# Accurate CPU Power Modeling for Multicore Smartphones

Yifan Zhang[1,2], Yunxin Liu[1], Li Zhuang[1], Xuanzhe Liu[1,3], Feng Zhao[1], and Qun Li[2]

[1]Microsoft Research, [2]College of William and Mary, [3]Peking University

## ABSTRACT

CPU is a major source of power consumption in smartphones. Power modeling is a key technology to understand CPU power consumption and also an important tool for power management on smartphones. However, we have found that existing CPU power models on 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 root cause is that those models estimate the power consumption of a CPU based on only frequency and utilization of the CPU, but do not consider CPU idle power states. However, we have found that CPU idle power states play a critical role in power consumption of modern multicore CPUs. Therefore, we have developed a new approach for CPU power modeling, which takes CPU idle power states into consideration, and thus can significantly improve the power estimation accuracy and stability for multicore smartphones. We present the detailed design of our power modeling approach and a prototype implementation on commercial multicore smartphones. Evaluation results show that our approach consistently achieves a high average accuracy of 98% for various benchmarks, and 96% for real applications, which significantly outperforms the existing approaches.

## 1. INTRODUCTION

Power consumption has been a paramount concern in battery-powered mobile devices such as smartphones, and CPU is a major source of power consumption in smartphones [10]. As multicore smartphones become increasingly popular, CPU power consumption becomes a more significant component in the smartphone power consumption portfolio. For example, 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 [11]. According to our measurements, the CPU power consumption of the Google Nexus series smartphones has increased significantly in the last three generations: the CPU power consumption of a Google Nexus 4 smartphone (quad-core, the 4th Nexus generation) could reach 4,065 mW, which is 2.03 times of the maximum CPU power of a Galaxy Nexus smartphone (dual-core, the 3rd Nexus generation), and is 4.51 times of that of a Nexus S smartphone (single-core, the 2nd Nexus generation). 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 power consumption of smartphone CPU. Proper and accurate power models of smartphone components benefit both users and developers. 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 consequently optimize, the energy consumption of their smartphone applications [18]. Because of its importance, power modeling has been attracting an increasing amount of research effort [12, 16, 23–26]. In this paper, we in particular study how to build accurate models for CPU power consumption in multicore smartphones.

Existing CPU power modeling approaches for smartphones assume CPU operating frequency and CPU utilization are the only major factors that impact CPU power consumption [24–26]. However, we find that this assumption does not hold with multicore CPUs in modern smartphones. Even under the same frequency and CPU utilization, two workloads with different CPU usage patterns (for example, shown in Figure 1) could consume significantly different amounts of energy. Our experiments show that the difference can be as large as 50% in a quad-core Google Nexus 4 smartphone (§3). Therefore, existing smartphone CPU power models are ill-suited for multicore smartphones. We will later show that the existing CPU power models give an estimation error as high as 34% on modern multicore smartphones (§6). Moreover, the estimation accuracy of existing models is also notably instable: the same CPU power model could generate an estimation variation larger than 30% for the different types of workloads (§6). The root cause of the estimation inaccuracy and instability come

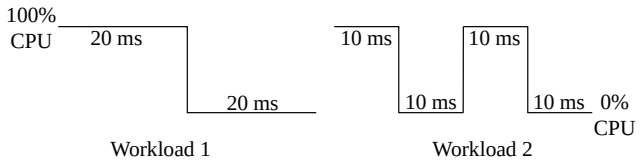



**Figure 1: Two periodic workloads with the same CPU utilization (50%) but different CPU usage patterns.**

from multiple newly introduced CPU *idle power states*, which consume markedly different amounts of power in multicore CPUs. Workloads with different CPU usage patterns cause CPU to enter different idle power states during the computation, which in turn leads to different amount of CPU power consumption. Since existing CPU power modeling methods do not take into account the impacts of CPU idle power states, they exhibit high estimation errors and instability for multicore smartphones in practice.

We have carefully analyzed the impacts of idle power states on CPU power consumption, and developed a new CPU power-modeling method that treats CPU idle power states as a new major factor of CPU power modeling (§4). As a result, the new modeling method is able to significantly improve power estimation accuracy and stability (§6). It is worth noting that CPU idle states play a less critical role in CPU power consumption on old single-core smartphones than they do on modern multicore smartphones (§3), which is probably the reason why CPU idle states are ignored in the existing CPU power models. However, to accurately model CPU power consumption in modern multicore smartphones, we need to develop a new CPU power modeling method that considers CPU idle power states.

The model building process is non-trivial indeed. As we will show in §4, 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 our proposed model, we have 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.

To the best of our knowledge, our work is the first to target accurate CPU power modeling for multicore smartphone CPUs. The main contributions of this paper are as follows.

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

The rest of the paper is organized as follows. In §2, we briefly introduce power management schemes used in contemporary smartphone CPUs. In §3, we discuss the limitations of existing smartphone CPU power modeling approaches. We develop our idle state based CPU power model in §4, present the system design and implementation in §5, and report the evaluation results in §6. We survey related work in §7. We conclude and discuss future work in §8.

## 2. BACKGROUND: SMARTPHONE CPU POWER MANAGEMENT

A smartphone CPU has different states: a CPU core can be either *online* (when the CPU core is enabled and used to process tasks), or *offline* (when the CPU core is entirely powered down and thus cannot be used to process tasks). An online CPU core can further work in either the *operating state* or an *idle state*. The operating system of a smartphone manages the states of CPU cores to reduce their total energy consumption. There are three CPU power management schemes used in modern smartphones, each of which is introduced below, with the emphasis placed on its implementation in the Android OS and the quad-core Nexus 4 smartphone.

