Towards Better CPU Power Management on Multicore Smartphones

Yifan Zhang[1,2], Xudong Wang[3,1], Xuanzhe Liu[3,1], Yunxin Liu[1], Li Zhuang[1], Feng Zhao[1]

[1]*Microsoft Research Asia, Beijing, China* [2]*College of William and Mary, Williamsburg, VA, USA*

[3]*Peking University, Beijing, China*

ABSTRACT

Although multicore smartphones have become increasingly mainstream, it is unclear whether and how smartphone applications can utilize multicore CPUs to improve performance. In this paper we study the performance of mobile applications using multicore CPUs, in terms of power and computation cost. Using Web browsing as an example, our preliminary measurement results show that even large applications like Web browsers with multi-threading acceleration cannot fully utilize the multicore CPUs. Furthermore, we find that the existing CPU power models on smartphones are ill-suited for modern multicore CPUs. We develop a new CPU power model with a high accuracy, 95.6% on average. Our work helps to better understand the performance of multicore smartphones and paves the way towards better CPU power management on multicore smartphones.

Categories and Subject Descriptors

C.4 [Performance of Systems]: Modeling Techniques

General Terms

Experimentation, Measurement, Performance

Keywords

Power Management, Multicore, Power Modeling, Smartphone

1. INTRODUCTION

Multicore smartphones are increasingly common. For example, Quad-core CPU is already the standard configuration of mainstream smartphones, such as Nexus 4, Samsung Galaxy S3, HTC One, and many others. Samsung Galaxy S4 even has an 8-core CPU. With more 8-core chipsets such as MediaTek MT6592 to be released, more 8-core CPU smartphones are coming.

Despite the proliferation of multicore smartphones in the market, the performance and end-user experience of multicore smartphones remains unclear. The purposes of the multicore design are to reduce the power consumption (a low-frequency CPU core consumes less power than a high-frequency one) and to improve the performance (e.g., applications may use multiple CPU cores simultaneously to speed up). However, in practice there are many customer complaints about the performance of multicore smartphones, particularly the high power consumption, and debate about the value of multicore CPU on smartphones [1].

In this paper, we present a study on the performance of mobile applications on multicore smartphones, on both power and computation cost. From such a study, we can understand how existing applications utilize multicore CPU and improve applications to better leverage the capability of multicores and the system to better manage the power. Towards this end, we have studied the performance of Web browsing on multicore smartphones, because it is one of the most frequently used application on smartphones; the browsers support multi-threading acceleration and thus are more multicore friendly than other applications.

Based on our preliminary measurement results using the Chrome browser (version 19), we find it cannot fully utilize the power of multicore CPU, although the browser supports multi-threading accretion and is designed with parallelization in mind [2, 3, 4]. Comparing to the single CPU core case, using more CPU cores only provides very small performance gain, both in total energy cost and page load time.

To scale our study to more applications, we need an automatic approach to measuring the power consumption which is currently done using an external power meter and thus very time consuming. Power-modeling is a promising approach to automatic measurement. However, we find the exiting CPU power-modeling methods, which estimate the power consumption of a CPU based on the frequency and utilization of the CPU [5, 6, 7], give very high errors on modern multicore smartphones. For example, two workloads with the same CPU utilization level but different CPU usage patterns may consume a significantly different amount of energy, with the difference up to 50%. To address this problem, we propose a new CPU power-modeling method for multicore smartphones. Our method considers the impacts of CPU *idle states* and is able to achieve a very high accuracy, 95.6% on average.

The main contributions of this paper are twofold:

- Using Web browsing as an example, our preliminary study shows that even large smartphone applications like Web browsers with multi-threading acceleration cannot fully utilize multicore CPUs. This calls for a deep study on application performance on multicore smartphones.

