High-Precision Power Modelling of the Tegra K1 Variable SMP Processor Architecture

Kristoffer Robin Stokke, Håkon Kvale Stensland, Pål Halvorsen, Carsten Griwodz
Simula Research Laboratory & University of Oslo
{krisrst, haakonks, paalh, griff}@ifi.uio.no

Abstract—**Energy efficiency is an important issue for many embedded systems, where limited battery lifetime and power-hungry hardware constrain the usefulness of such devices. Modern Systems-on-Chip (SoCs) such as the Tegra K1 employ advanced power management capabilities such as two CPU clusters, clock-gating, power-gating and dynamic frequency tuning to meet application demands. At design or runtime phases, it is challenging for system architects and software developers to understand the effects that these mechanisms have in terms of power and performance in all parts of the system. This is because it is impossible to measure directly the power usage of cores, caches, memory and other hardware components. Rate-based power models are often proposed as a solution for this, unfortunately these can mispredict substantially on the Tegra K1 up to 30 %. In this paper, we propose a power modelling method for the Tegra K1 CPU which overcomes the limitations of the most common types of models found in literature, but still only requires power measurement of the board. Through extensive empirical validation, we demonstrate an accuracy which is close to 100 %. Preliminary experiments show that our methodology is able to capture instruction power of individual system processes and applications and produce detailed power breakdowns of all components in the system.**

I. Introduction

Power usage is a timely topic in multicore embedded systems. Devices such as smart phones, drones and laptops are limited by a combination of power-intensive hardware and limited battery capacity. For such devices, it is important that they stay alive for as long as possible while at the same time providing acceptable quality of service. One example is video processing. According to Cisco, by 2019, the sum of all forms of video will be in the range of 80 to 90 percent of global consumer traffic, and of this, 14 percent will be mobile data traffic [4]. When recording live events such as sports using your smart phone or even a drone, it is important that the device can manage the limited energy resource.

It is well known that state-of-the-art power management generally reacts too quickly to changes in hardware utilisation, and consequently end up staying at unnecessarily high CPU and RAM operating frequencies [13], [15], [17], [22]. For system architects it is therefore important to fine-tune software algorithms, gating techniques, platform frequencies and workloads for the different parts of the system at early design stages, or even design intelligent power management schemes at runtime. However, this facilitates the need to understand power usage of individual units in the Tegra K1 Variable SMP (vSMP) [12] processor architecture, such as the Tegra K1's Low Power (LP) core, High Performance (HP) cluster, caches, RAM banks and clocks under the influence of software, which is often unfeasible due to circuit-layout limitations.

It is not trivial to understand power usage of embedded SoCs. Simply measuring power is a serious challenge due to a lack of power measurement sensors. Research has also shown that such sensors are inaccurate [5] and often only capable of measuring total power usage. This includes individual power draw from processor elements, buses, caches, graphics processing units, hardware accelerators, regulators [3] and other components [2]. Researchers often attempt to *model* power usage of individual components by correlating total power usage with hardware activity and/or hardware states, and three model types are dominant in literature: *state-*, *rate-* and *cmos-based* power models.

Aside from their individual strengths and limitations, the three different modelling methodologies share a common deficiency: they have the potential for serious misprediction of power on the Tegra K1. This flaw has not been apparent before because the model implementations are not tested extensively enough over *all CPU and memory frequency combinations*. Rate-based power models, which are by far the most commonly found in literature [5], [8], [21], [23], can for example mispredict up to 30 % on the Tegra K1 when the CPU frequency is above 1 GHz. This occurs for several reasons. Rate-based models correlate total power usage with hardware activity, such as the number of instructions, cycles or cache misses per second. However, many mechanisms that have non-negligible impact on power usage, such as frequency scaling, variable cost of instructions, resource contention [1] and clock, core and rail gating are usually ignored. Additionally, rate-based models do not consider that rail voltages vary with operating frequencies, having an adverse affect on power usage [3], [9], [15]. For this reason, the accuracy of rate-based models is entirely dependent of the frequency levels selected by Dynamic Voltage and Frequency Scaling (DVFS) algorithms at the time of the verification.

Models are necessary to understand the power usage of small hardware components where direct measurement is otherwise physically impossible or unsupported. They are also necessary to attribute energy consumption to individual software applications, determine and reduce the power usage of components and to evaluate how systems consume power. However, existing modeling methods fail to predict power accurately on the Tegra K1.

In our previous work [18], we developed a high precision power model for the GPU on the Tegra K1. This model included a very simplified model for the Tegra K1's LP core. In this paper, we extend this model to predict power usage of the Tegra K1's quad-core CPU, different CPU caches and memory. The accuracy is close to 100 % for multimedia workloads such as the Discrete Cosine Transform (DCT), huffman encoding, motion vector search and rotation. The main contribution compared to the GPU model is that the CPU cannot be modelled *generally*. This is because there is a lack of Hardware Performance Counters (HPCs) that can measure individual integer, floating point, data movement and other types of instructions. Instead, our method for the CPU estimates the average capacitive load per instruction on a per-process basis. By taking into account measured rail voltages and fine-grained hardware activity predictors, our model exposes detailed insight into the power usage of individual components, such as rail and core leakage currents, RAM and CPU clock, RAM reads and writes, application-specific workload power and cache hierarchies. Using the model, it is possible to identify power-intensive hardware components and software workloads. This can for example be used to optimise the power usage of individual system services, reducing idle-system workload power by 21 %. Combined, our GPU and CPU models comprise a *complete* heterogeneous power model for the Tegra K1, covering all general purpose compute aspects of the SoC from its dual-cluster CPU, RAM and GPU.

