Enabling Accurate and Efficient Modeling-based CPU Power Estimation for Smartphones

Yifan Zhang
SUNY Binghamton
Binghamton, NY, USA

Yunxin Liu
Microsoft Research
Beijing, China

Xuanzhe Liu
Peking University
Beijing, China

Qun Li
College of William and Mary
Williamsburg, VA, USA

Abstract—**CPU is one of the most significant sources of power consumption on smartphones. Power modeling is a key technique and important tool for power estimation and management, both of which are critical for providing good QoS for smartphones. However, we find that existing CPU power models for smartphones are ill-suited for modern multicore CPUs: they can give high estimation errors (up to 34%) and high estimation accuracy variation (more than 30%) for different types of workloads on mainstream multicore smartphones. The cause is that the existing approaches do not appropriately consider the effects of CPU idle power states on smartphones CPU power modeling. Based on our extensive measurement experiments, we develop a new CPU power modeling approach that carefully considers the effects of CPU idle power states. We present the detailed design of our power modeling approach, and a prototype CPU power estimation system on commercial multicore smartphones. Evaluation results show that our approach consistently achieves higher power estimation accuracy and stability for various benchmarks programs and real apps than the existing approaches.**

I. Introduction

Battery life is among the most important factors that affect the quality of service (QoS) provided by battery-powered mobile devices such as smartphones. CPU is a major source of energy consumption on smartphones [7]. For example, a recent study shows that, on a quad-core Samsung Galaxy S3 smartphone, the CPU power is as high as 2,845 mW, which is 2.53 times of the maximum power of the screen, and is 2.5 times of the maximum power of the 3G interface [8]. Therefore, accurate estimation and efficient management of CPU power consumption are among the most important issues in power management of multicore smartphones.

Power modeling is a lightweight and effective approach to estimate smartphone CPU power consumption. Proper and accurate power models of smartphone components benefit both users and developers. For example, accurate power models help to detect power hungry applications, and thus users get better battery life of their smartphones [1]. Accurate power models also help developers profile and optimize the energy consumption of their smartphone applications [16]. Because of its importance, power modeling has been attracting an increasing amount attention recently [10], [14], [19]–[22].

Existing CPU power modeling approaches for smartphones assume CPU operating frequency and CPU utilization are the major factors that impact CPU power consumption [20]–[22]. However, we find that this assumption does not hold with multicore CPUs on modern smartphones. Specifically, even under the same frequency and CPU utilization, two workloads with different CPU usage patterns (for example, as shown in Figure 1) can consume very different amounts of energy. Our experiments show that the difference can be as large as 50% on a quad-core Google Nexus 4 smartphone (§III). Therefore, current smartphone CPU power models are ill-suited for modern multicore smartphones. For example, we will later show that the current CPU power models can give an estimation error as high as 34% (§VI). Moreover, the same power model can have strikingly different estimation accuracy for different types of workloads: the variation can be larger than 30% (§VI). The root cause of this estimation inaccuracy and instability stems from the CPU *idle power states*, which consume markedly different amounts of power in multicore CPUs. Workloads with different CPU usage patterns cause the CPU to enter different idle power states during the computation, which in turn leads to a different amount of CPU power consumption. Since existing CPU power modeling methods do not take into account the impact of CPU idle power states, they exhibit high estimation errors and instability for multicore smartphones in practice.

We carefully analyzed the impact of idle power states on CPU power consumption, and developed a new CPU power-modeling method that treats CPU idle power states as a major factor of smartphone CPU power modeling. As a result, the new modeling method is able to significantly improve power estimation accuracy and stability. The model building process is non-trivial indeed. As we will show in §IV, simply using durations or numbers of entries of CPU idle states does not work. Instead, we have found and experimentally verified that *weighted average entry duration* of CPU idle states is a good predictor to estimate the power consumption of a multicore CPU. Based on the proposed model, we have designed and implemented a prototype CPU power estimation system for commercial state-of-the-art multicore smartphones. We have also conducted extensive experiments to evaluate our prototype system using a set of commercially representative embedded benchmarks, as well as real mobile applications. The evaluation results show that our method achieves a consistent and high average accuracy of 98% for various benchmarks, and 96% for real applications, with negligible system overheads.

Our main contributions can be summarized as follows.

• We identify that existing CPU power modeling methods for smartphones can be notably inaccurate and unstable in

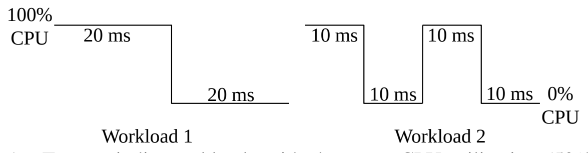


Fig. 1. Two periodic workloads with the same CPU utilization (50%) but different CPU usage patterns.

estimating power consumption of multicore CPUs. We analyze the root cause of the estimation inaccuracy and instability, and show that different CPU idle power states, which are not considered in existing CPU power modeling approaches, have big impacts on CPU power consumption.

• We propose and develop a new *idle-state-based* CPU power modeling method for accurate CPU power estimation on multicore smartphones. We demonstrate that simply considering durations and numbers of entries of CPU idle states do not work, and we show that *weighted average entry duration* of CPU idle states is a good predictor to estimate the CPU power consumption on multicore smartphones.

• We design and implement a prototype CPU power estimation system based on the proposed model using commercial multicore smartphones. We also conduct extensive experimental evaluations to evaluate our prototype system. The experimental results show that our system achieves high accuracy and incurs a negligible amount of overheads.

II. Background and Related Work

A. Background

A smartphone CPU has different states: a CPU core can be either *online* or *offline* (i.e., powered down). An online CPU core can further work in either the *operating state* or an *idle state*. Smartphone OS manages the states of CPU cores to reduce their total energy consumption. There are three CPU power management schemes used on modern smartphones: *CPU performance state management*, *CPU idle state management*, and *CPU hot-plugging*. We briefly introduce the first two, which are more related to this work.

CPU performance state management. When a CPU core works in the operating state, all processor components are powered up. In the operating state, a CPU core may operate in different *performance states* (also known as "*P-states*" in the ACPI specification [12]). Practically, each P-state is associated with a fixed CPU operating voltage and frequency. A technique called Dynamic Voltage and Frequency Scaling (DVFS) is employed to adjust the operating voltage/frequency, and thereby switch between different P-states.

In Android kernel (Linux-based), the "*CPUfreq*" subsystem specifically copes with this task by dynamically adjusting the operating frequency according to the system load [15].

CPU idle state management. Smartphone OS may put an online CPU core into an idle state when there is no workload. CPU idle states are called "*C-states*" in the ACPI specification [12]. CPU in different C-states has different CPU components switched to low power mode to reduce power consumption.

Table I shows the four CPU idle power states of the Google Nexus 4 smartphone: C0 C1, C2, and C3. A CPU core in the state C0 only disables most of the CPU clocks, while

TABLE I
CPU Idle Power States on Nexus 4.

