Graceful Performance Adaption through Hardware-Software Interaction for Autonomous Battery Management of Multicore Smartphones

Anup Das, Domenico Balsamo, Geoff V. Merrett, Bashir M. Al-Hashimi and Francky Catthoor

***Abstract*—Despite advances in multicore smartphone technologies, battery consumption still remains one of customer's least satisfying features. This is because existing energy saving techniques do not consider the electrochemical characteristics of batteries, which causes battery consumption to vary unpredictably, both within and across applications. Additionally, these techniques provide application specific fixed performance degradation in order to reduce energy consumption. Having a performance penalty, even when a battery is fully charged, adds to customer dissatisfaction. We propose a control-based approach for run-time power management of multicore smartphones, which scales the frequency of processing cores in response to the battery consumption, taking into account the electrochemical characteristics of a battery. The objective is to enable graceful performance modulation, which adapts with application and battery availability in a predictable manner, improving quality-of-user-experience. Our control approach is practically demonstrated on embedded Linux running on Cortex A15-based smartphone development platform from nvidia. A thorough validation with mobile and Java workloads demonstrate 2.9x improvement in battery availability compared to state-of-the-art approaches.**

I. Introduction

With every new generation of smartphones, additional features are integrated to improve quality-of-user-experience. Modern smartphones feature multiple high speed processing cores enabling 3D games, 4K videos and desktop class web browsers. While processing cores become high performing, advances in battery technologies have been slow. As a result, the "gap" between energy demand of these features and energy supply from the battery is increasing with every new smartphone generation, reducing the battery life (i.e., operating life between two successive charging). A recent study on smartphones shows CPU power as a major contributor to overall power consumption [1], and is the scope of this work.

Several techniques have been proposed recently to reduce the energy consumption of processing cores. Most of these do not consider the electrochemical characteristics of lithium-ion or lithium-polymer batteries that guide their discharge behavior [2], [3]. Specifically, a portion of the battery energy becomes unavailable [4] during discharge depending on the power consumption (refer to Section III). We demonstrate in this work that ignoring this residual energy can result in significant underestimation of the battery consumption, leading to a reduced (5%-41%) and often unpredictable battery life.

To address this, battery models have been proposed, differing in accuracy, complexity and the number of parameters needed for modeling [5]. Power optimization using these models (eg. [6]) often provide application-specific performance degradation, without considering the battery energy available at the time this application is initiated on the platform. To effectively manage the "battery gap" in modern smartphones, it is essential to match battery supply with consumption: when the battery energy is low, the rate of battery consumption can be reduced by down scaling the processor frequency and gracefully reducing performance; conversely, when the battery energy is high, the rate of battery consumption can be increased by up scaling the processor frequency and gracefully boosting performance. This is the objective of our work.

Contributions: Following are our contributions.

- run-time adaptation of processor frequency in response to the battery energy availability considering the electrochemical characteristics governing residual energy;
- a control approach with feedback from the hardware to adjust processor frequencies such that energy supply from the battery is matched with energy demand from the executing application;
- a variable interval for frequency switching, adaptive with workload; and
- graceful performance adaptation of an application at all battery energy levels.

The proposed approach is validated through its implementation as a run-time manager and is integrated into embedded Linux (eLinux) running on quad core Cortex A15-based smartphone from nvidia. A range of mobile and embedded applications from MiBench and DaCapo Java benchmarks running on this smartphone demonstrate the proposed approach is able to increase the available battery energy by 2.9x over existing approaches. Additionally, the performance modulation feature results in an 18% improvement in the number of instructions executed per second for workloads typically observed on smartphones. The remainder of this paper is organized as follows. Related works are discussed in Section II. A brief introduction to battery energy consumption is provided in Section III and analytical formulation of the proposed control approach implementing the performance modulation in Section IV. Results and discussions are provided next in Section V and the conclusions are presented in Section VI.

II. Related Works

Battery consumption of smartphones remains one of customers' least satisfying features. Significant research studies have been conducted in recent years to improve smartphone's battery consumption by effectively managing multicore processors within these smartphones. Two prominent approaches are dynamic power management (DPM) [2], in which cores are dynamically shutdown or put into sleep modes; and dynamic voltage-frequency scaling (DVFS) [7], [8], in which the operating voltage and frequency are altered dynamically during application execution. There are also techniques, which exploit the trade-off between DPM and DVFS [3], [9], [10]. Most of

TABLE I
SUMMARY OF RELATED WORKS.