### 2.1 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 [14]). Practically, each P-state is associated

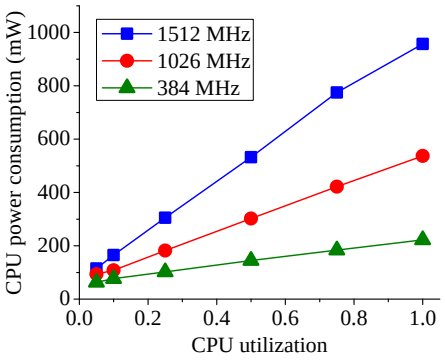



**Figure 2: Power consumption of a single Nexus 4 CPU core at different CPU utilizations.**

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.

The Nexus 4 smartphone supports 12 different CPU operating frequencies, ranging from 348 MHz to 1,512 MHz. Operating frequencies can be independently set in each CPU core. Choosing a proper frequency for an operating processor core is an important task for CPU P-state management. In Android kernel (Linux-based), a subsystem called "*CPUfreq*" specifically copes with this task by dynamically adjusting the operating frequency according to the system load [17].

Each P-state (i.e. CPU operating frequency) consumes a different amount of power. Figure 2 shows the CPU power consumption of the Nexus 4 smartphone under different CPU utilizations with only one CPU core enabled. The operating frequency is fixed at 1,512 MHz, 1,026 MHz, and 384 MHz respectively. It shows that CPU frequencies have significant impacts on CPU power consumption: the maximum power consumption of frequency 1,512 MHz is more than 3 times of that of frequency 384 MHz. In each P-state, the power consumption is determined by the CPU core utilization. Figure 2 also shows that different core utilization may have significantly different power consumptions.

## 2.2 CPU Idle Power State Management

Smartphone OS may put an online CPU core into an idle power state when there is no workload. Idle power states are called "*C-states*" in the ACPI specification [14]. CPU in different C-states have different CPU components switched to low power mode to reduce power consumption.

Table 1 shows that the Nexus 4 smartphone has four CPU idle power states: C0[1], C1, C2, and C3. A CPU core in the state C0 only disables most of the CPU clocks, while 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 in the state C1, 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.

**Table 1: CPU Idle Power States in Nexus 4.**

| Idle State | Name | Idle System Power (mW) | Latency ($\mu$S)† |
|---|---|---|---|
| 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.

We have measured the idle system power in each C-state of a Nexus 4 smartphone. The third column of Table 1 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, as we will show in §3.

In old single-core smartphones, there are less CPU idle power states. For example, the Nexus S smartphone has only one idle state, which is equivalent to the C0 state in 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 1 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 (Linux-based), a subsystem named "*CPUidle*" 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

[1]In the ACPI specification, "C0" refers to the operating state, and "C1, C2, ⋯" refer to the idle states. Here we follow the naming convention in the Nexus 4 stock kernel source code, where the state C0 refers to the shallowest CPU idle state.

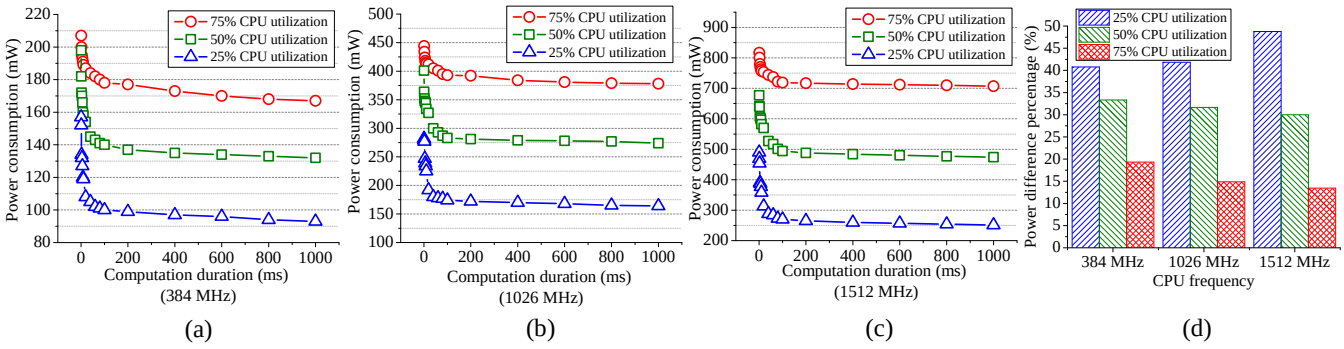



**Figure 3: Workloads running in multicore CPU with the same CPU utilization and frequency consume notably different amounts of CPU power.**

individual idle state.

## 2.3 CPU Hot-Plugging

In a multicore smartphone, the OS turns off a CPU core from the system when it has no workload for a certain period of time, and takes it back to online when the core is needed on the fly. This technique is known as CPU *hot-plugging*. While the CPU hot-plugging technique saves more power than the deepest CPU idle state, its major disadvantage is that the unplugging/replugging process requires expensive global operations, which causes a large amount of latency [19].

In Nexus 4, the stock Android system uses a user space daemon called "*mpdecision*" to manage the CPU hot-plugging process. The daemon monitors the load on CPU cores, enables and disables cores through the */sys* interface accordingly.