Idle State	Name	Idle System Power (mW)	Latency (μS)†
C0	Wait for Interrupt	433	1
C1	Retention	390	415
C2	Power Collapse Standalone	330	1300
C3	Power Collapse	200	2000
Without entering idle states		1,060	0

†: The data is obtained from the Nexus 4 kernel source code.

keeping the core logic powered up. A core in the state C1 has its logic powered down, but retains the in-core L0/L1 cache content by keeping the cache powered up. A core in the state C2 has more power savings than when in the C1 state, since the in-core L0/L1 cache are also flushed and disabled. Finally, a core in the state C3 achieves the most power savings by further disabling the shared L2 cache. We have measured the idle system power of each C-state in Nexus 4. The third column of Table I shows the results. As a comparison, we have also measured the case of not entering C-states, where the idle system power is 1,060 mW. Entering a C-state can save much power when a system is idle. It also shows that power consumption of different C-states varies: the power of C0 is as much as 2.1 times of the power of C3. Consequently, entering different C-states may cause significantly different power savings. In old single-core smartphones, there are fewer CPU idle power states. For example, the Google Nexus S smartphone (single-core) has only one idle state, which is equivalent to the C0 state on Nexus 4. Therefore, CPU idle states do not play a critical role in CPU power consumption on old single-core smartphones as they do on modern multicore smartphones.

Although entering idle power states reduces power consumption when a CPU is idling, it comes with a price of state switching overhead: the deeper an idle state is, the larger the switching overhead will be. The fourth column of Table I shows the latencies of switching between the operating state and an idle state.This operating/idle state switching latency has significant impact on performance of time-critical operations, such as video and audio decoding.

In Android kernel, the "*CPUidle*" subsystem is specifically designed for managing the CPU idle states.When the OS finds no task to schedule, it directs the control to the *CPUidle* subsystem, which then decides to put CPU into a proper idle state based on several factors, including the predicted length of the current idle period (based on the information on the kernel scheduler and timers) and the operating/idle switching latency of each individual idle state.

B. Related Work

Existing approaches for modeling CPU power consumption can be classified into two categories as below.

CPU frequency/utilization based approaches. Existing approaches for modeling CPU power consumption [9], [10], [14], [18], [20], [22] on smartphones are all CPU frequency and utilization based. They assume CPU frequency and utilization as two major factors impacting CPU power consumption. While this assumption works well for single-core smartphones, where CPU idle states have little impact on CPU power, it does not hold for multicore smartphone with multiple CPU idle states, in which power consumptions are significantly different.

Some existing approaches of CPU power modeling also consider CPU idle states [10], [19]. Specifically, Koala [19] proposes a model based approach to estimate runtime system power. In this approach, CPU idle states are considered as a factor affecting system power consumption. However, Koala only considers the time duration of each idle state, while ignoring overheads of the operating/idle transitions. As we will show later, even for two workloads with the same CPU frequency/utilization and the same residency of idle states, the CPU power consumption could have more than 20% difference. Moreover, it only reports evaluation results on the x86 architecture. Sesame [10] also considers CPU idle states in modeling CPU power consumption. However, it does not provide description about how this particular information is used in the modeling process. Similar to Koala, the idle states are only considered in the laptop model (x86-based) in Sesame. In our work, we focus our attention on measuring/investigating the impacts of CPU idle state on ARM-based smartphone CPUs. We also developed a new idle-state-aware CPU power modeling approach based on the investigation results.

CPU hardware events based approaches. Another way of performing CPU power modeling is to model the relationship between CPU power and CPU hardware events [5], [6], [13], [17]. For example, Power Containers [17] considers a linear model between CPU power consumption and a series of hardware events, including retired instructions, floating point operations, last-level cache requests, and memory access. While the CPU hardware events based approaches work well for PC or server CPUs, whose ISA are mostly x86 based, they cannot be applied in current smartphones. This is because although many hardware events are recommended to be implemented in the hardware monitor by the ARMv7 architecture specification [4], only very few of them are mandated. For example, in the CPU used by the Nexus 4 smartphone, only the hardware events of instruction rate, number of instructions retired, and branches executed and missed are implemented, which is not enough to support the hardware events based modeling.

III. Motivation: Limitations of the Existing Smartphone CPU Power Models

The existing smartphone power models [10], [14], [19]–[22] have achieved a good accuracy (e.g., more than 90%) on previous single-core smartphones, such as the Nexus one and Nexus S smartphones. These models consider only CPU utilization and operating frequency as predictors in modeling [20]–[22]. Usually, they use a linear CPU power model: for each CPU frequency $freq_i$, the power consumption of a CPU core is estimated as

$$P_{cpu} = \alpha_{freq_i} \times U_{cpu} + \beta_{freq_i} \quad (1)$$

where U_{cpu} is the CPU core utilization, and α_{freq_i} and β_{freq_i} are two constant parameters whose values are determined through linear regression during the model generation process.

However, the existing CPU power models are not suited for modern multicore CPUs. In particular, we find that CPU power consumption can exhibit a large range of variation even when both CPU frequency and utilization are fixed. In our experiments performed on quad-core Nexus 4 and octa-core Samsung Galaxy S4, we develop a *workload generator program* that periodically performs continuous computation followed by an idle period (see Figure 1). By controlling the ratio of the idle period to the computation period, the workload generator program generates workloads with different CPU utilizations. In the continuous computation, the program runs a busy loop of computing a large prime. By changing the busy loop count, we can also control the length of each continuous computation period. We find that by adjusting the length of each continuous computation, the power consumption of a CPU core exhibits a large range of variation, even when the CPU operating frequency and the utilization were fixed. For example, Figure 2(a) shows the power consumption of a CPU core of the Nexus 4 smartphone when the operating frequency was fixed at 384 MHz. With fixed CPU utilization, the power consumption of the CPU core dropped while the duration of the continuous computation increased. Figure 2(b) and Figure 2(c) show the results when CPU frequency was 1,026 MHz and 1,512 MHz, respectively. They show exactly the same trend. Figure 2(d) further summarizes the difference of power consumption with the three CPU frequencies. Each value in Figure 2(d) is the percentage of the difference between the maximum and minimal powers over the maximum power for each frequency/utilization configuration. It shows that the CPU power difference of workloads causing the same CPU utilization under the same CPU frequency is significant, especially when the CPU utilization is at a low level: when frequency/utilization is fixed at 1,512 MHz/25%, the power difference can reach as high as 50%. As we will explain later, this is because *the less a CPU core is being utilized, the more chance the CPUIdle subsystem puts the CPU core into a deeper idle state, which causes larger power consumption variation when running different workloads*.

The above results demonstrate that using only CPU operating frequency and utilization is not enough to build an accurate CPU power model for multicore smartphones. On modern multicore smartphones like Nexus 4, CPU power is determined not only by the CPU frequency and utilization, but also by the CPU *idle power states* which are not considered in the existing smartphone CPU power models. Modern multicore CPUs like the one of Nexus 4 have multiple idle power states which have significantly different power consumptions. When utilization is fixed, prolonging the duration of continuous computation causes the corresponding idle period to increase accordingly. Longer idle period allows the OS to put the CPU core into deeper idle states more frequently, which in turn lowers the CPU power consumption.