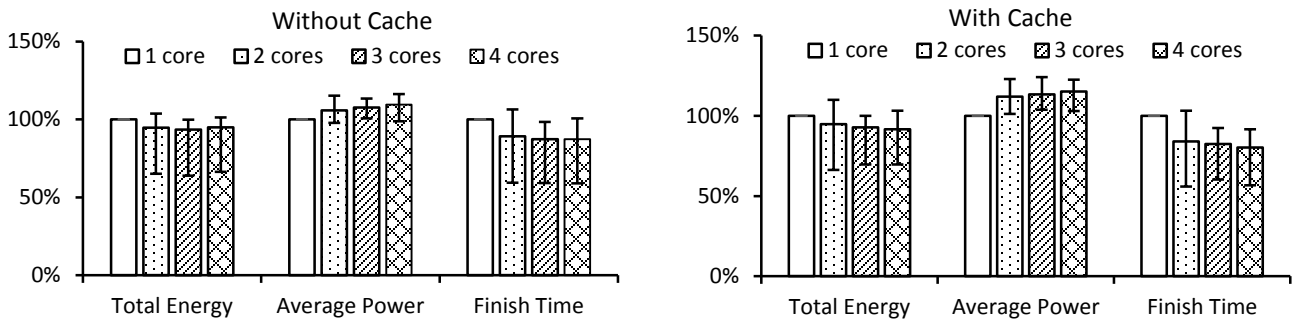

Figure 1: Energy/power consumption and finish time in loading the ten webpages with different numbers of CPU cores used.

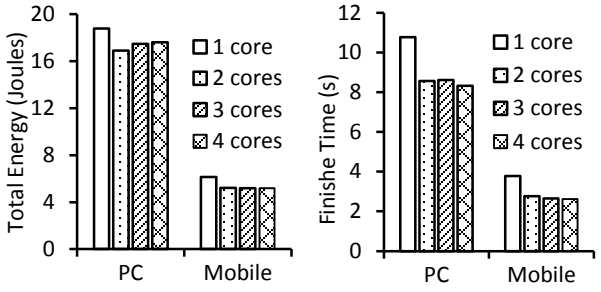

Figure 2: Total energy and finish time of loading the homepage of QQ, both PC version and mobile version.

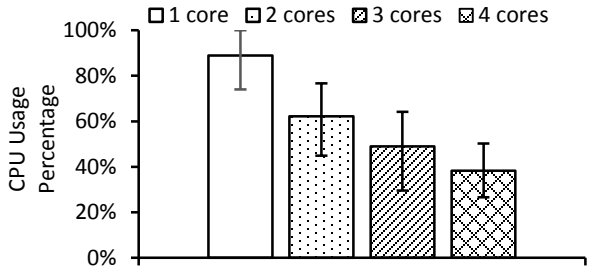

Figure 3: Average CPU usage of loading the ten webpages when different numbers of cores are used.

- We show that the existing CPU power models on smartphones cannot be used to estimate the power consumption of multicore CPUs. We propose a new CPU idle-state-based power model which is able to achieve a high accuracy, as an important step stone towards a large scale study of application power performance and better CPU power management on multicore smartphones.

2. PERFORMANCE OF WEB BROWSING

2.1 Experimental Setup

We conducted experiments to measure the performance of Web browsing on a Nexus 4 smartphone running Android 4.2, with a 1.5GHz Quad-core Snapdragon S4 Pro processor, 2GB RAM and 8GB internal storage. The power consumption was measured by a Monsoon Power Monitor [8], with minimal background processes and the lowest screen backlight level. We used Ftrace, a Linux kernel tracer [9], to log the CPU scheduling information in the kernel to precisely calculate the CPU utilization of every process. We developed a tool that controls the ON/OFF states of each CPU core and thus is able to measure application performance using arbitrary numbers of CPU cores.

We used the popular Google Chrome browser (version 19) and have measured the total energy cost, the average power and the finish time (i.e., the page load time) in loading a webpage. We measured ten popular websites, including a search query with two search engines (Google and Bing), and homepages of three Web portals (Sina, QQ and 163), three e-commerce websites (Amazon, eBay and Taobao), Wikipedia and YouTube. Each experiment was repeated for 5 times and we report the average results.