The rest of this paper is organised as follows. We survey related work and background on power modelling in Section II. Section III describes the Tegra K1 platform and the details that are important to model the platform accurately, in particular, why it is impossible to build an entirely generic power model for the Tegra K1's CPU. The modeling methodology, model training benchmarks as well as the coefficients are introduced in Section IV. Furthermore, we extensively verify our model over all platform frequencies using several intense compute workloads in Section V, where we also investigate workload-dependent power components and some preliminary use cases of our model. We conclude our work in Section VI.

II. Background and Motivation

There is an abundance of work which attempts to model power usage of embedded systems. These generally fall into three categories of models: state-, rate- and cmos-based models. Each type has its own advantages and disadvantages; but as mentioned in the previous section, they can mispredict badly depending on operating frequencies. Surprisingly, all of them contribute with important insight into power, and they all hold some merit which is key to building accurate models. In this section, we take a brief look at these model types and study their accuracy on the Tegra K1. More extensive details about model predictors and estimated coefficients for these are available for download here[1].

Modern SoCs are based on CMOS technology and are built with *voltage-islands*. This means that the components within the SoC (clock generators, caches, buses, compute cores etc) are divided into regions supplied by individual *power rails* (see Figure 1). The power on a rail P_R is the sum of static $P_{R,stat}$ and dynamic $P_{R,dyn}$ power [9], [3] and make up the foundation for cmos-based models:

$$P_R = I_{R,leak} V_R + \alpha_R C_R V_R^2 f_R \tag{1}$$

In this equation, rail voltage V_R and leakage current $I_{R,leak}$ defines the static power component. Dynamic power is caused by transistor switching activity, where f_R is the operating frequency in cycles per second on that rail, C_R is the potential maximum switching capacitance per cycle (in coloumbs per volt), and $\alpha_R \in [0, 1]$ is best viewed as a workload-specific factor which decides how much of C_R is being switched through the circuitry on that rail, per cycle. A key point from Equation 1 is that rail power is strongly dependent on rail voltage, which again depends on operating frequencies in that domain (see Figure 2).

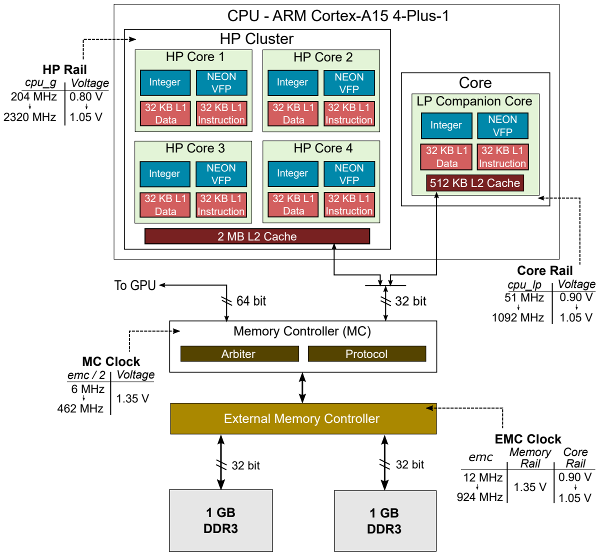
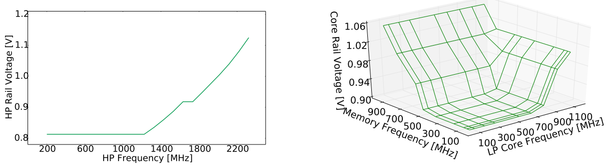


Fig. 1: Overview of the Tegra K1 SoC architecture [11].



(a) The HP rail.

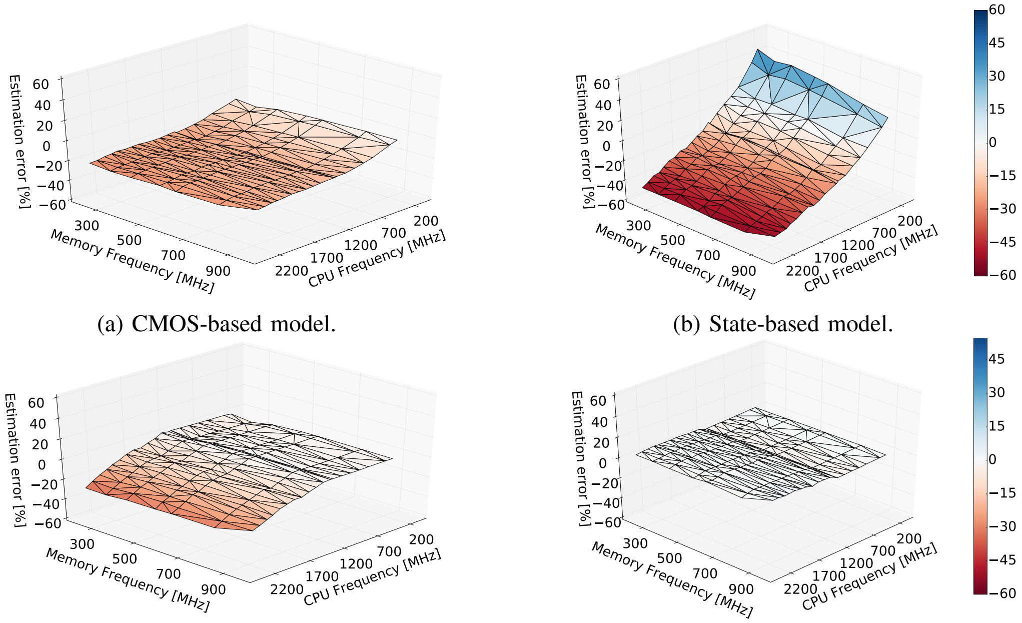
(b) The core rail.

Fig. 2: Measured rail voltages.