# 3. LIMITATIONS OF THE EXISTING SMARTPHONE CPU POWER MODELS

The existing power models [12,16,23–26] have achieved a good accuracy (e.g., more than 90%) on previous single-core smartphones such as the Nexus one and Nexus S smartphones. Those models consider only CPU utilization and operating frequency as predictors in modeling [24–26]. Usually, they use a linear CPU power model. For each CPU frequency $freq_i$, they estimate the power consumption of a CPU core as below:

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

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

However, the existing CPU power models are ill-suited for modern multicore CPUs. In particular, we have found that CPU power consumptions in two quad-core CPU smartphones with different chipsets, Nexus 4 and Samsung Galaxy S4[2], exhibit a large range of variation even when both CPU frequency and utilization are fixed. In our experiments, we first use a *workload generator program* that periodically performs continuous computation followed by an idle period (see Figure 1) in a Nexus 4 smartphone. By controlling the ratio of the idle period to the computation period, the workload generator program generates workloads with different CPU utilizations. During 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 have found 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 3(a) shows the power consumption of a CPU core[3] 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 3(b) and Figure 3(c) show the results when CPU frequency was 1,026 MHz and 1,512 MHz, respectively. They show exactly the same trend. Figure 3(d) further summarizes the difference of power consumption with the three CPU frequencies. Each value in Figure 3(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

[2]Technically the Samsung Galaxy S4 smartphone has 8 CPU cores: a quad-core ARM Cortex-A7 and a quad-core ARM Cortex-A15. However, these two quad-core CPUs cannot run concurrently, since the smartphone is using the ARM big.LITTLE task migration use model [20].

[3]The CPU power consumption is measured as the system power when the smartphone is configured in a way that CPU is the only main source of power consumption. See §6.1 for more details.

**Table 2: 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 |

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.

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. As we discussed in §2, in modern multicore smartphones like Nexus 4, the 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 the same CPU frequency/utilization, we list in Table 2 the statistics of the idle states of two workloads (W1 and W2) that were run in 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 2: 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 2 are 0s.

We have also performed the experiments in a Samsung Galaxy S4 smartphone, which is equipped with a chipset different from Nexus 4, and obtained similar observations. With the Galaxy S4 smartphone, when the CPU frequency/utilization were fixed at the top frequency/25%, the power consumption of a CPU core could exhibit up to 38% difference when we adjusted the length of continuous computation in the workload generator program. The power difference we observed in Galaxy S4 (38%) was slightly smaller than that we have seen in Nexus 4 (50%). This is probably because Nexus 4 implements deeper idle power states than Galaxy S4 does. According to our measurement, the ratio of the power consumptions of the deepest idle state over the shallowest idle state in Nexus 4 is smaller than that in Galaxy S4 (0.46 for Nexus 4, 0.51 for Galaxy S4). Since implementing deeper idle power states is a clear trend in future multicore smartphones (for more energy efficiency), we can expect the possible power difference under the same CPU frequency/utilization setting will keep growing in future smartphones, which urges the need for developing a new CPU power modeling method that considers CPU idle power states.

Some existing system power models take CPU idle states into account. For example, Koala [23] proposes to takes CPU idle states into account when estimating system power consumption. However, it only reports the results on a laptop and a server, both x86-based. Furthermore, it only considers the portion of each CPU idle state duration over the whole idle period. As we will show in §4, even when portion of each idle state duration is fixed, CPU could have more than 20% variation of power consumption. Sesame [12] also considers CPU idle states when modeling CPU power consumption, but also only for x86-based laptops. Due to the Instruction Set Architecture (ISA) difference, those power models for x86 may not be used on ARM-based devices such as smartphones. To the best of our knowledge, our work is the first comprehensive study focusing on CPU-idle-state-based power modeling approach on ARM-based smartphone CPUs. We have conducted thorough experiments to investigate how CPU idle states can affect power consumption of smartphone CPUs. We also developed a new CPU power modeling approach based on the investigation results, designed and implemented a prototype system using the new approach, and evaluated the prototype system with comprehensive experiments. We discuss more related work in §7.

Since older single-core smartphones have less CPU idle states than modern multicore smartphones and CPU idle states play a less important role in old single-core smartphone, it is reasonable that existing smartphone CPU power models ignore CPU idle states. However, to build an accurate CPU power model for multicore smartphones, we must consider CPU idle states. Next in §4 we show how we take CPU idle states into consideration and build an accurate CPU power model for multicore smartphones.