Approaches	Battery Model	Technique	Platform	Performance
[2], [3]	×	Q-Learning	Real	Adaptive with applications
[7], [8]	×	Control	Simulation	
[11]–[13]	✓	Control	Simulation	
[6], [14]	✓	Control	Real	
Proposed	✓	Control	Real	Graceful modulation; adaptive with applications and available battery energy

these techniques do not take into account the electrochemical battery discharge characteristics, which determine the available and residual battery energy. As an example, the learning-based approach [3] results in under prediction of battery consumption and over prediction of battery life by 5%-41% (Section V) by not considering battery residual energy.

To overcome this limitation, optimization techniques [6], [11]–[14] have been proposed considering battery discharge models [4]. All these optimization techniques improve battery life while executing a typical workload, incorporating the electrochemical battery characteristics. These approaches differ in the type of control principles (learning vs control theoretic) and the efficiency in solving battery dynamics. Performances obtained using these approaches are usually application-dependent and do not consider the battery energy available at the time of initiation of these applications. As an example, the approach of [6] achieves 22% power savings while playing a 1080p video with a constant 10% performance degradation measured in terms of frames per second. This degradation is fixed for a video, independent of the battery energy at which the video playback is initiated. Although a 10% performance degradation may be acceptable for a low battery (say at 5%), such degradation often adds to users' dissatisfaction particularly when a smartphone is fully charged.

Table I summarizes related works in this domain. As can be seen, all existing works adapt performance with respect to application only. To better address the "battery gap" for modern smartphones, we adapt performance with respect to application demand and battery energy availability, incorporating the electrochemical battery discharge model.

III. MODELING BATTERY CONSUMPTION

We formulate battery energy consumption similar to that used in [15]. We use the following notations,

$\mathcal{S}$	=	rated battery capacity
$\mathcal{S}(T)$	=	battery capacity after T years
t	=	time elapsed since the smartphone is operating on battery, after being charged to its current capacity
$R(t)$	=	residual (unavailable) energy of the battery at time t
$E(t)$	=	energy consumed by the smartphone in the interval t
$P(\tau)$	=	power consumption of the smartphone at time τ
$A(t)$	=	available energy of the battery at time t

The battery capacity can be expressed as $\mathcal{S}(T) = \mathcal{S} - \alpha \cdot T$, where α is the average aging of the battery over years, which depends on factors such as the ambient temperature, the number of charging-discharging cycles, etc. For simplicity, an average value is considered in this work [16]. The energy consumption of a smartphone in the time interval 0 to t is given by

$$E(t) = \int_0^t P(\tau) d\tau \tag{1}$$

Most smartphones are not equipped with a power meter. Several models have been proposed recently to estimate the power consumption $P(\tau)$ at run-time. We use the model proposed in our earlier work [17], which uses ARM CPU performance monitoring counters (PMCs) to estimate the power consumption of an application. This model is shown to be accurate within 5% for ARM A15 cores.

The residual battery energy can be expressed as [15]

$$R(t) = \int_t \mathcal{G}\left(P(\tau)\right) d\tau \tag{2}$$

where $\mathcal{G}$ is a function of the electrochemical characteristics of the battery and is also dependent on the current power consumption. The available energy at time t is expressed as

$$A(t) = \mathcal{S}(T) - [E(t) + R(t)] = \mathcal{S}(T) - C(t) \tag{3}$$

where $C(t)$ is the battery consumption during the time interval 0 to t. In our proposed approach, the control parameter is the frequency of processing cores, which governs the energy consumption $E(t)$. The objective of our approach is to minimize the battery energy consumption $C(t)$. To do so, it is necessary to measure or model the residual energy $R(t)$. Although there are models that provide over 95% accuracy, these models are fairly complex involving fifteen or more parameters [5] and are associated with large computation overhead, which is crucial from real-time perspective. We therefore adopt measurement-based approach similar to [15]. For this, we use the battery meter accessible in smartphones, which is an indication of the available energy, i.e., the battery percentage m is related as

$$m = \frac{A(t)}{\mathcal{S}(T)} \cdot 100 \tag{4}$$

The residual energy in terms of the battery indicator m is

$$R(t) = \left[1 - \frac{m}{100}\right] \cdot \mathcal{S}(T) - E(t) \tag{5}$$

In our work, we try to minimize the residual energy $R(t)$ and increase the delivered energy $E(t)$, keeping the consumption constant $C(t)$, such that an application can use this energy to deliver higher performance. In the subsequent section, we formulate the optimization problem we aim to solve.

