Workload- and User-aware Battery Lifetime Management for Mobile SoCs

Sofiane Chetoui, Sherief Reda
School of Engineering, Brown University, Providence, Rhode Island 02912
Email: {sofiane_chetoui,sherief_reda}@brown.edu

***Abstract*—Mobile devices have become an essential part of daily life with the increased computing capabilities and features. For battery powered devices, the user experience depends on both quality-of-service (QoS) and battery lifetime. Previous works have been proposed to balance QoS and battery lifetime of mobile devices; however, they often consider only the CPU. Additionally, they fail in considering the user's desired battery lifetime while having a high QoS variation, which undermine the user satisfaction. In this work, we propose a CPU-GPU workload- and user-aware battery lifetime management technique for mobile devices using machine learning. Firstly, we design a workload-aware governor through an offline and an online analysis. A set of CPU and GPU performance counters is used during the offline analysis to identify a set of canonical phases (CP). In runtime, k-means is used to classify each sample of the performance counters to one of the predefined CP. Afterwards, we build a model that predicts the energy consumption given the user usage history. Finally, the energy model is used to find the optimal frequency settings for the CPU and GPU to provide the best QoS while meeting the target battery lifetime. The evaluation of the proposed work against state of the art techniques in a commercial smartphone, shows 15.8% and 9.4% performance improvement on the CPU and GPU, respectively. The proposed technique also shows 10× improvement in QoS variation, while meeting the desired battery lifetime.**

***Index Terms*—Mobile Devices, CPU, GPU, Battery Management**

I. Introduction

In recent years, increased processing capability of mobile devices has enabled new applications that require intensive computation. However, mobile devices are battery constrained, thus developing energy management techniques to provide an extended battery lifetime, while meeting the user performance expectation is critical for the user experience.

Battery lifetime has become one of the top usability concerns of mobile systems [1]. Various studies [2]–[4] tried to build systems that adaptively balance performance and battery lifetime, since the user experience is mainly determined by these two factors. In a study about users and batteries interactions [5], the authors show that there is a great variation among users in the battery usage and recharge patterns. Thus, to best serve the user, the mobile device should consider the user desired battery lifetime. In this paper we refer to the performance as *quality of service* (QoS), which represents the runtime for the CPU and the frames per second for the GPU.

In some prior efforts [1]–[3], [6], only the CPU is considered by the proposed systems, while one of the highly power consuming units in the mobile SoC, like the GPU, is not taken into account. Additionally, workload-awareness allows to make informed management decisions, which potentially could improve performance or save energy; however it has not been widely explored. Mobile systems should also minimize the performance variation to provide a seamless usage experience to the user. In this paper we refer to the performance variation as the *QoS variation*, which is the percentage difference between the highest and the lowest performance achieved through several runs.

In this paper, we propose a novel workload- and user-aware battery lifetime management that maximizes the performance under the user's desired battery lifetime. Our approach leverages insights about the running workloads by collecting CPU-GPU performance counters, which are used to proactively scale the CPU-GPU frequencies using machine learning. Additionally, to enable the user-awareness we design a model that predicts energy consumption based on the user usage history.

To summarize, the contributions of this paper are as follows:

- We design the first workload- and user-aware battery lifetime management technique. The workload-awareness is achieved through performance counters, while the user awareness is incorporated by representing the user-usage history through a set of canonical phases (CP).
- We propose a novel model that predicts the energy consumption based on the user usage history. This model makes the proposed technique user-aware, and helps in better meeting the user desired lifetime.
- The proposed battery lifetime management is achieved by scaling both the CPU and the GPU DVFS levels, unlike previous techniques that do not consider the GPU.
- We implement our technique on a commercial smartphone and compare its performance against state-of-the-art battery management techniques. We show that our technique achieves 15.8% and 9.4% *QoS* improvement on the CPU and GPU, respectively, while meeting the lifetime target and decreasing the *QoS variation* by 10x.

The paper is organized as follows: Section II introduces related work. Section III motivates the proposed work. Section IV describes our proposed technique. Section V presents the evaluation results of our technique compared against state-of-the-art techniques. Section VI concludes the paper.