To further demonstrate how CPU idle power states can affect power consumptions of different workloads running with

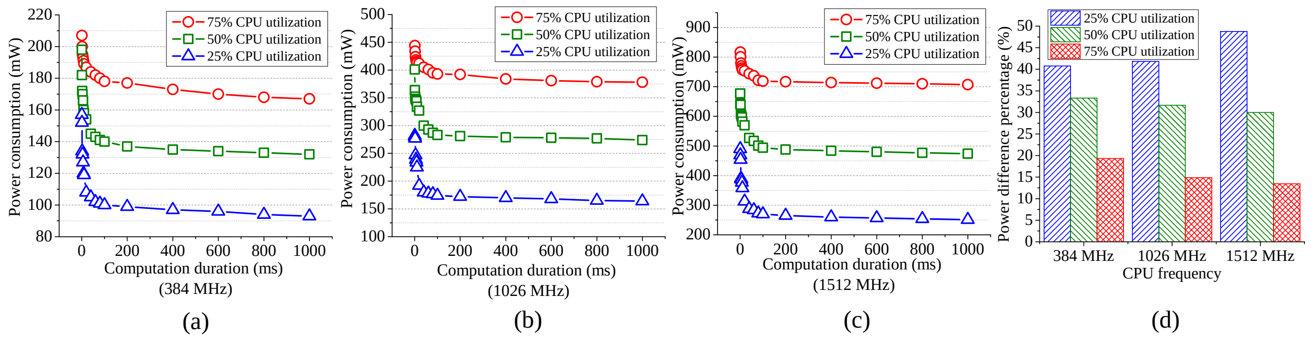


Fig. 2. Workloads running in multicore CPU with the same CPU utilization and frequency consume notably different amounts of CPU power.

TABLE II
TIME DURATION PER SECOND AND NUMBER OF STATE ENTRIES PER SECOND IN TWO WORKLOADS OF THE SAME CPU UTILIZATION (50%) UNDER THE SAME CPU OPERATING FREQUENCY (1,512 MHZ).

Idle State	Time duration (ms)		# of state entries	
	W1	W2	W1	W2
C0	491.85	1.08	468	1.99
C1	0	0	0	0
C2	1.18	1.43	0.1	0.2
C3	5.12	496.86	0.2	7.3

the same CPU frequency/utilization, we list in Table II the statistics of the idle states of two workloads (W1 and W2) that were run on a Nexus 4 smartphone: the time duration per second of each state, and the total number of entries per second of each state. These two workloads were run with the same CPU frequency (1,512 MHz) and the same CPU utilization (50%), but they had significantly different power consumptions (644 mW for W1, and 499 mW for W2). The two workloads had notably different idle-state transition statistics as shown in Table II: with the workload W2, the CPU core stayed at the deepest idle state much longer than with the workload W1. This explains why W2 consumed significantly less CPU power than W1. Note that because the stock Nexus 4 kernel does not enable the idle state C1, the numbers of C1 in Table II are 0s.

We also performed the above experiments on a Samsung Galaxy S4 smartphone, which is equipped with a chipset different from Nexus 4, and obtained similar observations.

IV. THE PROPOSED CPU POWER MODEL

In this section, we first present the development of our power modeling approach for the single-core case, followed by the discussion that how the single-core power model can be extended to the multicore case. All the experiments described in this section are performed on a Nexus 4 smartphone.

A. Power Modeling for a Single CPU Core

Similar to existing work, we use regression-based method to integrate the predictors. To determine what statistics of CPU idle states should be used as a predictor variable of the regression model, we first consider T_{C_i}, which is the *total time duration* that a CPU core stays in the idle state C_i per second when frequency f and utilization U are fixed. Suppose the total CPU idle time per second is T_{idle}, we have

$$T_{idle} = \sum_i T_{C_i} \tag{2}$$

Figure 3(a) shows T_{C_i} for idle states C_0 to C_3 when we ran our workload generator program on a single CPU core (with $f = 1,512$ MHz, $U = 75\%$). Since the stock Nexus 4 kernel does not enable the idle state C_1, statistics of C_1 remain zero in Figure 3. Figure 3(a) shows that the CPU core spent more time staying in deeper idle states as duration of the continuous computation increased, because the idle period also increased accordingly. However, T_{C_i} is not a good predictor of CPU power consumption. For example, after the computation duration increased to 20 millisecond, T_{C_i} $(i = 0, 1, 2, 3)$ stayed stable, but the CPU power actually kept decreasing as the computation duration increased (see Figure 2(c)). In fact, in our experiment, the power difference could reach 24% for the same T_{C_i} $(i = 0, 1, 2, 3)$ (when $f = 1,512$ MHz, U=25%).

Figure 3(b) shows E_{C_i}, which is the *number of entries* for idle state C_i per second, in the same experiment. For the same T_{C_i}, smaller E_{C_i} means less operating/idle transition energy overhead, and thus more energy savings. This explains our previous observation that CPU power kept decreasing when T_{C_i} is unchanged. However, E_{C_i} alone is also not a good predictor of CPU power consumption, as it has no direct link to energy savings by idle states.

We then look at the *average entry duration* for idle state C_i, which is notated as ED_{C_i}:

$$ED_{C_i} = \frac{T_{C_i}}{E_{C_i}} \tag{3}$$

Generally, ED_{C_i} should be a good predictor of CPU power, as it involves both idle state duration and state transition overhead. However, ED_{C_i} could suffer from noise, which comes from those sporadic entries of idle state C_j when the CPU remains in state C_i most of the time. For example, Figure 3(c) shows ED_{C_i} in the experiment. We can see that ED_{C_3} was greater than ED_{C_0} when C_0 is the dominant idle state.

To eliminate noises in ED_{C_i}, we apply a weight w_i, which is the portion of time the CPU stay at the state C_i over the whole idle period, to ED_{C_i} to form *weighted average entry duration* WED_{C_i}:

$$WED_{C_i} = w_i \times ED_{C_i}, \ where \ \ w_i = \frac{T_{C_i}}{T_{idle}} \tag{4}$$

Figure 3(d) shows WED_{C_i} in the experiment. From the figure we can see that *weighted average entry duration* is a good predictor of CPU idle state's impact on CPU power consumption: when CPU stays in idle state C_i, WED_{C_i} increases as CPU

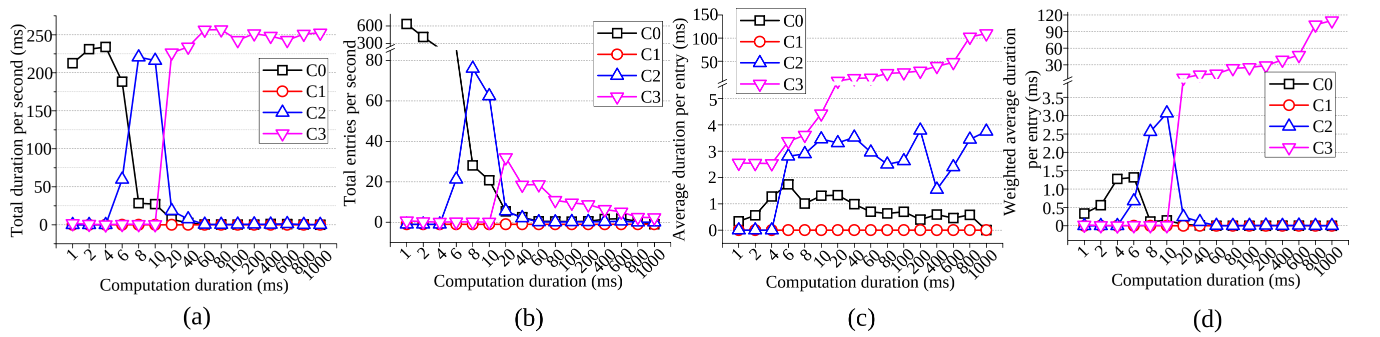


Fig. 3. Single-core power model development. Figures (a)-(d) show T_{C_i}, E_{C_i}, ED_{C_i}, and WED_{C_i} for the four CPU idle states C_0 - C_3, respectively (with CPU frequency $f = 1,512$ MHz, utilization $U = 75\%$).