2.2 Measurement Results

Figure 1 shows the energy/power cost and the finish time in loading ten webpages. We compare the average performance of loading those webpages when different numbers of the CPU cores are enabled, using the performance on a single core as the baseline (i.e., the 1-core bars are always 100%). The figure on the left shows the results of loading webpages without any web caching, and the figure on the right shows the results when webpages have been visited and thus cached.

It shows that the overall performance is improved when two cores are enabled. For example, in the "Without Cache" case, the finish time decreases by 10.9% and the total energy cost is reduced by 5.2%, although the average power increases by 5.7%. This demonstrates that the Chrome browser is able to benefit from using two cores, both in the total energy cost and in the finish time. However, the performance improvement is relatively small, far from being doubled. Furthermore, when more cores are enabled, the extra performance gains are almost negligible. From 2 cores to 3 cores, the extra savings of total energy and finish time are only 1.3% and 1.8%. The performance of the case of 4 cores is even slightly worse than the one of the case of 2 cores. We see similar results in the "With Cache" figure. Comparing to the "Without Cache" experiments, the saving of finish time in the 2-core case is slightly higher, 12.0%, due to the less time spent on network data fetching. However, the reduction of total energy is the same, 5.2%, due to the higher average power.

To measure the performance of more complex webpages, we repeated the experiments using the PC version of the above webpages. We observed the similar results as the mobile version webpages. Figure 2 shows the results of QQ, a popular Web portal in China, both the mobile version and the PC version. We can see that

Table 1: CPU idle power states on Nexus 4.

State	Name	Idle Power (mW)
C0	Wait for interrupt	433
C1	Retention	390
C2	Power collapse standalone	330
C3	Power collapse	200
Without entering idle states		1,060

compared to loading the mobile version webpage, loading the PC version webpage takes a much longer time and costs much more energy, because there are much more data to be fetched and processed. However, we see the same performance trend when different numbers of CPU cores are used: using two cores can help improve the performance slightly but using more cores cannot help further. In fact, when 3 or 4 cores are used, the energy cost is even higher than the one of using only 2 cores. These results indicate that the Chrome browser cannot leverage multicore to boost its performance, even for complex webpages which require more computations.

By analyzing the traces of Ftrace, we further study the CPU usage in loading the ten webpages, when different numbers of CPU cores are used. In loading the webpages, the Chrome browser created 15-21 threads, each for a task or worker such as parsing and rendering the webpage or handling I/O and cache. Figure 3 shows the average results of the ten webpages in terms of the total CPU utilization ratio. That is, when multiple CPU cores are used, we calculate the average usage percentage of all the cores. We can see that the total CPU usage percentage significantly decreases when more cores are used. Clearly, the Chrome browser fails to fully utilize the power of multicore CPU.

Summary of findings. From the above preliminary measurement results, we can see that despite the Chrome browser employs multiple threads to parallelize and accelerate the loading of webpages, it still cannot fully leverage the power of multicore CPU to reduce the page load time or the total energy cost. We guess that this is probably due to the tight coupling among the threads of the Chrome browser. There are on-going research efforts on parallelizing browsers [2, 3, 4]. We plan to investigate more on this.

3. POWER MODELING OF MULTICORE CPU

In this section we first give the background on CPU idle states. Then we show that the existing CPU power models do not work well on multicore smartphones and propose a new power model for accurate CPU power modeling.

3.1 Background: CPU Idle States

An online CPU works in either the *operating state* in which all the CPU components are powered up and there are tasks to process, or an *idle state* when there is no workload in the CPU and thus some parts of the CPU is put into low-power mode. The operating state and the idle states are also called "C-states" in the ACPI specification [12].