IV. PROPOSED APPROACH

Figure 1 shows the block diagram of a typical smartphone, with the software and the hardware stacks. Our approach is implemented as a closed-loop controller within the software stack. The hardware stack consists of a multicore processor, a DVFS controller and PMCs for performance statistics. The controller interfaces with the OS, instructing it to alter the operating frequency by one step each time. This instruction is translated to the hardware using the `cpufreq` API which sets the DVFS controller appropriately. This is the **feedforward** part of the controller. Performance counters, together with

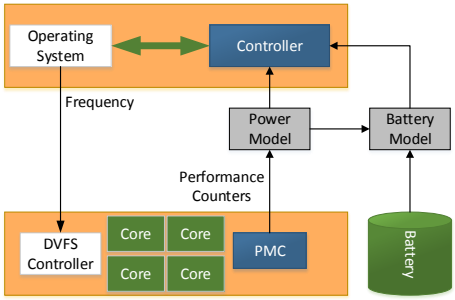

Fig. 1. Block diagram of the proposed approach.

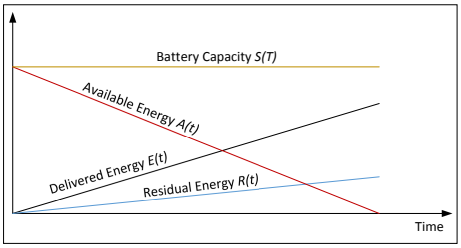

Fig. 2. Available energy at a constant power consumption.

the battery indicator are used to model the battery energy consumption. This is the **feedback** loop of our controller.

Figure 2 plots the available battery energy $A(t)$ as a function of time for a fixed power consumption and a battery capacity $\mathcal{S}(T)$. Both, the delivered energy $E(t)$ and the residual energy $R(t)$ increase with time t. This causes the available energy $A(t)$ to decrease with time t as seen in the figure. This can be expressed as $A(t) = A_0 - k \cdot t$, where $A_0 = \mathcal{S}(T)$ is the initial battery energy, which is equal to the battery capacity and $k = \frac{C(t)}{t}$ is the rate of decrease in available energy.

A. Modeling the Dynamics of the Proposed Controller

Without Workload Adaptation: Our controller adapts processor frequency in response to the battery consumption, which in turn adapts both the delivered energy $E(t)$ and the residual energy $R(t)$. This results in decrease of available energy over time as shown in Figure 3. We formulate the controller action as follows. Let f_1, f_2, f_3 and f_4 be the four frequencies supported on the hardware; A_1 be the available battery energy at start of execution (time $\tau = 0$); and N be the fixed frequency switching interval (this is relaxed in the next section), i.e., at every N cycles, the controller alters the frequency by one step. Using Figure 3, $A_2 = A_1 - k_1 \cdot (t_1 - 0)$, where k_1 is the rate of decrease in available energy, which is a function of the power consumption at f_1. Generalizing, the available battery energy after the i^{th} interval

$$A_{i+1} = A_i - k_i \cdot (t_i - t_{i-1}) = A_i - k_i \cdot \frac{N}{f_i} \tag{6}$$

where the time interval $(t_i - t_{i-1})$ can be expressed in terms of the number of cycles N and the frequency f_i as $(t_i - t_{i-1}) = \frac{N}{f_i}$. In the following equations, we determine the frequency switching interval N using a worst case scenario that a given application is executed until the battery energy drains to zero.

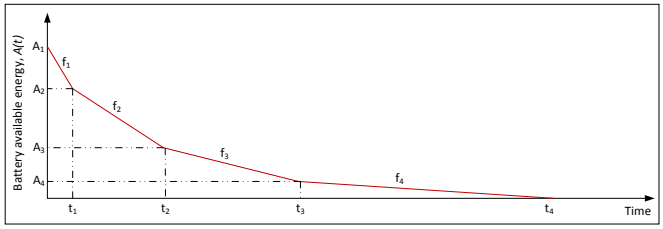

Fig. 3. Adaptation of the available energy.