TABLE III
CPU POWER WITH DIFFERENT NUMBERS OF CORES RUNNING (WITH UTILIZATION U=50%).

N_c	f=384 MHz			f=1512 MHz		
	P_{BL,N_c} (mW)	P_{CPU} (mW)	$P_{\Delta,core}$ (mW)	P_{BL,N_c} (mW)	P_{CPU} (mW)	$P_{\Delta,core}$ (mW)
1	62	144	82	62	495	433
2	73	213	70	73	902	415
3	73	282	70	73	1,312	413
4	73	348	69	73	1,732	415

N_c: number of cores that ran the workload.
P_{BL,N_c}: baseline CPU power with N_c cores enabled.
P_{CPU}: whole CPU power.
$P_{\Delta,core}$: power increment per core.

spent more time in C_i, while WED_{C_j} $(i \neq j)$ remains zero. As a result, we model power consumption of a single CPU core working at frequency f as

$$P_{core} = \sum_i \beta_{C_i} \cdot WED_{C_i} + \beta_U \cdot U + c \tag{5}$$

where β_{C_i} and β_U are the coefficients of WED_{C_i} and the utilization U, and c is a constant. For each CPU frequency f supported by Nexus 4, we obtain the coefficients and the constant by running linear regression analysis on the training data containing different T_{C_i} and U, and the corresponding P_{core} (see §V).

B. Power Modeling for Multicore CPU

We further conduct an experiment to study how the single-core CPU power model can be extended to multicore scenario. In the experiment, we enabled different numbers of CPU cores, which were running at the same frequency, and then generated the same amount of workload on each enabled core. We measure the CPU power while varying the core frequencies and utilization. Table III presents the results for the cases when core frequencies are fixed at 384 MHz and 1,512 MHz, and the core utilization was 50%. In the table, the power increment per core is calculated as $P_{\Delta,core} = \frac{P_{CPU} - P_{BL,N_c}}{N_c}$, where N_c is the number of cores enabled, P_{BL,N_c} is the baseline CPU power when N_c cores are enabled, and P_{CPU} is the whole CPU power measured. We can see that $P_{\Delta,core}$ is consistent for the same "frequency/utilization" with more than one core enabled, but is notably smaller than the value when there is only one core running the workload. The reason is that on Nexus 4, when there are more than one core running, the deepest CPU idle state each running core can enter is state C_2. The state C_3, where the shared L2 cache is disabled, can only be entered by core-0 when no other core is online. Therefore, $P_{\Delta,core}$ for the single-core case is always greater than that for the multicore case.

Based on our observation, we model a multi-core CPU power consumption P_{CPU} as

$$P_{CPU} = P_{BL,N_c} + \sum_i^{N_c} P_{\Delta,core,U_i,f_i} \tag{6}$$

where N_c is the number of cores enabled, P_{BL,N_c} is the baseline CPU power with N_c enabled cores, and $P_{\Delta,core,U_i,f_i}$ is power increment of core-i when it is working at frequency f_i with utilization U_i. For each frequency f_i, $P_{\Delta,core,U,f_i}$ can be predicted using the single-core power model developed previously, while P_{BL,N_c} is a constant value that can be measured beforehand. For Nexus 4, we need to model $P_{\Delta,core_U,f_i}$ separately for the case when there is only one core is online and when there are multiple cores are online, because these two cases have different sets of CPU idle states.

V. Prototype Smartphone CPU Power Estimation System

We have designed and implemented a prototype CPU power estimation system using our idle-state-based CPU model on Android platform. Figure 4 shows an overview of the system. The system contains two parts: one runs in the kernel space, and the other runs in the user space. In kernel space, the *data collector* component collects necessary CPU usage data including the CPU frequency, CPU utilization, and CPU idle state statistics. In user space, the *controller* component controls the procedure of model generation. To generate a CPU power model, the controller runs a set of training programs, starts the data collector, and collects CPU usage data. At the same time, we measure the system power consumption using a power meter, with the smartphone configured in a way that CPU is the only major hardware component consuming system power (see §VI-A). Using the measured power data and the collected CPU usage data, the *model generator* component creates a CPU power model through linear regression. Although our implementation is based on Android platform, we expect the system design can also work on other mobile platforms such as Windows Phone and iOS.

Collecting data in OS kernel. The data collector is designed to work in OS kernel lightweight and efficient

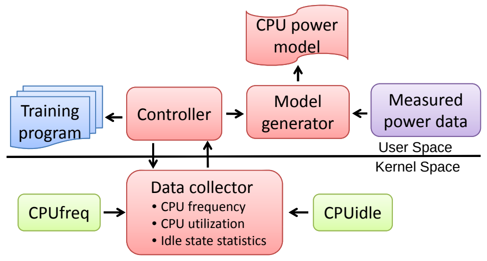


Fig. 4. CPU power estimation system overview.

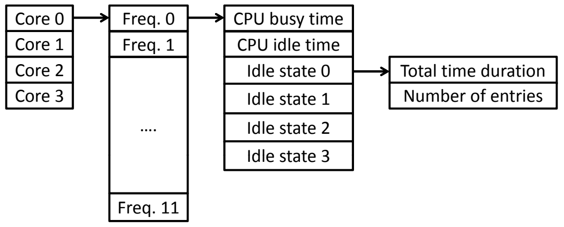


Fig. 5. Data structure used in the data collector.

data collection. An alternative is to periodically sample CPU utilization and CPU idle states in the user space via the high-latency */proc* and */sys* filesystems. However, since our power modeling approach needs CPU statistics for each working frequency, which may change tens of times per second, the user space alternative would need to poll the kernel with an equally high frequency, which is impractical and inefficient. With the kernel-mode data collection approach, we can aggregate raw data, and report only the aggregated data to the user programs via the system call interface. Consequently, this approach can significantly reduce the number of user-kernel mode switchings, and incurs much less system overheads in collecting the data. Moreover, running the data collection in the kernel allows us to obtain fresh and accurate data without the latency of user-kernel mode switching.

The need of our approach to gather CPU statistics for each working frequency also imposes a practical challenge on data sampling. An accurate estimation would require the sampling rate to be set to the highest possible value of frequency changing rate, which can incur unnecessary system overheads, since the sampling would be always done in a high rate even when the actual CPU frequency changing rate is low. To address the challenge, we took a different approach: we take advantage of the *CPUfreq* and *CPUidle* subsystems of the Android kernel to collect data efficiently. Specifically, we piggyback our data collecting with activities of these two subsystems. We instrument the subsystems so that we know when the CPU frequency or CPU idle state are changed. Each frequency change in the *CPUfreq* subsystem triggers a new process of data collection for the new working frequency. For each CPU idle state change in the *CPUidle* subsystem, we collect new data about the previous CPU idle state and aggregate them to the existing data. As a result, our data collection can automatically adapt to CPU frequency changes, and thus avoids unnecessary system overheads.