## 4. IDLE-STATE-BASED CPU POWER MODEL

We propose a new CPU power modeling approach for

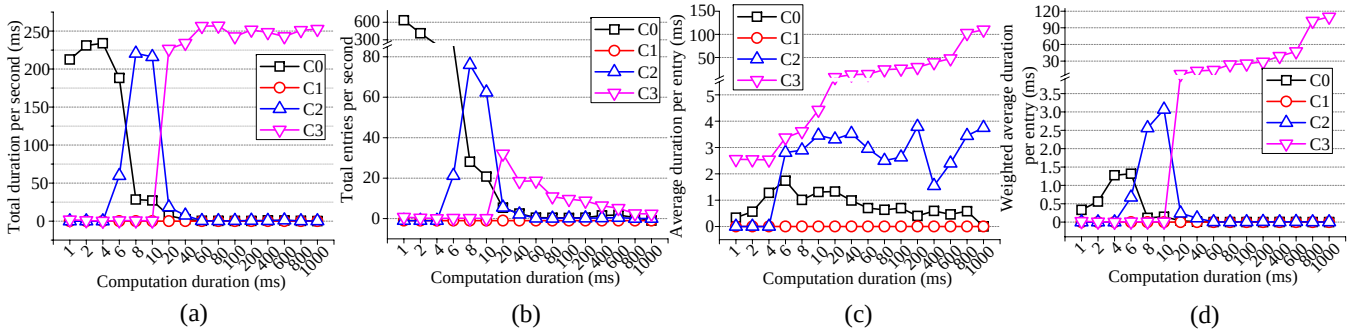



**Figure 4: Single-core power model development. Figures (a)-(d) show $T_{C_i}$, $E_{C_i}$, $ED_{C_i}$, and $WED_{C_i}$ for the four CPU idle states $C_0$ - $C_3$, respectively (with CPU frequency $f = 1,512$ MHz, utilization $U = 75\%$).**

smartphones, which considers not only CPU frequency and utilization, but also the impacts of CPU idle states. In this section, we first present the development of our power modeling for the single-core case. Then, we show how the single-core power model can be extended to the multicore case. All the experiments described in this section are performed in a Nexus 4 smartphone.

## 4.1 Power Modeling for a Single CPU Core

Similar to existing work, we use regression-based method to integrate the predictors. Our predictors include not only CPU frequency and utilization, but also idle states. To determine what statistic 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 4(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 for $C_1$ remain zero in Figure 4. The figure 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 the computation duration increased (see Figure 3(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 4(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, $ED_{C_i}$ is 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 enters state $C_i$ most of the time. For example, Figure 4(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 4(d) shows $WED_{C_i}$ in the experiment.

Finally, 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 $\beta_{C_i}$ and $\beta_U$ 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 §5).

## 4.2 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 differ-

**Table 3: CPU power with different number 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.

ent number of CPU cores, which are 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 3 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 in 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

$$P_{CPU} = P_{BL,N_c} + \sum_{i}^{N_c} P_{\Delta,core,U_i,f_i} \quad (6)$$

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.

## 5. SYSTEM DESIGN AND IMPLEMENTATION

We have designed and implemented a prototype CPU power estimation system using our idle-state-based CPU model on Android platform. Figure 5 shows an overview

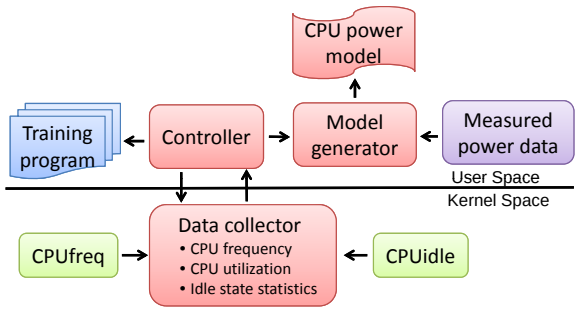



**Figure 5: System overview.**

of the system. The system contains two parts: one runs in the kernel space, and the other runs in the user space. In the kernel space, the *data collector* component collects necessary CPU usage data including the CPU frequency, CPU utilization, and CPU idle state statistics. In the 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 §6.1). 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 the kernel**. We design a data collector to work in the kernel space for lightweight and efficient data collection. A design alternative is to periodically sample CPU utilization and CPU idle states in the user space via the high-latency */proc* and */sys* filesystems. However, because our power model 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 our 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, we significantly reduce the number of user-kernel mode switching, and thus introduce 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.

To guarantee accuracy, it is straightforward to periodically sample data in the kernel, with the sampling rate set to the highest possible value of frequency changing rate. However, this method would incur unnecessary system overheads, since it requires a high sampling

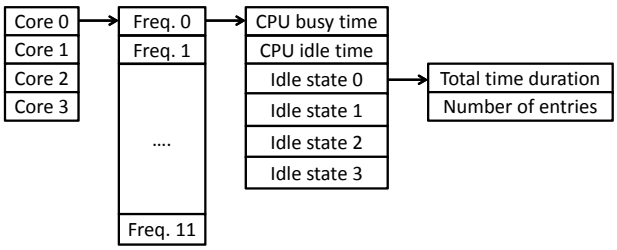



**Figure 6: Data structure used in the data collector.**

rate even when the actual CPU frequency changing rate is low. We take a different method in our implementation. 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 the subsystems. We have instrumented the subsystems so that we know when the CPU frequency or CPU idle state are changed. Each frequency change in *CPUfreq* triggers a new process of data collection for the new working frequency. For each CPU idle state change in *CPUidle*, we collect new data about the previous CPU idle state and aggregate them to the existing data. Therefore, our data collection automatically adapts to CPU frequency changes, and thus avoids unnecessary system overheads.

Figure 6 shows the data structure used in our data collector (assuming a Nexus 4 smartphone is used). We collect the CPU usage data for each CPU core and each CPU frequency separately. Each CPU core has an array of 12 CPU 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 6, 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, the data collector consumes a small fixed amount of memory, which is independent of the time duration of data collecting. Compared to recording the raw data, this approach also uses much less memory, especially when the data collecting time is long.