Using this worst case boundary condition of $A_5 = 0$, we can rewrite Equation 6 for the last interval as

$$A_5 = 0 = A_4 - k_4 \cdot \frac{N}{f_4} \tag{7}$$

Using $i = 1, 2$ and 3 in Equation 6 and substituting their values in Equation 7, the following can be written

$$0 = A_1 - k_1 \cdot \frac{N}{f_1} - k_2 \cdot \frac{N}{f_2} - k_3 \cdot \frac{N}{f_3} - k_4 \cdot \frac{N}{f_4} \tag{8}$$

The number of clock cycles needed before changing the frequency, N can be expressed as

$$N = A_1 \cdot \left[\sum_{i=1}^{4} \frac{k_i}{f_i}\right]^{-1} = A_1 \cdot \left[\sum_{i=1}^{F} \frac{k_i}{f_i}\right]^{-1} \text{ (generalizing to F frequencies)} \tag{9}$$

Extension to Workload Adaptation: In all existing works, N is fixed. However, power consumption in the switching interval (N cycles) can change due to variations in workload of an executing application. Therefore, the actual available energy at the end of the i^{th} interval can be greater or lower than the predicted value A_{i+1} (Equation 6). To adapt to application workload dynamics, we propose to re-evaluate the switching interval after every period, considering the actual available battery energy at that instant. For the first period, the controller waits N_1 cycles before altering the frequency, where

$$N_1 = A_1 \cdot \left[\sum_{i=1}^{F} \frac{k_i}{f_i}\right]^{-1} \tag{10}$$

After N_1 cycles, the controller reduces the frequency by one step and re-evaluates the frequency switching interval N_2 as

$$N_2 = A_2 \cdot \left[\sum_{i=2}^{F} \frac{k_i}{f_i}\right]^{-1} \tag{11}$$

where A_2 is now the actual battery energy at the start of the second period and the scale down frequency steps involve $f_2 \cdots f_F$ (f_1 is already used in the first period). Generalizing, the frequency switching interval for the $(i+1)^{\text{th}}$ period is

$$N_{i+1} = A_{i+1} \cdot \left[\sum_{j=i+1}^{N_f} \frac{k_j}{f_j}\right]^{-1} \tag{12}$$

B. Proposed Control Algorithm

Algorithm 1 shows the pseudo-code of our control approach implemented inside the software stack of a smartphone. An application starts executing at the highest frequency for the switching interval N_1 calculated using Equation 10. When this interval expires, the algorithm re-evaluates the frequency switching interval and reduces the frequency by one step at a time. This process is continued as long as the application is executed or until the lowest frequency is reached.

ALGORITHM 1: Control algorithm implimented in the OS.

Input: Battery availability A_i, invocation i.
Output: Frequency level f, switching interval N.

```
if i = 1 then
    f = f_1; //Set the highest frequency for the cores
    Calculate switching interval N = N_1 using Equation 10;
end
else
    if i ≥ N_c then f = f_{N_c} else f = f_i; //Set the corresponding frequency level
    Calculate switching interval N = N_i using Equation 12;
end
cpufreq-set -f f; //Setting the voltage and frequency
Execute application for N cycles ;
```

TABLE II
FEATURES OF JETSON DEVELOPMENT BOARD.

CPU	4+1 Cortex A15	GPU	Kepler GPU (192 cores)
SoC Architecture	Tegra K1 SoC	L1 Cache	32KB (I) + 32KB (D)
Operating System	eLinux	L2 Cache	2MB
Core Frequency	50MHz – 2.32GHz	RAM	2 GB
Selected Frequency	2.32, 2.22, 2.11, 1.94, 1.73, 1.63, 1.43, 1.33, 1.12, 0.96 (GHz)		