Figure 5 shows the data structure used in our data collector. We collect the CPU usage data for each CPU core and each CPU frequency separately. Suppose the CPU supports 12 working frequencies. For each CPU frequency, we record the total CPU busy time and the total CPU idle time, based on which the CPU utilization can be calculated. We also record the CPU idle state information, including the total residency time duration and the total number of entries of each CPU idle state. With the data structure in Figure 5, we do not need to record the raw data (e.g., the CPU usage data of every trigger of data collection). Instead, for each trigger of data collection, we simply update the corresponding values in the data structure to aggregate the new data with the existing data. As a result, our approach consumes a small and fixed amount of memory. It uses much less memory when compared to the approach of recording raw data, especially when the data collecting time is long.

Generating CPU power model. To generate a CPU power model, we run a set of training programs with various workloads and CPU usage patterns. We use the workload generator described in §III to create training programs with different CPU frequencies, utilization, and continuous computation durations. For each CPU frequency, we train 3 CPU utilization levels (25%, 50%, and 75%). For each CPU utilization level, we train 8 computation durations (1 ms, 2 ms, 4 ms, 8 ms, 20 ms, 40 ms, 80 ms, and 200 ms). For each CPU frequency, we also train the CPU idle case (5% utilization), and the CPU busy case (100% utilization), but with a fixed computation duration (100 ms). In total we have created 312 different training programs. During the training, we first enable only one CPU core, and run these training programs on the CPU core to generate the single-core power model described in §IV-A. Then we enable all cores, and run the training programs with an identical process on each core, to generate the multicore power model described in §IV-B. The whole model generation procedure takes about 2 hours. We obtain the ground-truth CPU power consumption manually by using a power meter. One could also obtain the ground-truth value by referring to the battery interface [10], [20], which allows for automated model generation.

Applying CPU power model. We develop a user space CPU power estimation C library that supports our CPU model in user space programs. The library gets CPU statistics from the data collector located in the kernel as shown in Figure 4, calculates the estimated CPU power consumption, and reports information to user programs as requested. The interfaces provided by our C library to user programs include starting and stopping the CPU power estimation period, getting the estimated CPU power consumption of the estimation period, and getting different CPU statistics, such as CPU online information, CPU utilization, and CPU idle states information.

In total, our implementation has about 3,000 lines of code (LOC) in C programming language, with 1,300 LOC in kernel implementation and instrumentation, 800 LOC in the controller component, 500 LOC in the CPU power estimation C library, and 300 LOC in the model generator component.

TABLE IV
BENCHMARKS TESTED IN THE EVALUATION.

Benchmark	Description
prime	Compute a large prime.
basicmath	Perform simple mathematical tasks.
qsort	Quick sort over an array of strings.
susan	Susan image recognition.
jpeg	Encode/decode a JPEG image.
dijkstra	The shortest path Dijkstra algorithm.
patricia	Patricia trees of routing tables.
stringsearch	Search for given words in phrases.
sha	SHA secure hash algorithm.
aes	Advanced Encryption Standard (AES).
crc32	32-bit Cyclic Redundancy Check (CRC).
fft	Fast Fourier Transform (FFT).
pcm	Pulse Cod Modulation (PCM).

VI. EVALUATION

A. Experimental Setup

We used a Nexus 4 smartphone, which is equipped with a 1.5 GHz quad-core Snapdragon S4 Pro CPU, 2 GB RAM, and 8 GB internal storage, and runs Android 4.2. We measured the system power consumption using a Monsoon power meter [2]. Since our focus is power consumption of CPU, we disabled other hardware components as much as possible, such as the screen, network interfaces (cellular, WiFi, Bluetooth, and NFC), and sensors (GPS, accelerometer etc.). We also killed all the background services and processes that were not required to run the experiments. Each experiment was repeated 5 times and we report the average results.

The benchmarks. We used 13 benchmarks in our evaluation. The first one is the workload generator described in §III. We call it the *prime* benchmark. By replacing the prime computation part of the benchmark with other types of computation, we created the other 12 benchmarks shown in Table IV. We ported these 12 types of computation from MiBench, a free and commercially representative embedded benchmark suite [11]. These benchmarks cover a diverse set of computation types that are widely used in networking, security, telecommunication, image processing, and many other scenarios and applications. We ran each benchmark for 15 seconds with the CPU utilization selected from 0% to 100% (randomly and fixed, details later), and the busy loop count randomly selected from 1 to 5 during each continuous computation period. Depending on the type of the computation, the length of continuous computation periods ranged from 10 ms to 1000 ms, with 250 ms being the average value.

The real applications. We also used the following 5 applications to evaluate our CPU power model.

• *Web browsing*: we used the Dolphin Browser [3] to load five web pages pre-downloaded from www.nytimes.com. The five pages include the homepage and four subpages. Dolphin Browser is a popular web browser similar to Google's Chrome web browser, both of which are based on the WebKit engine. We chose the Dolphin browser over the Chrome browser because the Dolphin Browser provides more control interfaces, which allow for automated tests.

• *Map*: we used Google Map to browse an offline map with operations including zooming in/out, swiping, and moving the map. We developed an automation tool to capture and replay the user inputs on the touch screen, so that we could perform the desired map operations automatically.

• *App loading*: we launched 8 real apps including Kingsoft Office, ThinkFree Office, Chrome browser, Firefox browser, Opera browser, Google Map, Baidu Map, and Ezpdf reader. We did not choose any games because (1) the loading process of many CPU intensive games (e.g., Angry Birds) terminates when the screen is turned off, and (2) these games usually use GPU for graphic processing, but GPU is not considered in our power model.

• *Video decoding*: we used Dolphin Player to play a local MP4 video clip (30 frames/sec, 611 kbps bitrate) for 20 seconds. We configured the player to perform software video decoding using CPU. The playback was done with screen off and sound volume set to minimal.

• *Audio decoding*: we used Google Music to play a local MP3 song clip (44.1 KHz sample rate, 64 kbps bitrate) for 20 seconds. The Google Music decodes an audio file with software.

Please note that the goal of conducting experiments on real applications is to evaluate how our CPU power modeling and estimation approach works on real app workloads in addition to those ported from MiBench. If one wants to estimate the power consumption caused by a particular app, she also needs to consider power consumption generated by other hardware components (e.g., WiFi, Bluetooth).

CPU power models to compare. To compare our power model (labeled as IM) with the existing CPU power models, we generated 4 utilization based CPU power models (i.e., current CPU power models that consider only CPU frequency and utilization) as follows. We used the same training programs as in our model generation process (§V), but only considered CPU frequency and utilization, ignoring the CPU idle states. The 4 utilization based models (labeled as UM-1, UM-2, UM-3 and UM-4) were generated using 4 different computation durations: 2 ms, 8 ms, 20 ms, and 200 ms, respectively. Once we generated the single-core power models, we further created the corresponding multicore models according to the procedure described in §IV-B. It is worth noting that in previous work, utilization based CPU power models were trained only on single-core CPUs. For fair comparison, we extended the CPU utilization based power models to multicore CPU case using the same method as we used in our approach.