II. Related work

Various studies tried to offer better energy savings by studying DVFS governors. In particular, Park *et al.* [7] proposed a cooperative CPU-GPU frequency capping technique that avoids unnecessary high frequencies. A hierarchical FSM-based

978-3-9819263-5-4/DATE21/©2021 EDAA

frequency capping technique was proposed to save power by allowing minor degradation in performance [8]. Choi *et al.* designed a graphics-aware power governing technique that solves the energy inefficiency of frame rendering [9]. A memory-aware cooperative CPU-GPU DVFS governor was proposed to maximize the energy efficiency while meeting a performance target [10]. A phase-aware web browser power management technique was proposed by Peters *et al.* [11]. In an other study, an energy-efficient mobile web interaction framework that leverages a cloud-based machine learning model was proposed [12]. Most of these techniques allow a slight degradation in performance to improve the energy efficiency, as result, they do not properly offer a fine-grain balancing of the QoS and the battery lifetime. Furthermore, they do not take into consideration the user's desired battery lifetime goal.

The most used technique in commercial smartphones for battery lifetime management is Powersave [13]. It sets the allowed frequency to the highest possible level, and when 20% of the battery capacity is reached, the allowed frequency is set to a lower level.

Prior studies have also explored improving the user satisfaction by balancing the battery lifetime and the QoS. Yan *et al.* [1] proposed a quality of experience(QoE)-aware frequency governor which dynamically scales the CPU frequency at low battery levels. Donohoo *et al.* [3] proposed a framework that optimizes the CPU and the screen backlight energy consumption. Poyraz *et al.* [14] used built-in sensors to predict the user satisfaction for CPU Settings to save energy. In a recent study, Lee *et al.* [2] proposed BUSQ1 and BUSQ3, two dynamic QoS scaling approaches that automatically balance QoS and energy. BUSQ1 defines the discharge profile linearly, while BUSQ3 defines it based on the user usage history. Afterwards, by comparing the current battery status against the predefined discharge profile, the frequency is decreased if energy savings are needed and is increased otherwise. However, the previously mentioned studies do not take into consideration the GPU. Furthermore, most of these studies do not leverage any workload-awareness, rather, they scale the frequency independently of the running workload. Thus, wasting several opportunities to save energy or to improve performance.

In order to extensively evaluate the proposed work, we compare our technique against Powersave, BUSQ1 and BUSQ3 [2]. Powersave is chosen because it is the most widely used technique in commercial smartphones, while BUSQ1 and BUSQ3 [2] are chosen because of the similarity of their problem formulation to our proposed work.

III. Motivation

The optimal user experience for mobile devices depends on providing a seamless performance until the next battery recharge. However, mobile devices are used differently across users. Furthermore, an important portion of the power consumption and performance in the new generation of mobile devices depends on other computing units than the CPU. Hence, we highlight four main motivations for the proposed work.

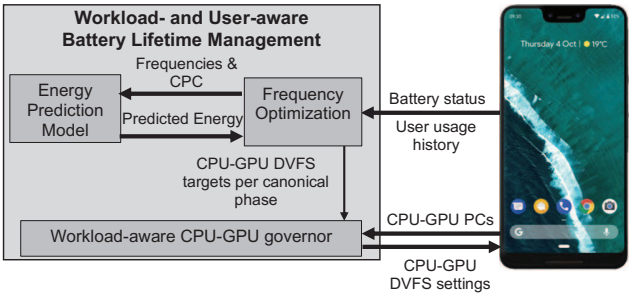

Fig. 1. Workload- and User-aware Battery Lifetime Management

a) Maximizing QoS under target battery lifetime: A recent study has shown that a considerable portion of the recharges are driven by context (e.g. time, location,etc.) and there is a great variation among users in exploiting the available battery charge [5]. Hence, to provide the best balance between QoS and battery lifetime, a management technique should consider the user desired target battery lifetime. Hence, the problem formulation of this work aims at maximizing the QoS, which is about minimizing workload runtime for the CPU and maximizing the FPS for the GPU, given a desired target battery lifetime.

b) High QoS variation: We implement several battery lifetime management techniques, namely, Powersave [13], BUSQ1 and BUSQ3 [2]. By evaluating the QoS variation we show that the performance variation can be as high as 60%, which greatly affects the user experience. Thus, we need to devise a management technique that provides a smooth experience to the user.