C. Analytical Estimation of Performance

Let IPC be the average number of instructions executed per CPU cycle. Assuming an application is executed until the battery drains out, the total number of CPU cycles consumed by the application and the performance (measured in terms of instructions executed), using our approach is given by

$$N = \sum_{i=1}^{F} N_i \text{ and } I_{proposed} = \sum_{i=1}^{F} N_i \cdot \texttt{IPC}_i \tag{13}$$

where $\texttt{IPC}_i$ is the average instructions per cycle in the i^{th} interval. In existing approaches, a fixed frequency (say f_{soa}) is decided for every application. The rate of decrease in available energy at this frequency is k_{soa}. The time taken by the application to drain the available energy to zero is

$$t_{soa} = \frac{A_1}{k_{soa}} \tag{14}$$

The number of elapsed CPU cycles during this time and the performance using state-of-the-art approaches are

$$N_{soa} = f_{soa} \cdot t_{soa} = \frac{f_{soa} \cdot A_1}{k_{soa}} \text{ and } I_{soa} = N_{soa} \cdot \texttt{IPC}_{soa} \tag{15}$$

where $\texttt{IPC}_{soa}$ is the average instructions per cycle using this approach. In Section V we evaluate and compare the performance of our approach with an existing approach using a range of mobile and embedded applications.

V. RESULTS AND DISCUSSIONS

All experiments are conducted on nvidia's Jetson development board (Table II). The scope of this work is CPU-based homogeneous systems. The GPU is forced into sleep mode by the OS to minimize its impact on the CPU. Our ongoing work demonstrates power saving principle considering CPU-GPU based heterogeneous systems. The experimental setup used in this work is shown in Figure 4. The Jetson development board is powered using Agilent Technologies DC Power Analyzer, which is configured as a lithium-ion battery. The development board runs embedded Linux (eLinux) and the proposed controller is implemented as part of this OS. An external computer is used to inject applications on the development board.

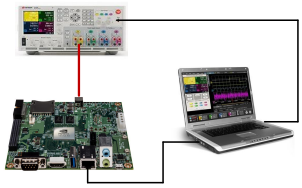

Fig. 4. Experimental setup for validating the proposed approach.

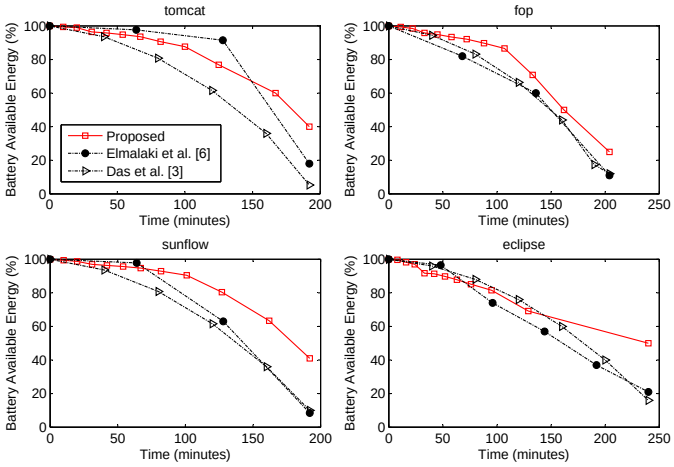

Fig. 5. Battery discharge behavior for four applications.

The proposed run-time approach is validated with a range of applications from the MiBench [18] and the DaCapo Java benchmarks [19] and compared against two existing approaches – the learning-based approach of [3] and the control-based approach of [6]. We have chosen 10 unique voltage-frequency pairs for our experiments (Table II).

A. Battery Discharge Behavior

Figure 5 plots the battery discharge behavior using our proposed approach while executing four Java applications. The discharge behavior is compared against two existing approaches – the control approach of Elmalaki et al. [6] and the learning-based approach of Das et al. [3]. For this comparison, the battery is charged to its full capacity (100%) before firing each of these applications for all three approaches. As seen for the tomcat application, battery discharge is similar among the three approaches during the initial few minutes. As time progresses, the available battery energy using [6] is the highest compared to the other two approaches up to around 120 minutes. Beyond this time, the battery energy starts declining at a faster rate, dropping below our approach when the application finishes execution at around 190 minutes. This rapid drop is because graceful performance modulation is not considered in [6]. Battery energy using [3] is the lowest overall. This is because, in this technique the electrochemical characteristics governing the residual energy