As shown in Table 1, on Nexus 4, there are four CPU idle power states, C0-C3, which are achieved by disabling different CPU components. For instance, in the state C0 only the CPU clocks are disabled but in the state C3 all the CPU caches are also flushed and disabled. By enabling each of the idle states separately, we have measured the idle system power of a Nexus 4 smartphone when the phone enters each idle state. As a comparison, we also measured

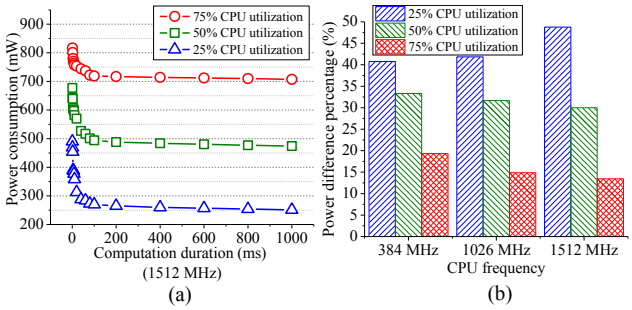

Figure 4: Workloads with the same CPU utilization and frequency may have very different power consumptions.

the case when all the CPU idle states are disabled. We can see that the idle states have much lower power than the operating state and the powers of different idle states are also very different, which affect the accuracy of the existing power models, as we will show next.

3.2 Limitation of Existing CPU Power Models

Most of the existing work on building CPU power models on smartphones consider only CPU utilization ratio and operating frequency as the predicators in the modeling [5, 6, 7]. However, we found that CPU power consumption in Nexus 4 exhibited a large variation even when both CPU frequency and utilization ratio were fixed. In our experiment, we wrote a workload generator program that periodically performed continuous computation followed by an idle period. By controlling the ratio of idle period with respect to the continuous computation duration, the program could generate workloads with different CPU utilization ratios. We found that by adjusting time duration of the continuous computation, power consumption of a single CPU core could exhibit a large range of variation even when the CPU frequency and utilization ratio were fixed.

For example, Figure 4(a) show the power consumption of a single CPU core when the operating frequency was fixed at 1512 MHz. We can see that, with a fixed CPU utilization ratio, the power consumption of the CPU core dropped while the duration of the continuous computation increased. Figure 4(b) summarizes the difference of power consumption with two more CPU frequencies (384 MHz and 1026 MHz). Each value in the figure is calculated as the percentage of the difference between the max and min powers over the max value for each frequency/utilization configuration. We can see that when frequency/utilization ratio were fixed at 1512 MHz/25%, the power difference could reach as high as 50%.

The reason is because modern multicore CPUs like the one of Nexus 4 smartphones have multiple *idle states* which have very different power consumptions. When utilization ratio was fixed, prolonging the duration of the continuous computation caused the corresponding idle period to increase accordingly. Longer idle period allowed the OS to put the CPU core into deeper idle states more frequently, which in turn lowered the CPU power consumption.

Koala [10] proposes a CPU power model that takes CPU idle states into account. However, it only considers the portion of each CPU idle state duration over the whole idle period. As we will showed later, even when portion of each idle state duration is fixed, CPU could have more than 20% variation of power consumption. Sesame [11] also considers CPU idle states when modeling CPU power consumption. But it does not provide description about how this