c) Workload-aware management: By running different workloads on the Google Pixel 2 XL at 2.45 GHz, 2.20 GHz and 2.11 GHz, we show that the decrease in performance caused by the frequency downscaling highly varies based on the workload. The performance decrease ranges from 2% to 10% for 2.20 GHz, and 2% to 15% for 2.11 GHz, as compared to the performance at 2.45 GHz. Thus, a workload-aware management would allow to save power when the frequency is over-scaled, and to improve performance when it is under-scaled.

d) Cooperative CPU-GPU management: Running 3DMark as a GPU workload and Geekbench as a CPU workload on a commercial smartphone, we show that the GPU has a higher power profile with an average power of 6.8 Watts, as compared to 5.5 Watts for the CPU workload. Thus, it is crucial to include both the CPU and the GPU in the management technique.

IV. Proposed Work

The proposed battery lifetime management technique, which is depicted in Figure 1, uses frequency optimization and an energy prediction model to find the optimal CPU-GPU frequency targets per canonical phase (CP). The workload-aware governor continuously collects performance counters (PC), and classifies each sample as one of the CP, then the frequency is set to the frequency target corresponding to this CP, such that QoS is optimized while meeting the target battery lifetime. In the following subsections, we describe the main blocks of the

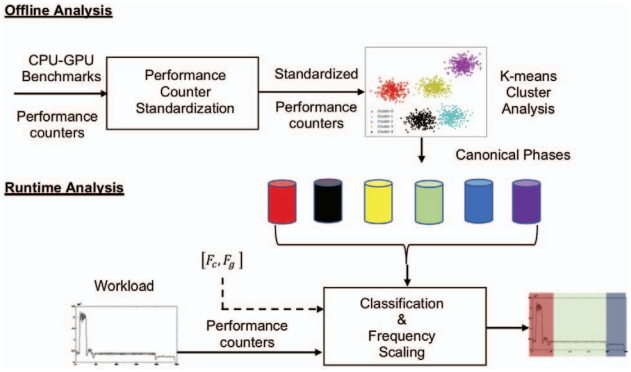

Fig. 2. Workload-aware governor

proposed work, namely: A) the workload-aware governor, B) the energy prediction model and C) the frequency optimization.

A. Workload-aware governor

The workload-aware governor (WLA) shown in Figure 2, is designed through an offline analysis, during which a set of CPs are identified through cluster analysis. Then in runtime, the workload-aware governor is used with the frequency optimization and the energy prediction model to choose the optimal DVFS settings.

In offline analysis we aim to identify a set of CPs that can be used to detect the different workload phases. The different steps of the analysis are presented in Algorithm 1. As shown in Figure 2, we start by running a benchmark suite at different frequency levels, while collecting the following PC on the CPU side: instructions executed, branch-misses, cache misses and cache references, and on the GPU side we collect: GPU bus frequency and normalized GPU utilization.

Since the values of the PCs are a function of the DVFS setting, we need to standardize their values. We build a mean and a standard deviation models of the PCs using regression analysis, as shown in line 3, 4 and 5 of Algorithm 1. In line 6 the standardization is performed by subtracting the mean and dividing by the standard deviation. The GPU performance counters are normalized in regard of their maximum possible value. Finally, we conduct a k-means cluster analysis [15]. We identify six (i.e., $k = 6$) clusters (i.e., CPs), and compute their corresponding centroids as shown in line 7 of Algorithm 1. The number of clusters was chosen such that each resulting canonical phase represents an actual workload phase. This is achieved by increasing the number of clusters, while keeping track of the time ratio of each CP when running different workloads. This time ratio represents the time spent on each CP, and it is referred to as the canonical phase composition (CPC).