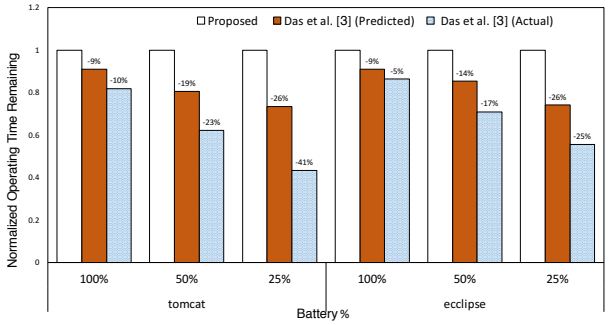

Fig. 6. Battery under prediction using [3] compared to the proposed approach.

is not accounted. Our approach results in 2.2x and 7.6x higher available battery energy than [6] and [3], respectively when the application finishes. A similar trend is observed for the three other applications. For the `sunflow` and the `fop`, our approach results in higher available battery energy throughout the duration of these applications. On average, our proposed approach results in 2.9x and 4.1x higher available energy compared to [6] and [3], respectively.

B. Battery Lifetime Overestimate

Figure 6 reports the remaining operating time of a battery when executing two Java applications – `tomcat` and `eclipse`. These applications are fired at three starting battery percentages as shown in the figure. The battery percentage indicates fraction of the battery capacity available for executing an application. The remaining operating time indicates the time (in minutes, but normalized with respect to the first bar in the figure), a battery will last using the remaining capacity while executing the application. Three results are reported for each application and battery percentage. The first bar in the figure is the result obtained using our proposed approach, which incorporates the residual battery energy. The second bar in the figure is the result obtained using [3]. However, this technique does not consider the residual battery energy, and therefore the predicted remaining operating time is an overestimate. The actual operating time using this technique is shown with the third bar. As we see, not taking into account the battery's residual energy can result in the battery remaining operating time to be 5% – 41% lower than predicted for all applications (bar 2 vs bar 3). The implication is that, if this existing technique is used for a smartphone, an user may experience a shorter battery operating time than what is conveyed using the battery meter. In other words, if 10% battery translates to 15 minutes of video playback one time before the battery runs out, the same video playback application may last more than or less than 15 minutes (with 10% battery) the next time, depending on the starting battery percentage at which the video application is initiated. This unpredictability causes dissatisfaction to end users as logging the battery percentage and inferring the time a battery will last when an application is initiated, is not always feasible.

Using our approach, a battery will last the same time every time for an application, independent of when the application is initiated. Additionally, our approach makes better utilization of the remaining battery capacity delivering up to 36% higher battery life than the existing approach of [3].

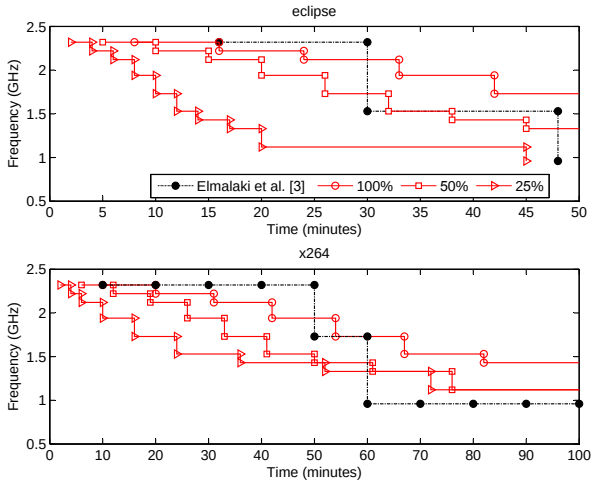

Fig. 7. Graceful performance modulation with different battery availability. Results of the proposed approach are indicated with red solid lines. Results using Elmalaki et al. [6] is indicated with black dashed lines.