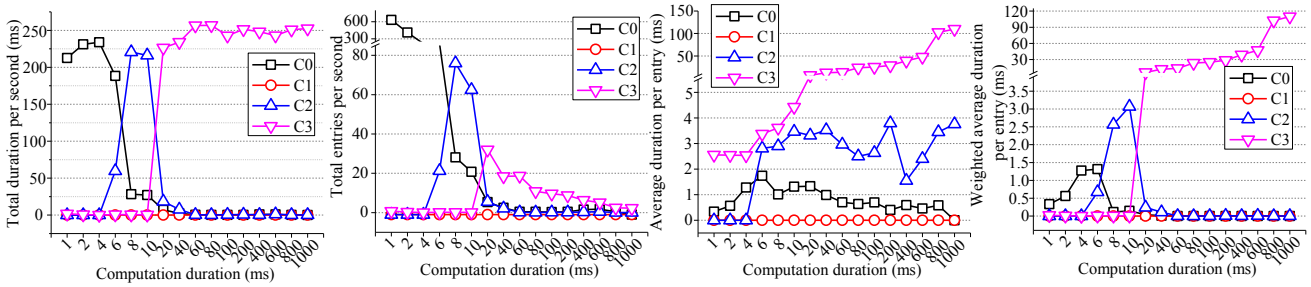

Figure 5: Single-core power model development. Figures (a)-(d) show T_{C_i}, E_{C_i}, ED_{C_i}, and WED_{C_i} for the four CPU idle states C_0 - C_3, respectively (with CPU frequency f = 1512 MHz, utilization ratio U = 75%).

particular information is used in the modeling process. Moreover, Sesame considers idle states only in its laptop power models, but not in the power models for smartphones.

3.3 Idle-State-Based CPU Power Model

We propose a new CPU power model for smartphones which not only considers CPU frequency and utilization ratio, but also takes into account the impacts of CPU idle states. In the following, we first present the development of our power modeling for the single core case. Then, we show how the single-core power modeling can be adjusted to the multi-core case.

3.3.1 Power Modeling for a Single CPU Core

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

$$T_{idle} = \sum_i T_{C_i} \quad (1)$$

Figure 5(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=1512 MHz, U=75%). Note that since the stock Nexus 4 kernel does not enable the idle state C_1, statistics for C_1 remain zero in Figure 5. We can see 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 kept decreasing as shown in Figure 4(a). In fact, in our experiment, the power difference could reach 24% for the same T_{C_i} (i=0,1,2,3) (when f=1512 MHz, U=25%).

Figure 5(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, which explains our previous observation that CPU power kept decreasing when T_{C_i} is unchanged.

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

Table 2: CPU power (mW) with different numbers of cores running (with utilization ratio U = 50%).

N_C	f = 384 MHz			f = 1512 MHz		
	P_{BL,N_C}	P_{CPU}	$P_{\Delta,core}$	P_{BL,N_C}	P_{CPU}	$P_{\Delta,core}$
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

$$ED_{C_i} = \frac{T_{C_i}}{E_{C_i}} \quad (2)$$

Generally, ED_{C_i} is a good predictor of CPU power, because 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 5(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 the noise in ED_{C_i}, we applied a weight w_i to ED_{C_i} to form *weighted average entry duration* WED_{C_i}:

$$WED_{C_i} = w_i \times ED_{C_i}, \ where \ w_i = \frac{T_{C_i}}{T_{idle}} \quad (3)$$

Figure 5(d) shows WED_{C_i} in the experiment.

Finally, we model power consumption of a single CPU core working at frequency f as

$$P_{core} = \sum_i \beta_{C_i} \cdot WED_{C_i} + \beta_U \cdot U + c \quad (4)$$

where β_{C_i} and β_U are the coefficients of WED_{C_i} and the utilization ratio U, and c is a constant. For each CPU frequency f supported by Nexus 4, we obtained 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}. Both T_{C_i} and U were calculated from the information obtained from the */proc* filesystem.

3.3.2 Power Modeling for Multi-core CPU

We further conducted an experiment to study how the single-core CPU power model can be extended to multi-core scenario. In the experiment, we enabled different number of CPU cores, which were running at the same frequencies, and then generated the same amount of workload on each enabled core. We measured the CPU

power while varying the core frequencies and utilization ratios. Table 2 presents the results for the cases when core frequencies were fixed at 384 MHz and 1512 MHz, and the core utilization ratio was 50%. In the table, the power increment per core was 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 were enabled, and P_{CPU} is the measured whole CPU power. We can see that $P_{\Delta,core}$ was consistent for the same "frequency/utilization ratio" when there were more than one core enabled, but was notably smaller than the value when there was 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 disable, can only be entered by core-0 when there is no other core is online. Therefore, $P_{\Delta,core}$ for the single-core case is always greater than that for the multi-core case.