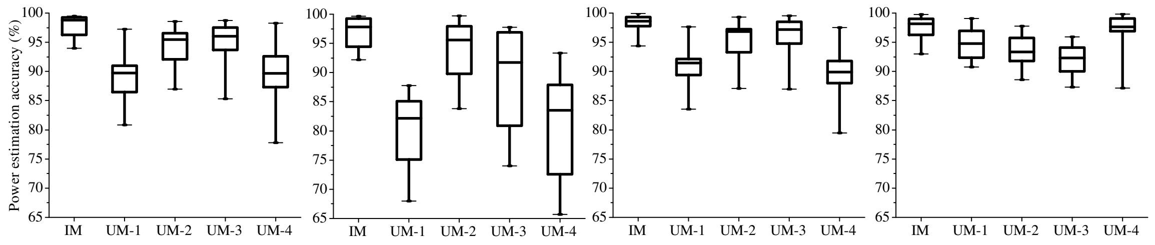
B. Experimental Results

Accuracy of single-core models - benchmark experiment results. We use *power estimation error percentage* (E_{pct}) to quantify the power estimation error made by different power models in the experiments. It is defined as

$$E_{pct} = \frac{100 \times (P_e - P_m)}{P_m}\% \quad (7)$$

where P_e is the power estimated by the power model, and P_m is the power measured using the power meter. Then the *power estimation accuracy percentage* (A_{pct}) is defined as

$$A_{pct} = 100\% - E_{pct} \quad (8)$$



(a) random CPU utilization (b) 30% CPU utilization (c) 60% CPU utilization (d) 90% CPU utilization

Fig. 6. Single-core model estimation accuracy percentage (A_{pct}) for the benchmark experiment.

TABLE V
POWER ESTIMATION ERROR PERCENTAGE OF SINGLE-CORE MODELS FOR THE BENCHMARK EXPERIMENT.

Benchmark	Power estimation error percentage E_{pct}																			
	Random utilization					30% CPU utilization					60% CPU utilization					90% CPU utilization				
	IM	UM-1	UM-2	UM-3	UM-4	IM	UM-1	UM-2	UM-3	UM-4	IM	UM-1	UM-2	UM-3	UM-4	IM	UM-1	UM-2	UM-3	UM-4
prime	0.9	13.5	5.7	-1.9	-10.4	2.2	16.1	-2.1	-19.5	-27.4	0.2	8.2	2.2	-3.7	-12.0	1.4	3.1	4.2	5.9	-2.4
basicmath	-0.7	10.4	5.3	3.4	-7.4	1.9	26.2	10.6	-3.7	-12.5	0.5	11.7	6.7	2.8	-6.0	-0.8	2.5	3.9	5.9	-2.4
qsort	-6.0	-2.7	-7.9	-14.7	-22.2	0.7	14.9	-4.4	-23.3	-30.8	-5.1	-2.4	-7.8	-13.0	-20.5	-7.0	-8.6	-7.3	-5.5	-12.9
susan	-1.0	6.6	-3.6	-8.0	-16.8	2.7	18.0	-4.1	-26.0	-34.3	0.1	7.3	0.9	-4.1	-13.6	3.2	5.2	6.7	8.7	0.2
jpeg	0.6	9.2	2.3	-4.3	-12.7	0.4	17.8	3.9	-7.0	-15.9	0.7	8.6	3.2	-1.9	-10.1	0.7	4.9	6.0	7.7	-0.8
dijkstra	4.9	19.2	13.0	7.9	-1.7	6.5	30.0	15.4	3.1	-6.7	5.6	16.5	12.9	7.0	-2.4	2.3	7.6	11.4	10.9	2.2
patricia	0.8	10.3	4.1	-1.3	-10.2	0.7	13.2	0.3	-10.4	-18.7	-0.9	8.0	3.0	-1.4	-10.7	-1.0	2.9	4.1	5.9	-2.4
stringsearch	1.5	9.8	1.4	-6.3	-15.6	4.0	21.3	-9.0	-19.1	-29.0	-2.1	6.1	-0.7	-5.8	-16.4	4.9	8.4	9.9	11.9	3.1
sha	5.8	18.0	11.6	6.1	-3.5	5.6	32.0	16.2	2.3	-7.2	5.2	14.6	11.3	5.2	-4.0	4.5	9.2	10.7	12.7	3.9
aes	0.5	9.0	3.4	-1.3	-9.9	5.8	17.7	5.8	-2.9	-12.0	1.8	9.8	4.9	0.5	-8.4	0.2	4.5	5.8	7.5	-0.9
crc32	2.2	12.3	4.5	-2.9	-11.5	1.2	12.9	-1.9	-15.4	-23.0	1.3	8.9	3.0	-2.5	-10.8	3.7	7.0	8.2	10.0	1.4
fft	-1.2	8.8	2.8	-2.5	-10.9	0.3	12.2	0.9	-8.3	-16.5	-1.4	7.8	2.8	-1.5	-10.1	-1.8	0.9	2.2	4.1	-4.1
pcm	3.7	15.2	9.2	4.0	-5.5	7.8	24.9	10.2	-2.8	-12.1	2.2	10.6	5.7	1.3	-8.2	1.0	6.2	7.5	9.4	0.9

Figure 6 shows the single-core models' power estimation accuracy percentage in the benchmark experiment, where the CPU utilization of running the benchmark programs was set (by controlling the ratio between the idle period and the computation period of the program) in four different ways: random, 30%, 60%, and 90%. In the figure, the bar in a box is the average accuracy of the model. The upper and bottom borders of a box represent 75 percentile and 25 percentile. The tips of the upper and bottom whiskers represent the max and min values. Figure 6 (a) shows the case of random CPU utilization. On average, our model achieved a high accuracy of 98%, with a small variation ranging from 94% to 100%. The average accuracy and the range of accuracy variation of the four utilization based models were (with the variation range shown in the parenthesis): 89% (81%-97%), 94% (87%-99%), 95% (85%-99%), and 89% (78%-98%), respectively. We can see that our model significantly outperforms the utilization based models in terms of *estimation accuracy* and *accuracy stability*. Although the average accuracy of UM-2 and UM-3 were not far below that of our model, they exhibited a much larger range of accuracy variation for different benchmarks. This is because *different benchmarks have different CPU usage patterns, which further cause different patterns of CPU idle state entries. The utilization based models are not able to capture the effect of these CPU idle state changes, which are important dynamics affecting CPU power consumption. By contrast, our model can well cope with this dynamic usage pattern.*

The accuracy of the existing utilization based models are also subject to CPU utilization. Figure 6 (b), (c) and (d) show more results when the CPU utilization was fixed at 30%, 60%, and 90%. We can see that the utilization based models gave notably high errors in some cases, especially when CPU utilization was at a low value. For example, when the CPU utilization was 30%, the accuracy of model UM-4 can be as low as 66%, and the accuracy of model UM-1 was only 68%. This is because *when CPU utilization is low, there is more idle time, which in turn leads to more dynamic pattern of idle state entries.*

Table V further shows the detailed results about power estimation error percentage for the same experiment. The result suggests that for a given benchmark and certain fixed CPU utilization, it is possible to find a CPU utilization based model that achieves high estimation accuracy. However, that model is likely to suffer high power estimation error in other benchmarks or with other CPU utilization levels. For example, when the CPU utilization was 30%, UM-2 achieved almost 100% estimation accuracy for the benchmark *patricia* (the estimation error percentage was only 0.3%). But UM-2 estimated about 16% more than the ground truth value for the benchmark program *sha*. Another example is that, when the CPU utilization was 60%, UM-3 achieved estimation accuracy of more than 95% for the benchmark *susan* (the estimation was only 4.1% less than the actual value). However, UM-3's estimation was 26% smaller than the ground truth value for the same benchmark program when CPU utilization was 30%. In summary, *it is not possible to have a single CPU utilization based model to achieve a high and consistent modeling accuracy in all the benchmarks and CPU utilization levels. By contrast, our model, which considers CPU idle states and thus can adapt to variation of CPU usage pattern, is able to achieve a consistently high estimation accuracy in*

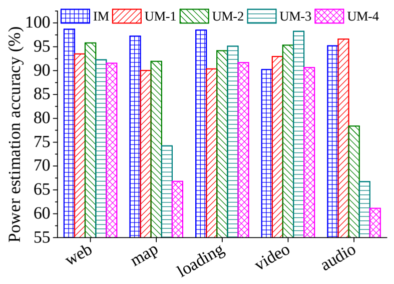


Fig. 7. Single-core model accuracy for the real-app experiment.