For illustration, Table I gives the *canonical phase composition* (CPC) of various workloads. The table shows that the memory copy and memory latency workloads are mainly dominated by CP1, which means that CP1 corresponds to memory operation phases. The table also shows that all the remaining workloads are mainly dominated by the combination of CP1 and one of the remaining CP. Finally, we notice that

Algorithm 1: Offline analysis of the WLA

Output: C_{CP} CP centroids

1 Run benchmark suite at different frequency levels
2 PC $\leftarrow$ collect performance counters
3 $\overrightarrow{\mathrm{m}}$, $\overrightarrow{\mathrm{s}} \leftarrow$ means and standard deviations of PCs
4 Mean PC model: $\overrightarrow{\mathrm{w_m}} = (\mathbf{F_c}^\intercal\ \mathbf{F_c})^{-1}\mathbf{F_c}^\intercal\ \overrightarrow{\mathrm{m}}$
5 Std PC model : $\overrightarrow{\mathrm{w_s}} = (\mathbf{F_c}^\intercal\ \mathbf{F_c})^{-1}\mathbf{F_c}^\intercal\ \overrightarrow{\mathrm{s}}$
6 $PC_s \leftarrow$ Standardize(PC,$\overrightarrow{\mathrm{w_m}}$, $\overrightarrow{\mathrm{w_s}}$)
7 $C_{CP} \leftarrow$ Apply k-means on PC_s and return centroids

TABLE I
THE CANONICAL PHASE COMPOSITION OF DIFFERENT WORKLOADS

	CP1 (%)	CP2 (%)	CP3 (%)	CP4 (%)	CP5 (%)	CP6 (%)
Ray Tracing	33	1	1	0	63	2
LLVM	31	4	56	0	4	5
Gaussian Blur	46	47	2	0	3	2
Speech Recognition	31	0	0	0	1	68
Memory Copy	96	0	1	0	3	0
Memory Latency	95	1	1	0	2	1
Sling Shot (GPU)	13	0	0	87	0	0

Sling Shot, which is a GPU workload, is mainly dominated by CP4, which means that CP4 corresponds to a GPU workload.

B. Energy prediction model

In order to maximize the performance while meeting a target battery lifetime, for a set of CPU-GPU frequencies we need to be able to predict the energy consumption given the user usage history. For this goal, we represent the usage history through a user-specific CPC vector, which gives the percentage of time the user spends in each CP throughout the day. Each CPC vector represents the fractions of time spent by the user in each phase aggregated over workloads used throughout the day. Then, we use offline regression analysis to build an energy prediction model based on the CPs discussed earlier. The model is defined as follows:

$$EP(F,CPC) = T_r \sum_{i=1}^{i=k} (w_{ci} \cdot CPC_i \cdot Fc_i + w_{gi} \cdot CPC_i \cdot Fg_i) + c, \quad (1)$$

where F represents a vector of CPU and GPU frequency pairs (one pair per CP), Fc_i and Fg_i correspond to the CPU and GPU frequencies of the i^{th} CP, CPC_i is the i^{th} element of the CPC vector, T_r denotes the total runtime, k represents the number of CPs, and w_{ci}, w_{gi} and c are the weights to be determined by the regression analysis.

C. Frequency optimization

The frequency optimization aims to find the best CPU-GPU frequencies per CP. The optimization consists of two steps that rely on the energy prediction model to find the optimal frequency settings given the desired battery lifetime.

The CPU and the GPU combined have usually more than 30 DVFS levels, which makes the task of choosing the best settings per CP intractable. In order to solve this efficiently, we construct offline a phase-aware performance-energy trade-off

Algorithm 2: Workload- and User-aware Battery Lifetime Management

Input: C_{CP} (CP centroids), B_s (Battery status), B_c (Battery capacity), PET table, CPC (CP composition), EP (Energy prediction model)

1 $E_r = B_s.B_c$

2 Find F_{CP} (i.e, column in PET) such that:

$$\min |E_r - EP(F_{CP}, CPC)|$$

3 Increment or decrement F_{CP} such that:

$$EP(F_{CP}, CPC) \leq E_r$$
$$\nexists F' \; EP(F_{CP}, CPC) < EP(F', CPC) \leq E_r$$

4 **While** workload is running

5 $PC_s \leftarrow$ collect and standardize PCs

6 Find phase i that has the closest C_{CPi} to PC_s

7 $(Fc, Fg) \leftarrow F_{CPi}$

8 **end while**

table (PET). The PET shown in Table II contains the CPU-GPU frequencies per CP that correspond to 5% decrements of the performance. It was obtained offline by running various benchmarks at different frequencies while keeping track of the performance.