**Generating CPU power model**. To generate a CPU power model, we have run a set of training programs with various workloads and CPU usage patterns. We use the workload generator described in §3 to create training programs with various CPU frequencies, utilization, and various 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. 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 §4.1. Then we enable all cores, and run the training programs with an identical process on each core, to generate the multi-core power model described in §4.2. The whole model generation procedure takes about 2 hours. It is worth noting that the ground-truth CPU power consumption is obtained manually by using power meter. One could also obtain the ground-truth value by referring to the battery interface [12, 24], which allows for automated model generation. We opted to manual measurement because we wanted to reduce the possible errors introduced by using the battery interface.

**Applying CPU power model**. We have written 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 5, 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.

# 6. EVALUATION

## 6.1 Experimental Setup

We used a Nexus 4 smartphone, which has 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 [3]. Since we focus on the CPU power consumption, we disabled other hardware components as much as possible including turning off 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. Note that the measured CPU power (i.e., the ground truth value) include

**Table 4: 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). |

power consumption by CPU, memory, and flash disk. Since our training programs also include memory activities, we expect power consumption on flash disk will incur small impact on the accuracy of our CPU models. Each experiment was repeated for 5 times and we report the average results.

**Benchmarks**. We used 13 benchmarks in our evaluation. The first one is the workload generator described in §3. 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 as showed in Table 4. We ported these 12 types of computation from MiBench, which is a free and commercially representative embedded benchmark suite [13]. As shown in Table 4, 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 randomly selected from 0% to 100%, and the busy loop count randomly selected from 1 to 5 during the continuous computation periods. Depending on the computation type, the continuous computation periods ranged from 10 ms to 1000 ms, 250 ms on average.

**Real applications**. Besides the above benchmarks, we also used the following 5 applications to evaluate our CPU power model.

- *Web browsing*: we used the Dolphin Browser [6] 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 Googles 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 used the tool [27] to capture and replay the user inputs on the touch screen, so that we could operate on the map with desired 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 processes of many CPU intensive games (e.g., Angry Birds) terminate 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 MP4 video clip (30 frames/sec, 611 kbps bitrate) for 20 seconds. We configured Dolphin Player to do video decoding in software using CPU rather than the dedicated video decoding hardware.
- *Audio decoding*: we used Google Music to play a MP3 song clip (44.1 KHz sample rate, 64 kbps bitrate) for 20 seconds. The Google Music decodes audio file with software.

Please note that the goal of conducting experiments on real applications is to evaluate how our power modeling approach, which focuses on estimating power consumption of the CPU component, 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) [24].

**Utilization based CPU power models**. To compare our power model (labeled as IM) with existing CPU power models, we generated 4 utilization based CPU power models (i.e., traditional CPU power models that consider only CPU frequency and utilization) as follows. We used the same training programs as in our model generation process (§5), 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 §4.2. 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 we used in our CPU idle state based power model.

We define the accuracy of a power model as follows:

$$Accuracy = 100\% - \frac{|P_e - P_m|}{P_m}\% \qquad (7)$$

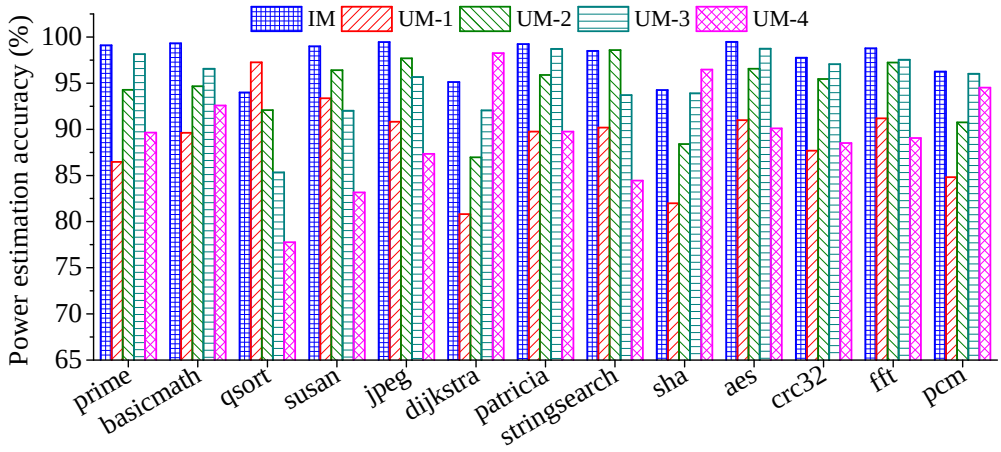



**Figure 7: Single-core model accuracy of the 13 benchmarks.**

**Table 5: Single-core model accuracy of the 13 benchmarks with different CPU utilizations.**

| Benchmark | Model Accuracy (%) | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 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 |
| **prime** | 98 | 84 | 98 | 81 | 73 | 100 | 92 | 98 | 96 | 88 | 99 | 97 | 96 | 94 | 98 |
| **basicmath** | 98 | 74 | 89 | 96 | 88 | 100 | 88 | 93 | 97 | 94 | 99 | 98 | 96 | 94 | 98 |
| **qsort** | 99 | 85 | 96 | 77 | 69 | 95 | 98 | 92 | 87 | 80 | 93 | 91 | 93 | 95 | 87 |
| **susan** | 97 | 82 | 96 | 74 | 66 | 100 | 93 | 99 | 96 | 86 | 97 | 95 | 93 | 91 | 100 |
| **jpeg** | 100 | 82 | 96 | 93 | 84 | 99 | 91 | 97 | 98 | 90 | 99 | 95 | 94 | 92 | 99 |
| **dijkstra** | 94 | 70 | 85 | 97 | 93 | 94 | 84 | 87 | 93 | 98 | 98 | 92 | 89 | 89 | 98 |
| **patricia** | 99 | 87 | 100 | 90 | 81 | 99 | 92 | 97 | 99 | 89 | 99 | 97 | 96 | 94 | 98 |
| **stringsearch** | 96 | 79 | 91 | 81 | 71 | 98 | 94 | 99 | 94 | 84 | 95 | 92 | 90 | 88 | 97 |
| **sha** | 94 | 68 | 84 | 98 | 93 | 95 | 85 | 89 | 95 | 96 | 96 | 91 | 89 | 87 | 96 |
| **aes** | 94 | 82 | 94 | 97 | 88 | 98 | 90 | 95 | 100 | 92 | 100 | 96 | 94 | 93 | 99 |
| **crc32** | 99 | 87 | 98 | 85 | 77 | 99 | 91 | 97 | 98 | 89 | 96 | 93 | 92 | 90 | 99 |
| **fft** | 100 | 88 | 99 | 92 | 84 | 99 | 92 | 97 | 99 | 90 | 98 | 99 | 98 | 96 | 96 |
| **pcm** | 92 | 75 | 90 | 97 | 88 | 98 | 89 | 94 | 99 | 92 | 99 | 94 | 93 | 91 | 99 |

where $P_e$ is the power estimated by the power model, and $P_m$ is the power measured using the power meter.

## 6.2 Experimental Results

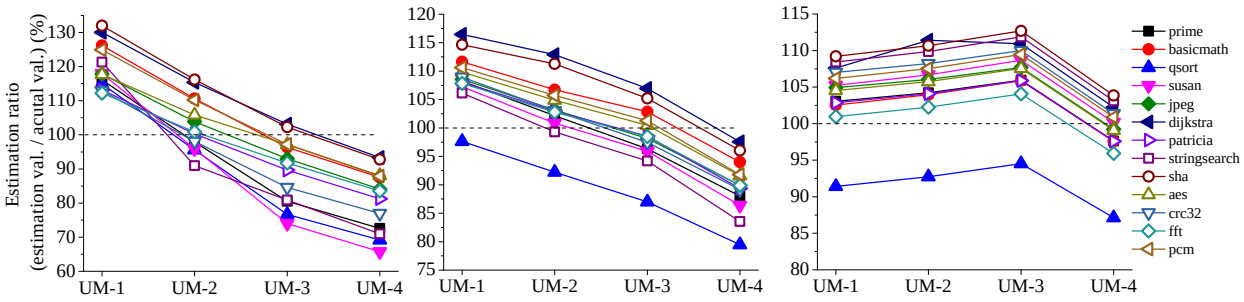
We evaluated our prototype system from two aspects: accuracy of our CPU power models and system overheads.

### 6.2.1 *Accuracy of single-core models*

We first evaluated the model accuracy when only a single CPU core was used.

**Benchmark experiments results**. Figure 7 shows the results of the 13 benchmarks with randomly decided CPU utilization in each computation period. On average our model achieved a high accuracy of 98%, with a small variation ranging from 94% to 100% for different benchmarks. 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 causes different patterns of CPU idle state entries. The utilization based models were unable to capture the effect of these CPU idles state changes, which are important dynamics affecting CPU power consumption. On the contrary, our model can well cope with this dynamic usage pattern, since it is designed with the impacts of idle states in mind.*

The accuracy of the existing utilization based models are also subject to CPU utilization. Table 5 shows 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 was only 66% in the *susan* benchmark, and the accuracy of model UM-1 was only 68% in the *sha* benchmark. This is *because when CPU utilization was*