Several authors [3], [15] have attempted to estimate leakage currents $I_{R,leak}$ and switching capacitance $\alpha_R C_R$ for rails directly using regression. However, despite being theoretically well-founded, cmos-based models mostly mispredicts between 20 to 25 % (see Figure 3a). This is a trend we also discovered in our cmos-model [15] which occurs because the model assumes an independent relationship between switching activity and frequency levels in different domains. It is implicitly assumed that the increase in power as for example CPU frequency increases, is caused only by increased switching activity on that rail. However, it is easy to imagine cases where this is not true, for example when executing a memory intensive program.

State-based models are perhaps the simplest model type found in literature [16]. Models of this type abstract hardware components into states and associate each with a constant power draw and transition cost between these [19]. For example, the Tegra K1's CPU can be abstracted into two states that reflect the currently active cluster, where either the Low Power (LP) core or High Performance (HP) cluster is active. It is further possible to add the state cost of having individual cores active. State-based models are relatable to the change in

[1] folk.uio.no/krisrst/papers/mcsoc/2016/model.ods



(a) CMOS-based model.

(b) State-based model.

(c) Rate-based model.

(d) Our hybrid model.

Fig. 3: Power estimation error for common model types running a DCT filter (four cores).

leakage current $I_{R,leak}$ on power rails which is directly tied to the activation and deactivation of hardware components such as cores and rails. However, they have bad accuracy on the Tegra K1 because they ignore changes in rail voltages and dynamic power completely (see Figure 3b).

Rate-based models are the most intensively used models in the literature [5], [8], [14], [20], [21], [23] and has seen widespread adoption since 1999 [6] (which is the first rate-based model we have found). Rate-based models attempt to correlate power usage with the rate at which hardware events occur according to the following formula:

$$P_{tot} = \beta_0 + \sum_{i=1}^{N_\rho} \rho_i \beta_i \tag{2}$$

In Equation 2, β_0 is the power of idle components with constant power draw, ρ_i is a predictor (hardware event) with units of accesses per second, β_i is the cost in Watt-seconds per access and N_ρ is the number of predictors. Example of hardware events can be instruction execution, elapsed cycles, cache hits and misses and branching. These events naturally reflect transistor switching activity, tying it to the *dynamic power* component of Equation 1. With a prediction error between 0 to 30 %, this model predicts better than the state-based one (see Figure 3c). However, rate-based models ignore rail voltages and static power. The only known exception is Hong, S. and Kim, H. [7] who presented a standard rate-based model, but it considers voltage as a part of the leakage current. In defence of rate-based models, ignoring voltage levels might not have a negative effect if power management is poorly designed. For example, it is much easier to stick the Tegra K1 HP frequency at 1.15 V (see Figure 2a) independently of frequency. It does however have an adverse effect on systems where this is not the case, such as the Tegra K1.

In summary, state-based models reflect static power, but do not capture dynamic power. Rate-based models are the opposite. These reflect hardware utilisation and dynamic power but not static power. Both ignore changes in voltage as a result of changes in frequencies, and is a key source of inaccuracy in these model types. Cmos-based models incorporate both static and dynamic power as well as rail voltages. In terms of accuracy, it shows less error variation, but has an inherent weakness in that they assume independency in switching activity and operating frequencies between rails. Our hypothesis is therefore that we can achieve better accuracy if hardware utilisation and switching capacitance can be captured more accurately and independently on each rail. In the following sections, we will outline important hardware activity predictors on the Tegra K1's CPU and how these can be combined with voltage measurements to achieve close to 100 % accuracy (see Figure 3d).

III. Tegra K1 Mobile SoC

In order to successfully build a high-precision power model, it is important to have a solid understanding of the platform (see Figure 1). We have chosen the Tegra K1 as a case study due to its similarity with modern island-style SoCs. In this section, we introduce the most important power rails and clocks, and we describe how it is possible to monitor hardware activity in the various units. Power measurement and synchronisation is done as in our previous work [15], where the Jetson-TK1 retrieves power measurement readings from a Keithley 2280S power source via a dedicated measurement machine. We measure individual rail voltages using the Keithley 2110 high-precision voltage measurement unit.

A. Tegra K1 Architecture: Rails and Clocks

The Tegra K1 is a complete SoC featuring several *rails* powering various functional units on the processor [10]. The rails which are relevant for our investigation (see Figure 1) are the *core*, *HP* and *memory* rails. These rails power the LP (core) and HP clusters, as well as the External Memory Controller (EMC) and memory banks. The power usage on all other rails are assumed to be constant with no hardware activity.

Clocks drive switching activity inside the Tegra K1's hardware components. Every clock cycle consumes energy, and CPU clock cycles normally correlate very strongly with

Clock	Affects	Description	Frequency		Voltage Range
			Steps	Range $[MHz]$	
cpu_g	HP Rail	HP cluster	20	$[204, 2320]$	$[0.80, 1.20]$
cpu_lp	Core Rail	LP core	9	$[51, 1092]$	$[0.90, 1.05]$
EMC	Core Rail	Memory	6	$[204, 924]$	$[0.90, 1.01]$

TABLE I: Tegra K1 cores, clock and voltage ranges.

power in rate-based models [20]. Clocks are also provided as a power management mechanism where the operating frequency of a component can be reduced to mitigate power dissipation (while reducing performance). The Tegra K1 features a range of different clocks driving different functional blocks (see Table I). When building power models, it is important to be aware that the voltages on different rails change, and that they change with clock frequency [3], [9]. For example, depending on the HP cluster clock (cpu_g), HP rail voltage V_{hp} varies between 0.81 and 1.13 V (see Figure 2a). The core rail voltage is more complex. Its voltage V_{core} depends on the LP core frequency (cpu_lp) as well as the EMC bus frequency. Unless the HP cluster is active, the voltage set on this rail is always the maximum required by any of these two clocks at any point in time (see Figure 2b). Finally, memory rail voltage is statically set to 1.35 V and does not change.