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

$$P_{CPU} = P_{BL,N_C} + \sum_i^{N_C} P_{\Delta,core,U_i,f_i} \quad (5)$$

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 ratio U_i. For each frequency f_i, $P_{\Delta,core,U_i,f_i}$ 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. Note that for Nexus 4, we need to model $P_{\Delta,core,U_i,f_i}$ 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.

3.3.3 Experimental Evaluation

We have conducted experiments on Nexus 4 to evaluate our idle-state-based multi-core CPU power model. In the experiments, we developed three benchmark programs that perform busy loop, busy floating point operations and busy cache accesses, respectively. Similar to our workload generator program previously, the busy operations are performed periodically followed by a configurable idle period, and the duration of the busy operations can also be adjusted.

For different number of enabled cores, we performed the experiment for 20 rounds. In each round, we ran each benchmark program on every enabled core, with randomly generated durations for the busy operations and the following idle period. We used our CPU power models to predict the CPU power consumption, and calculated the prediction accuracy by comparing the predicted values to the ground truth values measured by the power meter. The average prediction accuracy for the benchmark programs is 95.6%, ranging from 95.0% to 96.2%, which demonstrates that our models significantly outperform the existing power models.

4. CONCLUSION AND ON-GOING WORK

In the paper we introduced our efforts toward providing better power management on multicore smartphones. We demonstrated that current smartphone applications are not fully utilizing multi-core capability by studying the Web browsing performance with the Chrome browser. We also showed the existing solutions of modeling power consumption of CPUs do not work well in Nexus 4, a quad-core CPU smartphone. Then, we proposed our idle-state-based CPU power model, which is shown to be able to achieve 95.6% prediction accuracy on different types of workloads.

To further understand and optimize multi-core CPU performance on smartphones, we are currently working or plan to work on the following three research directions. 1) *Further improvement on CPU power modeling*. For example, to support heterogeneous multicore architectures, such as ARM big.LITTLE architecture used in Samsung Galaxy S4. 2) *Comprehensive app study on multicore performances*. By using the proposed CPU power model and kernel scheduling tools, we are conducting a comprehensive study on how applications are utilizing smartphones' multi-processing capability as well as the corresponding power efficiency. We expect the study results can help us to identify, analyze and improve the bottlenecks in applications or OSes of fully exploiting computation power of multicore smartphones. 3) *OS/API support for multicore CPU performance improvement in smartphones*. We plan to investigate how to improve applications' ability in exploiting multicore CPUs. E.g., to improve OS's efficiency regarding scheduling smartphone workloads, and to provide developer-friendly APIs which require little or no effort from developers to exploit smartphones' multi-processing capability.

REFERENCES

[1] Multicore Madness in Smartphones, http://eecatalog.com/multicore/2013/04/10/multicore-madness/.

[2] C. J. Jones, R. Liu, L. Meyerovich, K. Asanovic, and R. Bodik, "Parallelizing the Web Browser", in *USENIX HotPar* 2009.

[3] C. Badea, M. R. Haghighat, A. Nicolau, and A. V. Veidenbaum, "Towards Parallelizing the Layout Engine of Firefore", in *USENIX HotPar* 2010.

[4] H. Mai, S. Tang, S. T. King, C. Cascaval, and P. Montesinos, "A Case for Parallelizing Web Pages", in *USENIX HotPar* 2012.

[5] 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.

[6] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. Dick, Z. Mao, and L. Yang, "Accurate Online Power Estimation and Automatic Battery Behavior Based Power Model Generation for Smartphones", in *CODES+ISSS*, 2010.

[7] 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.

[8] Ftrace, http://elinux.org/Ftrace.

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

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

[11] M. Dong, and L. Zhong, "Self-constructive High-rate System Energy Modeling for Battery-powered Mobile Systems", in *MobiSys*, 2011.

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