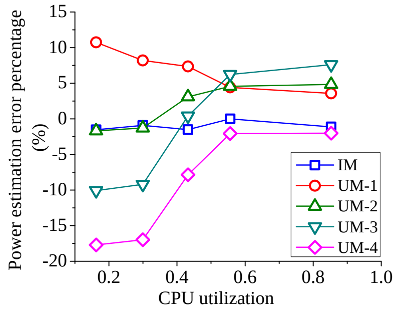


Fig. 8. Power estimation error percentage for the real-app experiment.

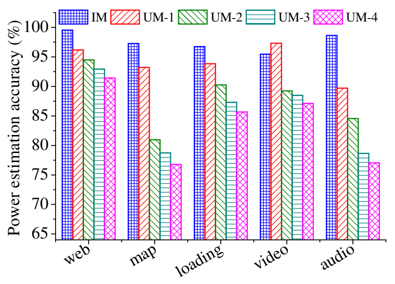


Fig. 9. Multicore model accuracy for the real-app experiment.

all the benchmarks and different CPU utilizations.

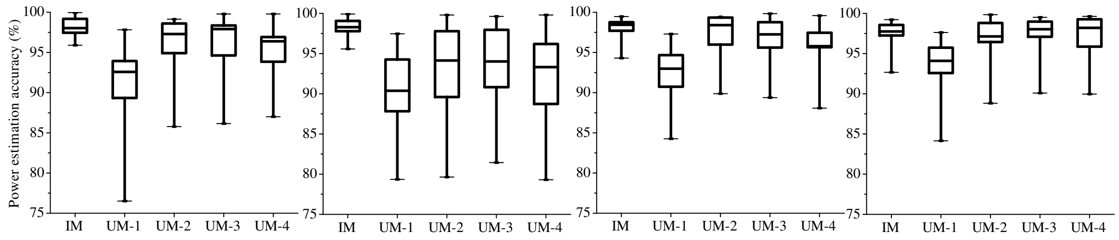
Accuracy of single-core models - real application experiments results. The similar observations can be found in the real application experiments as well. Figure 7 shows the single-core model accuracy in the five real application experiments. We can see that our model also achieved a high accuracy, 96% on average, with a variation ranging from 90% to 99% for different applications. The accuracy is slightly lower than that of the benchmark experiment. This is likely because the applications had more flash disk operations, but our model does not consider flash disk. Our model had the lowest accuracy of 90% in video decoding. This is probably because that the player used GPU which is also not considered in our model. For the utilization based models, their accuracy in the real application experiments exhibited a large range of variation. The average accuracy and the range of accuracy variation were: 93% (90%-97%), 91% (78%-96%), 85% (67%-98%), and 80% (61%-92%), respectively.

We also examined the relationship between power estimation error percentage and CPU utilization for the real application experiments. Figure 8 shows the estimation error percentage of all the models for the Web browsing application under different CPU utilizations. We controlled the CPU utilization by changing the time interval between loading the webpages. We can see that the CPU utilization based models gave a large range of accuracy variation when the CPU utilization was different. In particular, when the CPU utilization was low, they generate low model accuracy. In the figure, the curve of our model is much flatter and the estimation ratios are consistently close to 100%, indicating that our model is also able to adapt to CPU utilization changes and achieve consistent high estimation accuracy under different CPU utilizations. We had the same observation for other applications tested.

Accuracy of multicore models. Figure 10 shows the multicore models' power estimation accuracy percentage in the benchmark experiment, where we ran the benchmarks and the real applications using all the four CPU cores. Table VI shows the detailed data about the corresponding estimation error percentage. Figure 9 shows the results of real-app experiment. Similar to the single-core case, our model achieved higher estimation accuracy, and a much smaller range of accuracy variation than the existing utilization based models. For example, when the CPU utilization was randomly selected, the average accuracy and the corresponding accuracy variation were as follows. For our model, the results were 98% (96%-100%) for the benchmark experiments, and 98% (95%-100%) for the application experiments. For the four CPU utilization based models, the results were 91% (76%-98%), 96% (86%-99%), 96% (86%-100%), and 95% (87%-100%) for the benchmark experiments; and 94% (90%-97%), 88% (81%-95%), 85% (79%-92%), and 84% (77%-91%) for the real application experiments.

Compared to the single-core case, the accuracy percentages of the four CPU utilization based models are relatively higher, and the differences among the four models are relatively smaller. This is because the Nexus 4 smartphone allows only two CPU idle states (C0 and C2) when multiple CPU cores are enabled. Thus, the impact of CPU idle states become smaller. Nevertheless, we still have the same observations as in the single-core case: 1) our model had a consistently high accuracy in all the benchmarks and applications, and significantly outperformed the CPU utilization based models; 2) the CPU utilization based models caused a large range of model accuracy variation for different benchmarks and applications, and gave lower accuracy when the CPU utilization is lower. *As smartphone CPUs are becoming increasingly powerful, smartphone CPU utilization is usually low for the most of time. Thus, the CPU utilization based models tend to generates high errors in practice. By contrast, our idle-state-based CPU power model is able to adapt to CPU usage pattern changes and utilization changes, and thus can accurately estimate CPU power consumption with different workloads and different CPU utilizations.*

System overhead. From the 312 training programs (§V), we chose those that cause the most frequent frequency changes and idle state entries to evaluate the CPU overhead of our system. On average, the chosen workloads incur about 40 frequency changes per second and about 450 entries of CPU idle states. Although our implementation should have the maximum system overhead when running these workloads, we have seen no noticeable CPU usage increase. This is because our data recording and reporting process is extremely lightweight: only several variable updates when a frequency change or idle state entry happens, and the data are reported to user space only at the beginning and end of the power estimation period. As for the memory usage, our prototype implementation use about 8 KB kernel memory, with the



(a) random CPU utilization (b) 30% CPU utilization (c) 60% CPU utilization (d) 90% CPU utilization

Fig. 10. Multicore model estimation accuracy percentage (A_{pct}) for the benchmark experiment.

TABLE VI
POWER ESTIMATION ERROR PERCENTAGE OF MULTICORE MODELS FOR THE BENCHMARK EXPERIMENT.