B. Power Saving Mechanisms

The Tegra K1 features several mechanisms to limit power usage depending on the current demand for resources and hardware utilisation. Generally, there are two types of optimisation approaches to achieve this goal, which have never been taken directly into account in previous models:

- *Clock gating* disables clock distribution to various entities of the SoC, which disables their function and reduces dynamic power. State is retained.
- *Power gating* disables supply voltage to various parts of the circuity. This completely removes static power draw and normally implies clock gating. Power gating causes loss of state (for example in caches).

Exactly when and for how long cores are gated is decided by both software and hardware. On the Jetson-TK1, the CPU-idle kernel drivers monitor idle CPU cores and make decisions to clock- or power-gate individual cores based on estimated idle intervals and state transition overheads. Additonally, the CPU cores perform clock-gating in hardware without the intervention of software.

Application demand for processing time is satisfied by automatic cluster and core selection as well as CPU and memory frequency tuning. All of these mechanisms impact power usage. Typically, when the Tegra K1 is idle, processing is restricted to the LP core, in which case the entire HP rail is power gated by disabling the HP rail voltage regulator. Kernel drivers monitor resource usage. These may decide to switch between the application clusters, activate additional CPU cores, clock- and power-gate individual cores in short idle periods and adjust EMC and CPU frequency. The challenge with regards to power modelling is to track when and for how long processors or other parts of the circuitry remains gated. To solve this, we modified the Tegra K1 kernel sources to count the time spent in the gated states and expose this information to running applications.

C. Instruction Cost (Workload-Dependent Power)

One of the main challenges of power modelling the Tegra K1 is that it does not have fine-grained accounting for the types and number of instructions executed. Pricopi et. al. [14] built a rate-based power model for a big.Little CPU, which is the same cluster technology the Tegra K1 uses. They show that it is possible to attribute an instruction cost to different instructions such as branching, integer and floating point operations, as well as RAM loads and stores. However, the CPU implementation they used had dedicated HPCs to count these instructions. This is also in accordance with our experiences on the Tegra K1's GPU [18], where we build an fully generic power model using fine-grained instruction accounting. Unfortunately, the Tegra K1 does not implement these HPCs on its CPU.

When there is no possibility to trace what types of instructions have been executed, loss of generality in terms of power modelling is unavoidable. Every application and system service has its own way of exersising the processing pipeline using different instructions. The only instruction counter we can use is the `instruction executed` HPC, which counts the number of instructions executed. Since the number of CPU instructions can be counted on a per-process basis, we choose to track these and estimate an average capacitance load per instruction for each workload. The assumption behind this is that each application will have roughly the same average capacitive load over time. ARM CPUs are not required to implement HPCs for fine-grained instruction accoutning, which means that for CPUs it will be unavoidable to model workload power in this way.

D. Cache Maintenance and Off-Chip Memory Access

The Tegra K1 implements a two-level on-chip cache hierarchy with last-recently used eviction policy. Cache costs usually surface as a part of rate-based power models, where authors attempt to attribute power costs to cache accesses and misses directly [20]. In our model training, we found that these events generally do not correlate well with power, often breaking the estimation such that coefficients are negative. This is not easy to understand, but we believe that the cost of *accessing* caches is not constant. A cache access may end up in other data being evicted elsewhere or trigger refills from other cache levels or memory: the cost is not *generic* in itself and depends on many other factors.

Because of the problems of modelling cache accesses, we decided to attempt to model instead the power which is spent *maintaining* the cache. The intuition behind this is that, if we can model the way the cache maintains its own consistency, cache accesses will instead just be an integral part of the average instruction cost (see Section III-C). The ARM HPC implementation has a range of cache performance counters. We use the following HPCs:

- **L1D_CACHE_REFILL.** Counts data cache refill events from external, off-chip memory.
- **L1D_CACHE_WB.** Counts data cache writebacks to external, off-chip memory.
- **L2D_CACHE_REFILL.** Counts data cache refill events from L1 cache or external, off-chip memory.

- **L2D_CACHE_WB.** Counts data cache writebacks to L1 cache or external, off-chip memory.

We then defined two new counters that trace cache maintenance operations between the cache hierarchies:

$$\rho_{l1l2} = (N_{l1,refill} - N_{l2,refill}) + (N_{l1,wb} - N_{l2,wb}) \quad (3)$$

$$\rho_{l2ram} = N_{l2,refill} + N_{l2,wb} \quad (4)$$

where ρ_{l1l2} is an indication of on-chip cache maintenance between L1 and L2 data caches, and ρ_{l2ram} is an indication of cache traffic between L2 and off-chip memory. Note that the Tegra K1 also implements other types of caches, for example L1 instruction cache. Since only seven HPCs can be occupied concurrently, and one of these always is the `active cycle` HPC, there is not enough HPC space to trace all the hierarchies.

Unique memory accesses ultimately end up in off-chip memory accesses at least once. Exactly how often this happens is very hard to trace. CPU HPCs exist, but there may be many other components in the system which also access RAM, such as PCI devices and peripherals. Fortunately, the Tegra K1 implements an activity monitor which can provide the number of active memory cycles spent serving the CPU or other peripherals. We modifiy the kernel driver for the activity monitor to provide us with these statistics, which represent our measure of memory utilisation.