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

**Figure 8: Estimation ratios of the four utilization based models (single-core).**

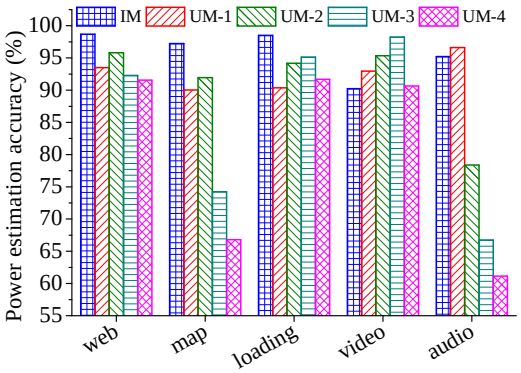



**Figure 9: Single-core model accuracy with real apps.**

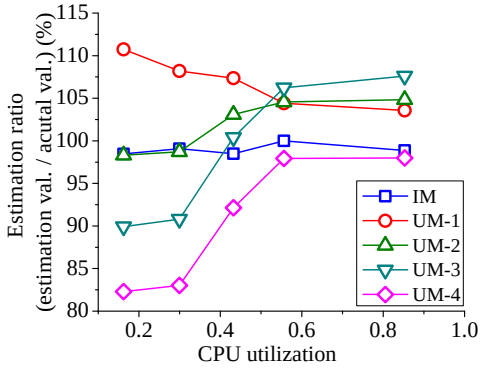


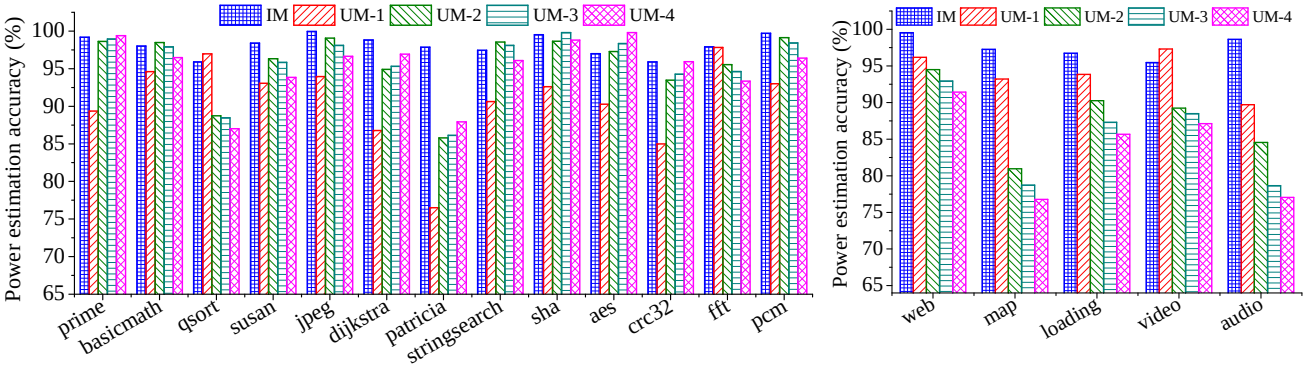

**Figure 10: Estimation ratio vs. utilization.**

*low, there were more idle time, which in turned led to more dynamic pattern of idle state entries.* Our model achieved consistently high accuracy, with the lowest accuracy of 92% (in the *pcm* benchmark with 30% CPU utilization).

To further study how different types of workloads and different CPU utilizations could affect the existing utilization based models, we show in Figure 8 the *power estimation ratio* of UM-1 to UM-4 when testing the 13 benchmarks with CPU utilization fixed at 30%, 60%, and 90%, respectively. The *power estimation ratio* is the percentage of the estimated power value over the measured (i.e., ground truth) power value. Thus, the closer to 100%, the better is the power estimation ratio. Figure 8 shows that for a given benchmark at fixed CPU utilization, it is possible to find a CPU utilization based model to achieve a high estimation accuracy. However, that model would have a much lower model estimation accuracy in some other benchmarks and other CPU utilization levels. For example, when the CPU utilization is 60% (Figure 8(b)), UM-2 achieves almost 100% estimation ratio for benchmark *stringsearch*, but UM-2 would estimate about 15% more than the ground truth value if it is used for benchmark *dijkstra*. Another example is that UM-3 achieves an estimation ratio slightly more than 95% for benchmark *susan* when CPU utilization is 60% (Figure 8(b)). However, the estimation ratio for the same benchmark drops below 75% when CPU utilization is 30% (Figure 8(a)). In sum, *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. On the contrary, 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 all the benchmarks and different CPU utilizations.*

**Real application experiments results**. The similar observations can be found in the real application experiments as well. Figure 9 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 ratio and CPU utilization for the real applica-




(a) the benchmark tests (b) the real app tests

**Figure 11: Multicore model accuracy**

**Table 6: Multicore model accuracy of the 13 benchmarks with different CPU utilizations.**

| Benchmark | Model Accuracy (%) | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 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 |
| **prime** | 100 | 93 | 90 | 91 | 89 | 99 | 91 | 98 | 99 | 100 | 98 | 98 | 98 | 97 | 96 |
| **basicmath** | 98 | 90 | 98 | 98 | 96 | 97 | 96 | 96 | 96 | 96 | 98 | 96 | 100 | 99 | 98 |
| **qsort** | 97 | 97 | 80 | 81 | 79 | 94 | 97 | 90 | 89 | 88 | 95 | 96 | 92 | 91 | 90 |
| **susan** | 99 | 97 | 86 | 87 | 85 | 98 | 93 | 99 | 98 | 96 | 99 | 95 | 99 | 100 | 98 |
| **jpeg** | 100 | 94 | 94 | 94 | 92 | 99 | 94 | 99 | 99 | 98 | 98 | 93 | 97 | 98 | 100 |
| **dijkstra** | 98 | 81 | 92 | 93 | 94 | 100 | 90 | 97 | 97 | 99 | 99 | 93 | 97 | 98 | 99 |
| **patricia** | 98 | 79 | 91 | 92 | 93 | 98 | 87 | 94 | 95 | 96 | 97 | 88 | 92 | 93 | 95 |
| **stringsearch** | 99 | 94 | 89 | 90 | 88 | 99 | 93 | 99 | 97 | 96 | 98 | 94 | 98 | 100 | 99 |
| **sha** | 98 | 87 | 98 | 99 | 100 | 99 | 93 | 99 | 100 | 99 | 99 | 93 | 97 | 98 | 99 |
| **aes** | 98 | 89 | 100 | 100 | 98 | 99 | 93 | 99 | 99 | 97 | 98 | 92 | 96 | 98 | 99 |
| **crc32** | 96 | 88 | 96 | 97 | 95 | 94 | 84 | 93 | 93 | 95 | 93 | 84 | 89 | 90 | 92 |
| **fft** | 99 | 95 | 94 | 94 | 92 | 98 | 97 | 96 | 96 | 94 | 97 | 97 | 99 | 98 | 96 |
| **pcm** | 98 | 90 | 99 | 100 | 98 | 99 | 95 | 99 | 98 | 97 | 99 | 95 | 99 | 100 | 98 |

tion experiments. Figure 10 shows the estimation ratios of all the models when the CPU utilization was different in the Web browsing application. 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 gave a lower model accuracy, which was also observed in other applications. 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.