The PET table is used within the runtime frequency optimization algorithm given in Algorithm 2 to identify the optimal DVFS settings. Algorithm 2 consists of three main steps.

a) **First step (lines 1-2):** This step aims to make a first estimation about the optimal DVFS settings per CP. It consists in going through the columns of the PET Table to find the column that has the closest energy to the remaining energy.

b) **Second step (line 3):** This step consists of doing frequency increments or decrements, to further minimize the difference between the expected energy consumption and the remaining battery energy. This is achieved by finding the setting F_{CP} that offers the smaller closest energy consumption to E_r. The procedure consists of choosing a CP and incrementing or decrementing its corresponding CPU and GPU frequency, which was obtained in the first step of the frequency optimization procedure. If from the first step we obtained $E_r > EP(F_{CP}, CPC)$, we increment the frequency, otherwise we decrement. After adjusting the frequency, we predict the new energy and check whether the equation in line 3 is satisfied, otherwise we repeat the same procedure by choosing a different CP. Throughout this procedure, the CPs are chosen based on their corresponding value in the CPC vector, such that the CP with smaller values are chosen first. The intuition is that CPs with smaller values are less executed by the user, thus, they allow small changes in the energy and performance.

c) **Third step (lines 4-7):** The optimal F_{CP} vector identified in the previous steps is given as input to the Workload-aware governor to assign the DVFS settings based on the current CP. As shown in line 5 to 7 of Algorithm 2, the PCs are continuously collected, and the classification is performed by measuring the Euclidean distance between the normalized PCs of the running workload, and the precomputed centroids of the chosen CPs from the offline analysis. The F_{CP} vector identified in the previous steps of Algorithm 2 contains frequency targets per CP , and is used after each classification to set the CPU and GPU frequencies to level of the corresponding CP.

TABLE II
DVFS SETTINGS (MHZ) OF THE PHASE-AWARE PERFORMANCE-ENERGY TRADE-OFF TABLE (PET)

	Performance 100%	Performance 95%	Performance 90%
CP1 (CPU, GPU)	(2112, 256)	(1958, 256)	(1804, 256)
CP2 (CPU, GPU)	(2342, 256)	(2208, 256)	(2112, 256)
CP3 (CPU, GPU)	(2361, 256)	(2265, 256)	(1958, 256)
CP4 (CPU, GPU)	(2342, 710)	(2208, 670)	(2035, 596)
CP5 (CPU, GPU)	(2342, 256)	(2112, 256)	(1881, 256)
CP6 (CPU, GPU)	(2342, 256)	(1958, 256)	(1804, 256)

V. EXPERIMENTS AND RESULTS

A. Experimental setup

All the experiments are performed on an Android-based (Oreo version 8.0.0) Google Pixel 2 XL phone. The CPU and GPU performance are measured based on the multi-core score of Geekbench 4.3.1, and 3DMark scores. The Geekbench score is based on the run-time of each workload, the lower the run-time the higher the score is. The 3DMark scores are based on the frames per second (FPS), the higher the FPS the higher the score. Geekbench and 3DMark were chosen because they are by far the most widely used benchmarks in the mobile market by both companies and end-users. We used the Monsoon HV power monitor AAA10F to measure the power. The proposed work is implemented in C and runs on the actual smartphone. The management technique takes a DVFS decision each 20 ms, it has a minor memory footprint, and its computational overhead is less than 1%. Additionally, all the presented results already incorporate all computation overhead.

B. Results

Energy prediction model: In order to validate the prediction model, we collect the training data by running various workloads at different frequency levels. After performing the regression analysis, the accuracy of the energy prediction model was measured as compared to the actual energy values through several runs of workloads. The average percentage error across all runs is 7.3%.

We compare against state of the art battery lifetime management techniques described in Section II, namely Powersave [13], BUSQ1 and BUSQ3 [2].

Powersave [13]: The used mobile platform allows a maximum CPU frequency of 2.45 GHz, and 710 MHz on the GPU side. When 20% of the capacity is reached, the maximum allowed CPU and GPU frequencies are decreased by 50% to 1.26 GHz and 342 MHz, respectively.