IV. High-Precision Power Modelling

We outline our methodology to model the Tegra K1 power usage with high precision. The method is an extension to our previous work on the Tegra K1's GPU [18]. The main challenge with modelling power on the Tegra K1's CPU compared to the GPU is that it is impossible to build an entirely generic model. This is because the CPU does not have dedicated HPCs for the type and number of instructions executed (see Section III-C). Therefore, the resulting model is a combination of *generic* power (most of the predictors seen in Table II) and *workload-specifc* power, which varies depending on the various types of instructions executed on each core.

A. Derivation

Following our discussion in Section II, we now describe the dynamic power component of Equation 1 in terms of measurable hardware activity predictors as outlined in Section III. The dynamic part of Equation 1 can be re-interpreted as follows:

$$P_{R,dyn} = \sum_{i=1}^{N_R} C_{R,i}\rho_{R,i}V_R^2 \quad (5)$$

In Equation 5, N_R is the number of hardware predictors, $\rho_{R,i}$ is a hardware activity predictor for rail R in occurrences per second (for example instructions per second), and $C_{R,i}$ is the unknown switching capacitance per occurrence of event $\rho_{R,i}$. Static power on a rail R is the product of that rail's voltage V_R and total leakage current $I_{R,ltot}$ [9]:

$$P_{R,stat} = V_R I_{R,ltot} \quad (6)$$

The Tegra K1 continuously performs power gating of processors on the HP and core rails. When power gating a core, the leakage current $I_{cpu,leak}$ from that core is also removed. The total leakage current on the HP and core rail, for N_c active cores, is:

$$I_{R,ltot} = I_{R,leak} + \sum_{i=1}^{N_c} I_{cpu,leak} \quad (7)$$

The total power usage of the Jetson-TK1 becomes:

$$P_{jetson} = \sum^{R\in\mathbb{R}} (P_{R,dyn} + P_{R,stat}) + P_{base} \quad (8)$$

Noting that the static power on the memory rail cannot be modelled, because the rail voltage on this rail does not change. It is instead a part of base power P_{base}, which also includes the constant power draw of all other rails and idle components.

B. Methodology

The total power usage of the Jetson-TK1 is shown in Equation 8, where the unknown variables are the capacitive loads, rail and core leakage coefficients ($C_{R,i}$, $I_{R,leak}$ and $I_{cpu,leak}$). Base power P_{base} is also unknown. Our methodology to find these terms is based on multi-variable, linear regression. We create seven benchmarks spesialised to stress different architectural units of the Tegra K1's processor (see Table III):

- We have several versions of a simple matrix-multiply program, where the element type, for example integer, floating point (VFP) or vectorised floating point (NEON), is being varied.
- An idle benchmark, where only the Linux kernel and system services are running, is useful to trigger power- and clock-gating mechanisms.
- We found it necessary to also implement specialised benchmarks to stress L1 cache refills and write-backs more than the other benchmarks did.

Each benchmark is run over all possible memory and processor frequency combination (see Table I), and over the five possible core combinations (LP core or any number of the four HP cores active). During the benchmarks, power is being logged while the HPCs are being collected at regular, short intervals of 100 ms. This is to avoid counter overflow, which occurs relatively easily for some of the HPCs (under loads, the active cycle counter overflows within two seconds at the maximum operating frequency). The final dataset size is of 30912 entries containing the necessary predictors and power measurement samples. The coefficient estimates can be seen in Table II.

As argued in Section III-C, it is not possible to generalise the cost of instruction execution because different instruction types (integer, floating point, data movement) have different costs. The Tegra K1 only has one instruction counter, and the estimated cost per instruction will vary depending on workload. We make a single instruction cost estimate for each of the benchmarks in Table III. The estimates are shown in Table III. Not considering the IDLE test, which has a significantly larger instruction cost, the cost is similar across the rest of the benchmarks. Higher instruction cost does not automatically imply a higher rate of energy consumption. For example, the IDLE test consumes less power on average than

Rail	Number	Predictor	Description	Coefficient	Value
HP	0	V_{hp}	HP rail voltage (when powered)	$I_{hp,leak}$	$59.8mA$
	1	$\rho_{hp,clk1}$	Active clock cycles per second (first core)	$C_{hp,clk1}$	$395.65\frac{pC}{V}$
	2	$\rho_{hp,clk2}$	Active clock cycles per second (second core)	$C_{hp,clk2}$	$270.40\frac{pC}{V}$
	3	$\rho_{hp,clk3}$	Active clock cycles per second (third core)	$C_{hp,clk3}$	$261.89\frac{pC}{V}$
	4	$\rho_{hp,clk4}$	Active clock cycles per second (fourth core)	$C_{hp,clk4}$	$213.95\frac{pC}{V}$
Core	0	V_{core}	Core rail voltage (always powered)	$I_{core,leak}$	$633.7mA$
	1	$\rho_{core,clk}$	Active clock cycles per second (LP core)	$C_{core,clk}$	$301.49\frac{pC}{V}$
Common	0	$V_{com,online}$	Rail voltage when any core is online (not gated)	$I_{cpu,leak}$	$24.00mA$
	1	$\rho_{com,l1l2}$	Cache maintenance, L1 and L2	$C_{com,l1l2}$	$2.35\frac{nC}{V}$
	2	$\rho_{com,l2ram}$	Cache maintenance, L2 and RAM	$C_{com,l2ram}$	$2.29\frac{nC}{V}$
	3	$\rho_{com,ips}$	Instructions per second (workload-specific)	$C_{com,ips}$	See Table III
Memory	0	$\rho_{mem,clk}$	Total clock cycles per second	$C_{mem,clk}$	$238.21\frac{pC}{V}$
	1	$\beta_{mem,204}$	Power offset at 204 MHz	$P_{mem,204}$	$11.80mW$
	2	$\beta_{mem,300}$	Power offset at 300 MHz	$P_{mem,300}$	$69.00mW$
	3	$\rho_{mem,CPU}$	CPU busy memory (EMC) cycles per second	$C_{mem,cpu}$	$2.15\frac{nC}{V}$
	4	$\rho_{mem,OTH}$	Other busy memory (EMC) cycles per second	$C_{mem,oth}$	$2.76\frac{nC}{V}$
Other	P_{base}		Base power	-	$0.87W$

TABLE II: Overview of generic energy model predictors and coefficients.