### 6.2.2 *Accuracy of multicore models*

Figure 11 shows the multicore model accuracy results when we ran the benchmarks and applications using all the four CPU cores. Figure 11 (a) and (b) show the results of the benchmark experiments and the real application experiments respectively. Similar to the single-core case, our model achieved a higher average accuracy, and a much smaller range of accuracy variation than the existing utilization based models. 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. Table 6 shows more results when the CPU utilization was fixed at 30%, 60%, and 90%.

Compared to the single-core case, the accuracies 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. However, we still have the same observations as in the single-core case: 1) our model has a consistently high accuracy in all the benchmarks and applications, and significantly outperforms the CPU utilization based models; 2) the CPU

utilization based models have a large range of model accuracies in different benchmarks and application, and give a 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. On the contrary, 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.*

#### 6.2.3 System overheads

From the 312 training programs (§5), 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 majority consume by the data recording data structure.

## 7. 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 [12, 16, 22, 24, 26] 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 [12,23]. Specifically, Koala [23] 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 have showed before, 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 [12] 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-based CPU power modeling approach based on the investigation results.

In [28], we propose to incorporate idle states into CPU power modeling for multicore smartphones, and present preliminary results. In this paper, we did more systematic study, presented the detailed prototype system design and implementation, and conducted comprehensive evaluations with various benchmarks and real applications.

**CPU hardware events based approaches**. Another way of performing CPU power modeling is to model the relationship between CPU power and CPU hardware events [8, 9, 15, 21]. For example, Power Containers [21] 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 [7], 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 CPU hardware events based modeling.

## 8. CONCLUSION AND FUTURE WORK

In this paper we demonstrated that existing CPU utilization based power models are ill-suited for modern multicore smartphones. Without considering the impacts of CPU idle states, existing power models give high errors in multicore smartphones. To address the limitations of existing power models, we developed an *idle-state-based* CPU power model for accurate CPU power modeling in multicore smartphones.

We have designed and implemented a prototype system of our new CPU power modeling approach using the quad-core CPU Nexus 4 smartphones, and also conducted comprehensive evaluations using a diverse set of benchmarks and real applications. Experimental results show that our CPU power model achieves a high model accuracy, which significantly outperforms the existing CPU utilization based power models, with negligible

system overheads.

Although our current implementation and evaluation are performed on the Nexus 4 smartphone only, our approach of modeling CPU power consumption can be applied to many other smartphones. The quad-core CPU used by Nexus 4, Qualcomm Snapdragon S4 Pro, has been widely used on many types of mainstream multicore smartphones, such HTC Droid DNA, LG Optimus G, Sony Xperia Z, and Samsung Galaxy S4 AT&T version. Our CPU power model should work for these smartphones.

We plan to further study the performance of our power modeling approach using more multicore smartphone chipsets, such as Samsung Exynos [5] and NVIDIA TEGRA 4 [4]. Besides homogeneous multicore CPUs, we are also interested in examining heterogeneous multicore architectures, such as the ARM big.LITTLE architecture [2] used in Samsung Galaxy S4 smartphones. Although these chipsets all share the same ARM architecture and we expect that our method will work on them, the different implementations regarding how different CPU cores cooperate demand more exploration to study how well our method may work. For example, in the Qualcomm Snapdragon S4 chipset (i.e., the one used in Nexus 4 smartphones), each running cores can be configured independently, while in the Samsung Exynos chipset, all the running cores must share the same configurations (e.g., operating frequency, max/min frequencies, *CPUfreq* governors). In the ARM big.LITTLE architecture, the "big" cores and the "little" cores can also cooperate in different ways: in the big.LITTLE task migration use model [20], the two types of cores take responsibility for different ranges of frequencies and cannot work concurrently, while in the big.LITTLE MP use model [20], the two types of cores can work together when more computation power is needed. Furthermore, as smartphone GPUs become increasingly powerful and consume a significantly high power [11], we also plan to study how to achieve accurate and lightweight GPU power modeling for smartphones.

# 9. REFERENCES


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

[2] ARM bit.LITTLE processing. `http://www.arm.com/products/processors/technologies/bigLITTLEprocessing.php`.

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

[4] NVIDIA TEGRA 4 processor. `http://www.nvidia.com/object/tegra-4-processor.html`.

[5] Samsung Exynos processor. `http://www.samsung.com/global/business/semiconductor/minisite/Exynos`.

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

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

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

[9] 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.

[10] A. Carroll and G. Heiser. An Analysis of Power Consumption in a Smartphone. 2010.

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

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

[13] 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.

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

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

[16] 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.

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

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

[19] S. Panneerselvam and M. M. Swift. Chameleon: Operating System Support for Dynamic Processors. In *ASPLOS*, 2012.

[20] Peter Greenhalgh. ARM White Paper: big.LITTLE Processing with ARM Cortex-A15 & Cortex-A7. 2011.

[21] 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.

[22] 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.

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

[24] 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.

[25] 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.

[26] 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.

[27] Y. Zhang, C. C. Tan, and Q. Li. CacheKeeper: A System-wide Web Caching Service for Smartphones. In *Ubicomp*, 2013.

[28] Y. Zhang, X. Wang, X. Liu, Y. Liu, L. Zhuang, and F. Zhao. Towards Better CPU Power Management on Multicore Smartphones. In *Hotpower*, 2013.