BUSQ1Ad [2]: In the case of a usage pattern that includes several stand-by phases, BUSQ1 ends up over-provisioning the available battery capacity, which results in a poor performance

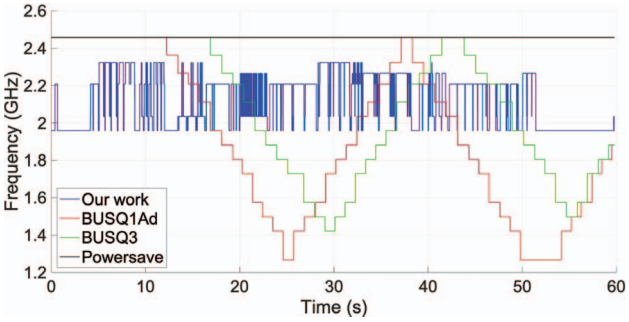

Fig. 3. CPU frequency traces while running Geekbench

TABLE III
CPU EVALUATION OF THE PROPOSED TECHNIQUE USING GEEKBENCH4 (HIGHER SCORES MEAN BETTER PERFORMANCE)

	Crypto	Integer	Float	Memory	Total	QoS Variation	Battery Life (h)
Our work	5148 -3.7%	7704 5.4%	5344 6.1%	2476 -1.5%	5823 6.1%	1.9%	8.6 2.5%
Powersave [13]	4944 -7.5%	7501 2.6%	5064 6.4%	2426 -3.5%	5627 2.5%	62%	8.04 -4.2%
BUSQ1Ad [2]	5022 -6.1%	7200 -1.5%	4655 -2.2%	2218 -11.8%	5331 -2.9%	23%	8.49 1.2%
BUSQ3 [2]	5348 0%	7310 0%	4761 0%	2514 0%	5488 0%	14%	8.39 0%

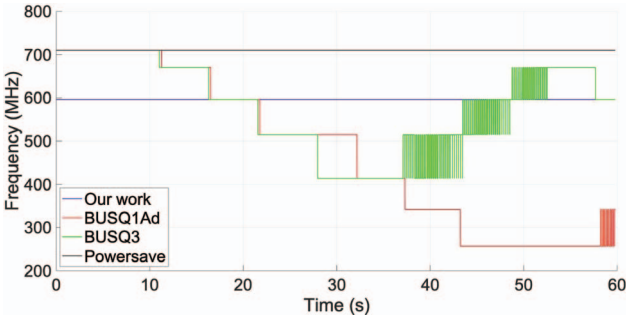

Fig. 4. GPU frequency traces while running 3DMark

TABLE IV
CPU-GPU EVALUATION OF THE PROPOSED TECHNIQUE USING 3DMARK (HIGHER SCORES MEAN BETTER PERFORMANCE)

	Graphics score	Physics score	Total score	QoS Variation	Battery Life (h)
Our work	4780 (9.4%)	2718 (15.8%)	4089 (11.9%)	3.6%	8.53 (0%)
Powersave [13]	4415 (1.5%)	2484 (5.8%)	3760 (2.9%)	50%	8.85 (3.4%)
BUSQ1Ad [2]	4578 (5.3%)	2624 11.8%)	3882 (6.3%)	38%	8.2 (-4.2%)
BUSQ3 [2]	4348 (0%)	2384 (0%)	3653 (0%)	55.4%	8.56 (0%)

and a higher battery lifetime than required. For the purpose of a fair comparison, we evaluate our approach against a modified version of BUSQ1, referred to as BUSQ1Ad. The only difference as compared to BUSQ1, is that the linear discharge profile is steeper. Hence, the technique gets better performance, while closely meeting the user target lifetime.

BUSQ3 [2]: Rather than using a moving average to define the discharge profile for BUSQ3, we provide the exact discharge profile as input.