Benchmark	Description	Components under explicit stress						Instruction Cost
		RAM (CPU)	L1 - L2	L2 - RAM	INT	FPU	NEON	
Idle CPU	CPU idle.		✓					$2.50\frac{nC}{V}$
L1-WB	L1 writeback stress.		✓					$0.15\frac{nC}{V}$
L1-RF	L1 refill stress.		✓					$0.18\frac{nC}{V}$
MMUL-INT	Matrix multiply (integer operations).	✓	✓	✓	✓			$0.45\frac{nC}{V}$
MMUL-INT-VOL	Same as above, volatile memory.	✓	✓	✓	✓			$0.27\frac{nC}{V}$
MMUL-F32	Matrix multiply (floating point operations).	✓	✓	✓		✓		$0.36\frac{nC}{V}$
MMUL-NEON	Matrix multiply (NEON floating point operations).	✓	✓	✓			✓	$0.44\frac{nC}{V}$

TABLE III: Benchmarks and components under stress.

the other benchmarks, despite the high estimated instruction cost. This is because so few instructions are executed compared to the other benchmarks. Similar conclusions can be made to the other benchmarks. Comparing for example floating point matrix multiply with VFP or NEON instructions, the NEON variant has a higher estimated cost per instruction. However, NEON instructions process four floating point values at a time whereas the VFP variant only process one at a time. Our training data therefore shows that the NEON variant draws less average power than the VFP variant.

V. Experiments

In this section, we verify the accuracy of our model by extensive validation over all operating frequencies. The workloads we use are common video processing operations for the DCT, Huffman coding, motion vector search and rotation. Due to space limitations, we are unable to describe these here, but interested readers can refer to our previous work for details [18]. We also argue that our method to model workload-specific instruction power is correct, and finally present some preliminary case studies of potential use cases for the model.

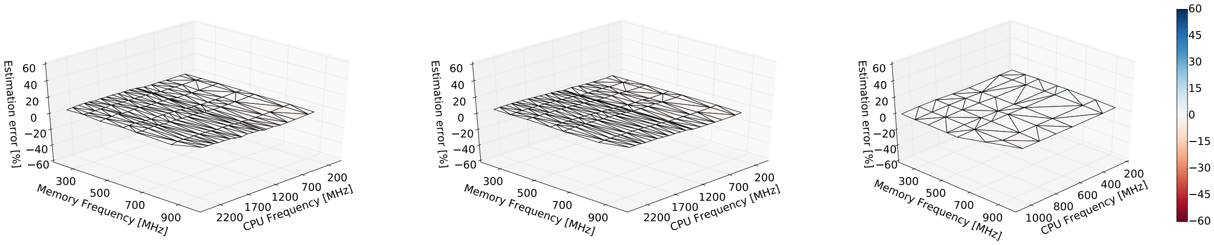
A. *Verification*

To verify our model, we let our video processing filters process an HD stream at 25 FPS over all possible operating frequencies. This process is done once for each core configuration (five possible combinations where either the LP cluster or any number of the four HP cores are active). While running, our implementation estimates generic platform power every 100 ms (see Section III-C) using the predictors in Table II and additionally logs measured total power usage. The workload-dependent power usage is now missing, and must be estimated per workload. When each filter is done processing the HD stream at all frequency combinations, per-instruction switching capacitance is estimated by subtracting predicted generic platform power from total measured power. The residual power is always positive and is used in a linear regression solver to estimate the switching capacitance per instruction, which is in turn used to estimate the power component directly related to instruction execution.

The resulting prediction error can be seen in Figures 3d and 4 for all of our filters. Due to space restrictions, we can not show all the error plots for each core configuration, but we include plots across all of them for at least one filter. The plots show that our model performs well, with a high accuracy across all frequency domains which is very close to 100 %. In general, the average accuracy over all frequencies is never below 98 % for any filter, with worst-case prediction errors that are never worse than ± 4 %. This is much better than the related modelling methods in Figure 3, where our model not only successfully captures hardware utilisation (dynamic power), but also takes into account increased power from higher rail voltages and leakage currents (static power).

B. *Residual (Workload/Instruction-Dependent) Power*

Our verification methodology has an unavoidable weak point in the way workload-dependent power is estimated. Per-instruction capacitance loss is estimated based on *residual* power *after* generic power components are removed from real measurements. The estimated workload power is subsequently *added* to the power prediction. It is possible to claim that this estimated workload-dependent power component ”simply happens to nicely cover” most of the residual power which would otherwise be the root cause of a large prediction error. It is difficult to conclusively disprove this claim. Per-instruction capacitance loss is tied to software implementation; it can not be re-used for other pieces of code and it cannot be guaranteed to be the same if the code is changed by recompilation or by changing input data (for example, reducing video resolution or changing algorithmic parameters). It is, however, possible to show in some more detail how residual workload power behaves across frequencies, how diverse per-instruction capacitance loss estimates are and how they can be used in practice.



(a) Rotation filter (three cores).
(b) Huffman filter (one HP core).
(c) Motion vector search (LP core).

Fig. 4: Power estimation error for idle and video processing scenarios.