Benchmark	Power estimation error percentage E_{pct}																			
	Random utilization					30% CPU utilization					60% CPU utilization					90% CPU utilization				
	IM	UM-1	UM-2	UM-3	UM-4	IM	UM-1	UM-2	UM-3	UM-4	IM	UM-1	UM-2	UM-3	UM-4	IM	UM-1	UM-2	UM-3	UM-4
prime	0.8	10.6	1.4	1.1	-0.6	0.1	7.3	-10.4	-9.2	-11.3	1.0	9.3	1.6	1.1	-0.4	-1.7	2.4	-1.8	-2.9	-4.1
basicmath	-2.0	5.4	-1.5	-2.1	-3.5	-1.7	9.6	-2.2	-2.1	-3.8	-3.1	4.3	-4.0	-4.0	-4.2	-2.2	4.3	0.2	-1.0	-2.3
qsort	-4.1	-3.0	-11.3	-11.5	-13.0	-3.5	-2.6	-20.4	-18.6	-20.7	-5.7	-3.3	-10.1	-10.6	-11.9	-5.5	-3.9	-7.8	-8.9	-10.0
susan	1.6	6.9	-3.7	-4.2	-6.2	0.8	3.3	-13.6	-12.7	-14.7	1.7	7.1	-1.0	-2.5	-4.3	1.4	4.8	0.6	-0.5	-1.8
jpeg	0.0	6.0	-0.9	-1.9	-3.4	0.4	5.7	-5.8	-5.8	-7.9	1.3	6.4	-0.7	-1.1	-2.5	1.8	7.1	2.9	1.7	0.4
dijkstra	-1.2	13.2	5.1	4.7	3.1	-2.3	19.4	7.8	7.3	5.7	0.5	10.2	3.5	2.7	1.5	1.1	7.4	3.1	1.9	0.6
patricia	-2.1	23.5	14.2	13.8	12.1	-1.8	20.6	8.6	8.1	6.7	2.2	13.3	6.4	5.5	4.2	2.8	12.5	8.1	6.8	5.4
stringsearch	2.6	9.4	-1.5	-1.9	-3.9	1.0	6.1	-10.9	-10.0	-12.5	1.2	7.0	-1.1	-2.7	-4.4	2.5	5.9	1.7	0.5	-0.8
sha	0.5	7.4	1.4	0.2	-1.2	-1.7	13.4	1.6	1.5	-0.2	1.3	7.1	0.6	-0.1	-1.3	1.2	7.4	3.2	2.0	0.7
aes	3.0	9.7	2.7	1.7	0.2	2.0	10.6	0.2	-0.5	-2.0	1.6	7.0	-0.6	-1.2	-2.8	2.4	7.9	3.6	2.4	1.1
crc32	4.1	15.0	6.5	5.7	4.1	4.4	12.2	-3.6	-3.0	-5.1	5.6	15.8	7.5	7.0	5.4	7.3	15.9	11.2	9.9	8.5
fft	-2.1	2.2	-4.5	-5.4	-6.7	-1.4	4.9	-5.8	-6.0	-7.6	-2.3	2.7	-3.7	-4.4	-5.6	-3.5	2.8	-1.2	-2.5	-3.7
pcm	-0.3	7.0	0.9	-1.6	-3.6	2.2	10.1	1.0	-0.4	-1.9	-1.0	5.3	-0.6	-1.7	-3.1	-0.8	4.8	0.9	-0.5	-1.8

majority consume by the data recording data structure.

VII. CONCLUSION AND FUTURE WORK

We demonstrated that existing utilization based smartphone CPU power models are ill-suited for modern multicore smartphones. To address the problem, we developed an idle-state-aware CPU power model for accurate CPU power modeling in multicore smartphones. We designed and implemented a prototype system to apply our new CPU power modeling approach, and evaluated it using a diverse set of benchmarks and real applications. Evaluation results show that our approach significantly outperforms the exiting methods in terms of estimation accuracy and stability.

ACKNOWLEDGEMENT

We thank the anonymous reviewers for their tremendously valuable feedbacks. This work was supported in part by NSF Award #1566375.

REFERENCES

[1] Antutu Battery Saver. https://play.google.com/store/apps/details?id=com.antutu.powersaver&feature=searchreult#?t=W251bGwsMSwxLDEsImNvbS5hbnR1dHUucG93ZXJz YXZlciJd.

[2] Monsoon Power Monitor. http://www.msoon.com/LabEquipment/PowerMonitor.

[3] The Dolphin Browser. https://play.google.com/store/apps/details?id=mobi.mgeek.TunnyBrowser.

[4] ARM. ARM Architecture Reference Manual - ARMv7-A and ARMv7-R edition, ARM DDI0406C.b.

[5] F. Bellosa. The Benefits of Event-driven Energy Accounting in Power-sensitive Systems. In *ACM SIGOPS European Workshop*, 2000.

[6] R. Bertran, M. Gonzalez, X. Martorell, N. Navarro, and E. Ayguade. Decomposable and Responsive Power Models for Multicore Processors using Performance Counters. In *ICS*, 2010.

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

[8] A. Carroll and G. Heiser. The System Hacker's Guide to the Galaxy Energy Usage in a Modern Smartphone. In *APSys*, 2013.

[9] M. Colmant, M. Kurpicz, P. Felber, L. Huertas, R. Rouvoy, and A. Sobe. Process-level power estimation in vm-based systems. In *EuroSys*, 2015.

[10] M. Dong and L. Zhong. Self-Constructive High-Rate System Energy Modeling for Battery-Powered Mobile Systems. In *MobiSys*, 2011.

[11] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown. MiBench: A free, commercially representative embedded benchmark suite. In *IEEE 4th Annual Workshop on Workload Characterization*, 2001.

[12] Hewlett-Packard and Intel and Microsoft and Phoenix Technologies and Toshiba. Advanced Configuration and Power Interface Specification, revision 5.0. 2011.

[13] C. Isci and M. Martonosi. Phase Characterization for Power: Evaluating Control-flow-based and Event-counter-based Techniques. In *HPCA*, 2006.

[14] W. Jung, C. Kang, C. Yoon, D. Kim, and H. Cha. DevScope: A Nonintrusive and Online Power Analysis Tool for Smartphone Hardware Components. In *CODES+ISSS*, 2012.

[15] S. Kim, H. Kim, J. Kim, J. Lee, and E. Seo. Empirical Analysis of Power Management Schemes for Multi-core Smartphones. In *ICUIMC*, 2013.

[16] R. Mittalz, A. Kansaly, and R. Chandray. Empowering Developers to Estimate App Energy Consumption. In *MobiCom*, 2012.

[17] K. Shen, Arrvindh, Shriraman, S. Dwarkadas, X. Zhang, and Z. Chen. Power Containers: An OS Facility for Fine-Grained Power and Energy Management on Multicore Servers. In *ASPLOS*, 2013.

[18] A. Shye, B. Scholbrock, and G. Memik. Into the Wild: Studying Real User Activity Patterns to Guide Power Optimizations for Mobile Architectures. In *MICRO*, 2009.

[19] D. C. Snowdon, E. L. Sueur, S. M. Petters, and G. Heiser. Koala: A Platform for OS-Level Power Management. In *EuroSys*, 2009.

[20] F. Xu, Y. Liu, Q. Li, and Y. Zhang. V-edge: Fast Self-constructive Power Modeling of Smartphones Based on Battery Voltage Dynamics. In *USENIX NSDI*, 2013.

[21] C. Yoon, D. Kim, W. Jung, C. Kang, and H. Cha. AppScope: Application Energy Metering Framework for Android Smartphone Using Kernel Activity Monitoring. In *USENIX ATC*, 2012.

[22] 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 *CODES+ISSS*, 2010.