C. Graceful Performance Modulation

To demonstrate the graceful performance modulation of our proposed approach, Figure 7 plots the frequency selection using our approach and using the existing technique of Elmalaki et al. [6]. Results using our approach are reported for three different starting battery percentages (100%, 50% and 25%). These are marked in the figure by solid lines. Technique [6] selects frequencies without considering the battery availability; results using this technique are indicated by dashed lines. Frequency selection results are demonstrated for two applications – the `eclipse`, a Java application and the `x264`, a video decoding application. As seen for the `eclipse` application, our proposed approach steps through the frequency levels progressively for all three battery scenarios providing a graceful performance modulation. However, the difference is in terms of the time spent at the different frequency levels. With a fully charged battery (100%), our approach scales down the frequency after about 8 minutes. For 50% and 25% charged batteries, scaling starts after about 5 and 2 minutes, respectively in order to step through all the frequency levels. It is also interesting to note that when starting with a 100% charged battery, our approach scales through the higher five frequency levels only (refer to Table II), delivering high performance to users. When the battery is low (25% charged), it steps through all frequency levels. In comparison to this, the existing technique selects only three frequencies (2.32 GHz, 1.53 GHz and 0.96 GHz) skipping the intermediate frequency steps (Table II). As a result, an user sees a sharp degradation of quality in the duration of an application. A similar trend is observed for all other applications used.

To summarize, Table III reports frequencies selected by the proposed approach and the approach of [6] for a range of

TABLE III
FREQUENCY SELECTION USING PROPOSED AND EXISTING TECHNIQUE.

Battery	Applications	Frequency (GHz)	
		Proposed	Elmalaki et al. [6]
100%	eclipse	5 (2.32–1.73)	3 (2.32,1.53, 0.96)
	tomcat	2 (2.32–2.22)	3 (2.32,1.73, 0.96)
	tradesoap	7 (2.32–1.43)	4 (2.32, 2.12, 1.33, 0.96)
	adpcm	4 (2.32–1.94)	2 (2.32, 1.12)
	x264	7 (2.32–1.43)	3 (2.32,1.73, 0.96)
50%	eclipse	7 (2.32–1.43)	3 (2.32,1.53, 0.96)
	tomcat	10 (2.32–0.96)	3 (2.32,1.73, 0.96)
	tradesoap	10 (2.32–0.96)	4 (2.32, 2.12, 1.33, 0.96)
	adpcm	8 (2.32–1.33)	2 (2.32, 1.12)
	x264	10 (2.32–0.96)	3 (2.32,1.73, 0.96)

TABLE IV
AVERAGE INSTRUCTIONS EXECUTED PER SECOND USING THE PROPOSED APPROACH IN COMPARISON WITH THE TWO EXISTING APPROACHES.

Battery	Applications	Average instructions per second (Million)		
		Proposed	Elmalaki et al. [6]	Das et al. [3]
100%	eclipse	12.9	12.4	12.0
	tomcat	10.1	10.1	10.1
	tradesoap	8.7	8.7	7.8
	adpcm	9.6	9.6	9.6
	x264	11.1	10.2	9.9
50%	eclipse	11.3	8.8	7.1
	tomcat	9.7	7.7	7.5
	tradesoap	8.0	7.9	7.3
	adpcm	9.1	7.6	6.6
	x264	10.6	9.2	8.9

applications. Results are also reported for starting battery percentages of 100% and 50%, respectively. As seen in Table III, starting with 50% battery energy, our proposed approach steps through more frequency levels than starting with 100% (i.e., fully charged). This implies that our approach delivers higher performance (due to less frequency scaling) when starting with high battery available energy. On the other hand, starting with 50% battery energy, the proposed approach steps through many frequencies, gracefully modulating the performance.

D. Performance Improvement

Table IV reports the number of instructions executed by our proposed approach for five applications starting at two different battery energy levels. Results are compared against two existing approaches – Elmalaki et al. [6] and Das et al. [3]. As can be seen from the table, starting at 100% battery charge, the average number of instructions executed per second by all the three approaches are similar, with the proposed approach delivering higher performance by average 2.6% and 6.2% compared to [6] and [3], respectively. Starting at lower battery energy, the average performance improvements using our approach are 18% and 31% compared to these other two approaches, respectively. Summarizing, our proposed approach delivers similar performance to existing approaches at higher starting battery levels. However, when applications are initiated at lower available energy levels, performance using our proposed approach is significantly better.