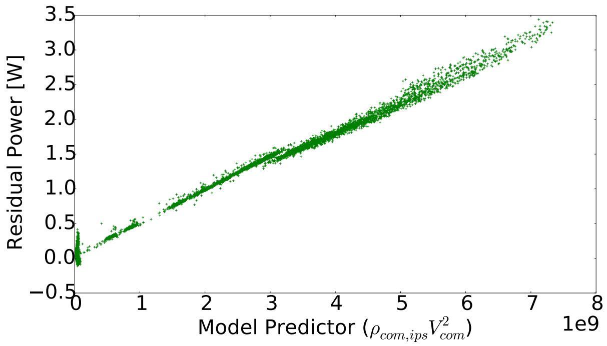


Fig. 5: Residual (workload-specific) power plotted for the model predictor (square-voltage instructions per second).

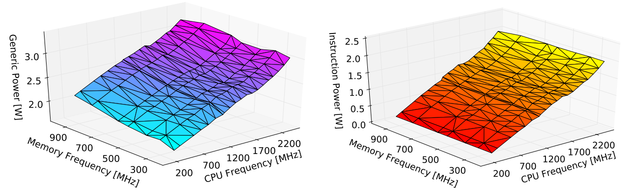
When generic power has been removed from measured power, the residual power P_{res} should reflect only the cost of executing instructions (see Equation 5):

$$P_{res} = C_{com,ips}\rho_{com,ips}V_R^2 \tag{9}$$

In Equation 9, the unknown variable is the per-instruction capacitance loss $C_{com,ips}$. Figure 5 shows the residual power P_{res} for the DCT filter running on four cores. Each residual, which represents a unique sample at an operating frequency point, is plotted for its corresponding predictor ($\rho_{com,ips}V_R^2$ in Equation 9). From the figure, we can see that residual power follows a linear trend with its predictor, which is as expected from Equation 9 and confirms our theory that instruction power can be modelled this way. At the data points around $3E9$, we can see that the residuals "drop" by some 50-60 mW. This is because the rail voltage reading, which is stored as a frequency-dependent static table in our code, is slightly incorrect: buck regulator output voltage is unpredictive and varies with output current. This is also visible in Figure 3d at around 1 GHz CPU frequency.

The gradient of the line in Figure 5 is the estimate of switching capacitance $C_{com,ips}$. This is easily found using regression, which yields a capacitive load of $439\frac{pC}{V}$ per instruction. We also use the estimated constant offset in the model, which is rarely above 30-40 mW. For the rest of the filters, estimated capacitive load is never above $607\frac{pC}{V}$ (HP, single-core DCT) and never below $197\frac{pC}{V}$ (LP core, huffman). In general, for all the filters, the more cores are active the higher the capacitive instruction load.

The capacitive load itself does not offer any insight into the actual power usage of instructions. This depends on how many of these are executed per second. To investigate this, we plot the generic and workload-specific power in Figures 6a and 6b. We see that instructions can account for a substantial fraction of the total estimated power. Not considering base power P_{base}, instruction power can account for up to 50 % of the total power and varies between 0.4 to 2.0 W. This means that instruction power is an important power component, and that care must be taken to estimate it correctly to achieve a high prediction accuracy. However, this may not always be easy to achieve in practice. Our workloads are *cyclic* in that they continuously perform the same work (and instructions) on new incoming frames. It is easy to imagine scenarios where it may not be so easy to estimate average instruction cost, for example for workloads which are not cyclic or incorporate heavy branching (not always following the same code paths). It is for example tempting to hypothesise that instruction cost could be estimate along code branches. SoCs with fine-grained instruction accounting (as for the Tegra K1's GPU) can potentially solve this problem by enabling fully generic models, but in practice, it is likely that there will always be SoCs which only implement HPCs for the total number of executed instructions.



(a) Generic components.
(b) Instructions.

Fig. 6: Estimated power.

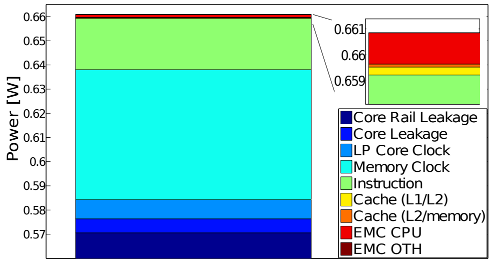


Fig. 7: Idle system power breakdown.

C. Use Cases

We now conduct a set of experiments to show how high-precision power modelling can be useful to developers and system architects. A basic use case is to produce power breakdown of the system, in order to analyse the power consuming components and identifying important candidates to minimise power usage. For example, Figure 7 shows the contribution to power from each of the generic power components. The system is idle, and under the control of the standard power management algorithms where the CPU and memory are running generally underutilised at the same frequency (204 MHz). We can see that cache maintenance and off-chip RAM accesses

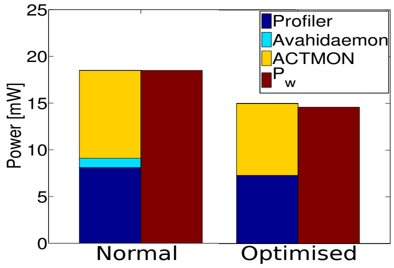


Fig. 8: Instruction power breakdown of the three dominating idle system services.

are virtually non-present. Workload power is negligible, only accounting for about 20 mW. The memory clock accounts for the second most substantial power component. This is interesting as the capacitance loss per cycle is comparative with the CPU and they are both running at the same frequency. However, the CPU is better at handling underutilisation with clock- and power-gating and is consequently better at hiding the unnecessary power overhead. The memory, however, is not clock-gated and therefore ends up wasting cycles which are not used. Among the static power contributors, the core rail leakage current is very large at 570 mW, but little can be done to this component other than ensuring that the core rail voltage is as small as possible.

