematic methodology to generate decomposable and responsive power models for CMPs, IEEE Transactions on Computers 62 (2012) 1289–1302.

[9] V. Chau, X. Chu, H. Liu, Y.-W. Leung, Energy efficient job scheduling with DVFS for CPU-GPU heterogeneous systems, in: Proceedings of the Eighth International Conference on Future Energy Systems, 2017, pp. 1–11.

[10] W. Lin, et al., A taxonomy and survey of power models and power modeling for cloud servers, ACM Computing Surveys (CSUR) 53 (2020) 1–41.

[11] F. Bellosa, The benefits of event: driven energy accounting in power-sensitive systems, in: Proceedings of the 9th workshop on ACM SIGOPS European workshop: beyond the PC: new challenges for the operating system, 2000, pp. 37–42.

[12] S. Mazzola, T. Benz, B. Forsberg, L. Benini, A data-driven approach to lightweight DVFS-aware counter-based power modeling for heterogeneous platforms, in: International Conference on Embedded Computer Systems, Springer, 2022, pp. 346–361.

[13] C. Yoon, S. Lee, Y. Choi, R. Ha, H. Cha, Accurate power modeling of modern mobile application processors, Journal of Systems Architecture 81 (2017) 17–31.

[14] R. W. Ahmad, et al., A survey on energy estimation and power modeling schemes for smartphone applications, International Journal of Communication Systems 30 (2017) e3234.

[15] O. Djedidi, M. A. Djeziri, Power profiling and monitoring in embedded systems: A comparative study and a novel methodology based on NARX neural networks, Journal of Systems Architecture 111 (2020) 101805.

[16] D. Zoni, A. Galimberti, W. Fornaciari, A survey on run-time power monitors at the edge, ACM Comput. Surv. 55 (2023). URL: `https://doi.org/10.1145/3593044`. doi:doi: 10.1145/3593044.

[17] L. Castro, An Engineer's Guide to Current Sensing (Rev. B), Texas Instruments, 2023, pp. 44–45.

[18] C. Isci, M. Martonosi, Runtime power monitoring in high-end processors: Methodology and empirical data, in: Proceedings. 36th Annual IEEE/ACM International Symposium on Microarchitecture, 2003. MICRO-36., IEEE, 2003, pp. 93–104.

[19] NVIDIA Corporation, Jetson AGX Xavier developer kit, 2018. URL: https://developer.nvidia.com/embedded/jetson-agx-xavier-developer-kit.

[20] ARM Holdings, ARM Cortex-A57 MPCore processor technical reference manual, 2016.

[21] T.-Y. Liu, J. Guo, B. Huang, Efficient cross-platform multiplexing of hardware performance counters via adaptive grouping, ACM Trans. Archit. Code Optim. 21 (2024). URL: https://doi.org/10.1145/3629525. doi:doi: 10.1145/3629525.

[22] M. Pi Puig, L. C. De Giusti, M. Naiouf, A. E. De Giusti, A study of hardware performance counters selection for cross architectural GPU power modeling, in: XXV Congreso Argentino de Ciencias de la Computación (CACIC)(Universidad Nacional de Río Cuarto, Córdoba, 14 al 18 de octubre de 2019), 2019.

[23] M. J. Walker, et al., Accurate and stable run-time power modeling for mobile and embedded CPUs, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 36 (2016) 106–119.

[24] Q. Wang, N. Li, L. Shen, Z. Wang, A statistic approach for power analysis of integrated GPU, Soft Computing 23 (2019) 827–836.

[25] N. Mammeri, M. Neu, S. Lal, B. Juurlink, Performance counters based power modeling of mobile GPUs using deep learning, in: 2019 International Conference on High Performance Computing & Simulation (HPCS), IEEE, 2019, pp. 193–200.

[26] A. Tarafdar, S. Sarkar, R. K. Das, S. Khatua, Power modeling for energy-efficient resource management in a cloud data center, Journal of Grid Computing 21 (2023) 10.

[27] R. Bertran, M. Gonzalez, X. Martorell, N. Navarro, E. Ayguadé, Counter-based power modeling methods: Top-down vs. bottom-up, The Computer Journal 56 (2013) 198–213.

[28] J. Phung, Y. C. Lee, A. Y. Zomaya, Lightweight power monitoring framework for virtualized computing environments, IEEE Transactions on Computers 69 (2020) 14–25.

[29] K. K. Pusukuri, D. Vengerov, A. Fedorova, A methodology for developing simple and robust power models using performance monitoring events, Proceedings of WIOSCA 9 (2009).

[30] K. Singh, M. Bhadauria, S. A. McKee, Real time power estimation and thread scheduling via performance counters, ACM SIGARCH Computer Architecture News 37 (2009) 46–55.

[31] W. L. Bircher, L. K. John, Complete system power estimation using processor performance events, IEEE Transactions on Computers 61 (2011) 563–577.

[32] M. Pricopi, T. S. Muthukaruppan, V. Venkataramani, T. Mitra, S. Vishin, Power-performance modeling on asymmetric multi-cores, in: 2013 International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES), 2013, pp. 1–10. doi:doi: 10.1109/CASES.2013.6662519.

[33] R. Rodrigues, A. Annamalai, I. Koren, S. Kundu, A study on the use of performance counters to estimate power in microprocessors, IEEE Transactions on Circuits and Systems II: Express Briefs 60 (2013) 882–886. doi:doi: 10.1109/TCSII.2013.2285966.

[34] J. C. Saez, A. Pousa, R. Rodriíguez-Rodriíguez, F. Castro, M. Prieto-Matias, PMCTrack: Delivering performance monitoring counter support to the OS scheduler, The Computer Journal 60 (2017) 60–85.

[35] V. M. L. Xu, L. W. McShane, D. Mossé, Lush: Lightweight framework for user-level scheduling in heterogeneous multicores, in: 2021 IEEE 14th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC), IEEE, 2021, pp. 396–404.

[36] J. C. McCullough, Y. Agarwal, J. Chandrashekar, S. Kuppuswamy, A. C. Snoeren, R. K. Gupta, et al., Evaluating the effectiveness of model-based power characterization, in: USENIX Annual Technical Conf, volume 20, 2011, pp. 19–20.

[37] J. Leng, T. Hetherington, A. ElTantawy, S. Gilani, N. S. Kim, T. M. Aamodt, V. J. Reddi, Gpuwattch: Enabling energy optimizations in gpgpus, ACM SIGARCH computer architecture news 41 (2013) 487–498.

[38] C. Bilbao, J. C. Saez, M. Prieto-Matias, Flexible system software scheduling for asymmetric multicore systems with pmcsched: A case for intel alder lake, Concurrency and Computation: Practice and Experience 35 (2023) e7814.

[39] S. Che, et al., Rodinia: A benchmark suite for heterogeneous computing, in: 2009 IEEE international symposium on workload characterization (IISWC), IEEE, 2009, pp. 44–54.

[40] INA3221 triple-channel, high-side measurement, shunt and bus voltage monitor with I^2C- and SMBUS-compatible interface, Texas Instruments, 2012. Rev. B.

[41] A. D. Malony, et al., Parallel performance measurement of heterogeneous parallel systems with GPUs, in: 2011 international conference on parallel processing, IEEE, 2011, pp. 176–185.