VI. CONCLUSION

We propose a control-based approach to gracefully modulate application performance. This approach takes into account the residual energy of a battery, which is governed by its electrochemical characteristics. This approach is integrated inside eLinux running on a smartphone SoC from nvidia, featuring a quad-core ARM A15 CPU. Our approach is practically validated with a range of multimedia applications from the MiBench and Java applications from the DaCapo benchmark suite. Results demonstrate the our proposed approach increases the battery available energy by an average 2.9x compared to existing approaches. Additionally, our approach ensures a graceful performance modulation when compared to existing approaches, improving the quality-of-user-experience. In terms of performance, our approach delivers similar performance to existing approaches, when the battery energy is high. However, at low battery energy, the performance delivered using our approach is significantly better, achieving an average 18% improvements. Our ongoing work is to demonstrate battery improvement considering CPU-GPU based systems.

REFERENCES

[1] A. Carroll and G. Heiser, "An analysis of power consumption in a smartphone." in *USENIX Annual Technical Conference*, 2010.

[2] R. Ye and Q. Xu, "Learning-based power management for multicore processors via idle period manipulation," *IEEE Transactions on CAD of Integrated Circuits and Systems (TCAD)*, 2014.

[3] A. Das, M. J. Walker, A. Hansson, B. M. Al-Hashimi, and G. V. Merrett, "Hardware-software interaction for run-time power optimization: A case study of embedded linux on multicore smartphones," in *ISLPED*, 2015.

[4] M. Gholizadeh and F. R. Salmasi, "Estimation of state of charge, unknown nonlinearities, and state of health of a lithium-ion battery based on a comprehensive unobservable model," *IEEE Transactions on Industrial Electronics*, 2014.

[5] R. Rao, S. Vrudhula, and D. N. Rakhmatov, "Battery modeling for energy aware system design," *IEEE Computer*, 2003.

[6] S. Elmalaki, M. Gottscho, P. Gupta *et al.*, "A case for battery charging-aware power management and deferrable task scheduling in smartphones," in *USENIX Power-Aware Computing and Systems*, 2014.

[7] M. E. Salehi, M. Samadi, M. Najibi, A. Afzali-Kusha, M. Pedram, and S. M. Fakhraie, "Dynamic voltage and frequency scheduling for embedded processors considering power/performance tradeoffs," *IEEE Transactions on VLSI Systems (TVLSI)*, 2011.

[8] V. Hanumaiah and S. Vrudhula, "Energy-efficient operation of multicore processors by dvfs, task migration, and active cooling," *IEEE Transactions on Computers*, 2014.

[9] V. Devadas and H. Aydin, "On the interplay of voltage/frequency scaling and device power management for frame-based real-time embedded applications," *IEEE Transactions on Computers*, 2012.

[10] G. Chen, K. Huang, and A. Knoll, "Energy optimization for real-time multiprocessor system-on-chip with optimal DVFS and DPM combination," *ACM TECS*, 2014.

[11] H. Shen, Q. Chen, and Q. Qiu, "Battery aware stochastic QoS boosting in mobile computing devices," in *DATE*, 2014.

[12] X. He, R. P. Dick, and R. Joseph, "Embedded system and application aware design of deregulated energy delivery systems," in *CASES*, 2015.

[13] D. Shin, E. Macii, and M. Poncino, "Statistical battery models and variation-aware battery management," in *DAC*, 2014.

[14] W. Lee, Y. Wang, D. Shin, N. Chang, and M. Pedram, "Optimizing the power delivery network in a smartphone platform," *IEEE Transactions on CAD of Integrated Circuits and Systems (TCAD)*, 2014.

[15] M. Kim, Y. G. Kim, S. W. Chung, and C. H. Kim, "Measuring variance between smartphone energy consumption and battery life," *IEEE Computer*, 2014.

[16] J. Lee, Y. Chon, and H. Cha, "Evaluating battery aging on mobile devices," in *DAC*, 2015.

[17] "Omitted for Blind Review," 2016.

[18] M. Guthaus, J. Ringenberg, D. Ernst, T. Austin, T. Mudge *et al.*, "MiBench: A free, commercially representative embedded benchmark suite," in *IEEE Workshop on Workload Characterization*, 2001.

[19] S. M. Blackburn, R. Garner, C. Hoffmann, A. M. Khang, K. S. McKinley, R. Bentzur *et al.*, "The DaCapo Benchmarks: Java benchmarking development and analysis," in *ACM Sigplan Notices*, 2006.