Our methodology is also fine-grained enough to estimate instruction power of individual system services. To show that it is able to handle very small changes in instruction power, we focus on the three most dominating system services for the idle system. The residual workload power from these is only 20 mW and the breakdown can be seen to the left in Figure 8 for the profiler (our modelling and estimation tool), the avahidaemon and an activity monitor which logs RAM utilisation. Studying the source code for the activity monitor, we found several optimisation points where we were able to remove expensive string comparison functions. We also disabled the avahidaemon, which is unnecessary. The result shows that we are able to reduce residual power by 20 % (P_w). While the actual impact of the system is small (some 5 mW), this proves that the model is indeed fine-grained and is able to pick up on very small changes in the system.

VI. Conclusion

Power models are important tools to diagnose modern embedded multicore SoCs at early design stages or for runtime power analysis and management. This is because it is not trivial to measure and attribute power of individual computational units, such as cores, caches, clocks and memory to applications. In this paper, we have shown that state-of-the-art power modelling methods can mispredict substantially on the Tegra K1. Our main contribution is a methodology which, by considering rail voltages and fine-grained hardware activity measurements, is able to predict power with an accuracy close to 100 %. It is not possible to build a fully generic model for the Tegra K1 because of a lack of fine-grained CPU instruction accounting. Instead, we extend our method from previous work [18] and show how instruction power can be measured on a per-process basis using only a single HPC for the total number of instructions executed. It has been extensively verified over all frequencies using several video processing workloads.

References

[1] R. Basmadjian and H. de Meer. Evaluating and Modeling Power Consumption of Multi-Core Processors. In *Proc of e-Energy*, pages 1–10, 2012.

[2] A. Carroll and G. Heiser. An Analysis of Power Consumption in a Smartphone. In *Proc of ATC*, 2010.

[3] A. Castagnetti, C. Belleudy, S. Bilavarn, and M. Auguin. Power Consumption Modeling for DVFS Exploitation. In *Proc of DSD*, pages 579–586, 2010.

[4] Cisco. Cisco Visual Networking Index: Forecast and Methodology, 2014-2019, White Paper. http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-481360.html.

[5] M. Dong and L. Zhong. Self-Constructive High-Rate System Energy Modeling for Battery-Powered Mobile Systems. In *Proc of MobiSys*, pages 335–348, 2011.

[6] L. M. Feeney. An Energy Consumption Model for Performance Analysis of Routing Protocols for Mobile Ad Hoc Networks. *Ad Hoc Networks*, pages 1–13, 1999.

[7] S. Hong and H. Kim. An Integrated GPU Power and Performance Model. In *Proc of ISCA*, volume 38, pages 280–289, 2010.

[8] W. Jung, C. Kang, C. Yoon, D. Kim, and H. Cha. DevScope: a Nonintrusive and Online Power Analysis Tool for Smartphone Hardware Components. In *Proc of CODES+ISSS*, pages 353–362, 2012.

[9] N. S. Kim, T. Austin, D. Blaauw, T. Mudge, K. Flautner, J. S. Hu, M. J. Irwin, M. Kandemir, and V. Narayanan. Leakage Current: Moore's Law Meets Static Power. *IEEE Computer*, pages 68–75, 2003.

[10] NVIDIA. Tegra K1 Circuit Schematics, Rev. 4.02. Technical report.

[11] NVIDIA. Tegra K1 Technical Reference Manual. Technical report.

[12] NVIDIA. Variable SMP – A Multi-Core CPU Architecture for Low Power and High Performance. Technical report, 2011.

[13] A. Pathania, Q. Jiao, A. Prakash, and T. Mitra. Integrated CPU-GPU Power Management for 3D Mobile Games. In *Proc of Design Automation Conference*, pages 1–6, 2014.

[14] M. Pricopi, T. S. Muthukaruppan, V. Venkataramani, T. Mitra, and S. Vishin. Power-Performance Modeling on Asymmetric Multi-Cores. In *Proc of CASES*, 2013.

[15] K. R. Stokke, H. K. Stensland, P. Halvorsen, C. Griwodz. Why Race-to-Finish is Energy-Inefficient for Continuous Multimedia Workloads. In *Proc of MCSoC*, pages 57–64, 2015.

[16] T. Simunic, L. Benini, P. Glynn, and G. De Micheli. Dynamic power management for portable systems. In *Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 11–19, 2000.

[17] K. R. Stokke, H. K. Stensland, C. Griwodz, and P. Halvorsen. Energy Efficient Continuous Multimedia Processing Using the Tegra K1 Mobile SoC. In *Proc of MoViD*, pages 15–16, 2015.

[18] K. R. Stokke, H. K. Stensland, P. Halvorsen, and C. Griwodz. A High-Precision, Hybrid GPU, CPU and RAM Power Model for Generic Multimedia Workloads. In *Proc of MMSys*, 2016.

[19] Y. Xiao. *Modeling and Managing Energy Consumption of Mobile Devices*. PhD thesis, 2011.

[20] Y. Xiao, R. Bhaumik, Z. Yang, M. Siekkinen, P. Savolainen, and A. Yla-Jaaski. A System-Level Model for Runtime Power Estimation on Mobile Devices. In *Proc of GCC and CPS*, pages 27–34, 2010.

[21] F. Xu, Y. Liu, Q. Li, and Y. Zhang. V-edge: Fast Self-Constructive Power Modeling of Smartphones Based on Battery Voltage Dynamics. In *Proc of NDSI*, pages 43–55, 2013.

[22] D. You and K.-S. Chung. Quality of service-aware dynamic voltage and frequency scaling for embedded GPUs. *IEEE Computer Architecture Letters*, 14(1):66–69, 2015.

[23] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang. Accurate Online Power Estimation and Automatic Battery Behavior Based Power Model Generation for Smartphones. In *Proc of CODES+ISSS*, pages 105–114, 2010.