In order to perform an extensive evaluation, using two different benchmarks, we define two different user usage patterns[1]:

User Usage Pattern 1:

- $[0s, 500s]$ Geekbench multi-core (CPU)
- $[500s, 700s]$ Stand-by period
- $[700s, 1200s]$ Geekbench multi-core (CPU)
- $[1200s, 2400s]$ Stand-by period

Figure 3 shows a portion of the CPU frequency traces of the implemented techniques. The frequency of BUSQ1Ad and BUSQ3 [2] starts at 2.457 GHz, afterwards their frequency is decremented until it reaches 1.267 GHz and 1.42 GHz, respectively. The frequency trace of Powersave [13] also starts at 2.457 GHz, but then as the battery capacity reaches 20% later in the experiment, the frequency is set to 1.267 GHz.

In the other hand, the energy prediction model allowed the proposed work to predict the optimal set of frequencies that can be maintained throughout the whole experiment. Figure 3 shows that the CPU frequency of the proposed work varies between 2.26 GHz and 1.95 GHz, by switching between 5 frequency levels, which means that the shown workload contains 5 CP phases. As shown in Table III, this leads to a better performance, reflected through a higher score, less QoS variation and a longer battery lifetime. As compared to BUSQ3 [2], the proposed work is showing an improvement of 5.4% and 6.1% on the Integer and Floating-Point sections, respectively. The total performance improvement is 6.1% with 2.5% improvement in the battery lifetime, as compared to BUSQ3. The QoS variation, which represents the percentage difference between the best and the worst score throughout the experiment, shows that the proposed work has only 1.88% performance variation, which is 7x less performance variation as compared BUSQ3.

User Usage Pattern 2:

- $[0s, 950s]$ 3DMark Sling Shot (CPU-GPU)
- $[950s, 1550s]$ Stand-by period
- $[1550s, 2500s]$ 3DMark Sling Shot (CPU-GPU)
- $[2500s, 5300s]$ Stand-by period

Figure 4 shows a portion of the GPU frequency traces of the implemented techniques. The frequency of BUSQ1Ad and BUSQ3 [2] start at 710 MHz, then it is decremented until it reaches 257 MHz and 414 MHz, respectively. For Powersave the frequency starts at 710 MHz, but then as the battery capacity reaches 20% later in the experiment, it is set to 342 MHz.

In the other hand, the GPU frequency of the proposed work in Figure 4 is stable at 596 MHz. The 60 seconds portion showed in the figure is a pure GPU workload phase, corresponding to CP4, whose optimal frequency was predicted to be 596 MHz. Figure 5 shows the battery discharge profile of our work as compared to the other techniques while running the usage pattern 2. The figure shows that the discharge profile of Powersave [13] drowns pretty fast in the first 1700 seconds,

[1]Based on the energy consumption during these two patterns, we calculate the expected battery lifetime given the battery capacity of the used platform.

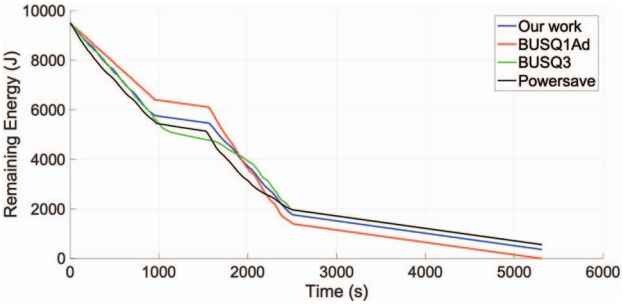

Fig. 5. Discharge profile

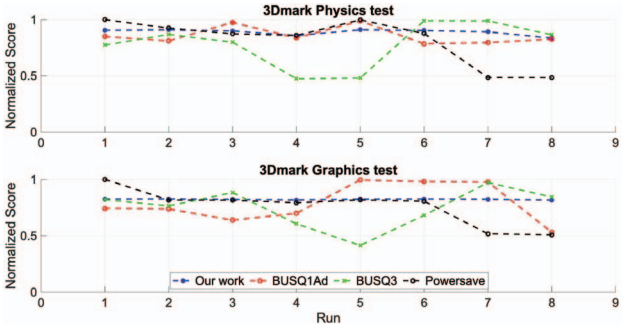

Fig. 6. QoS variation using 3DMark

then the energy consumption slows down significantly. In contrast to Powersave, in the first 1550 seconds, BUSQ1Ad is over-conservative in energy consumption, then the remaining energy drowns rapidly. On the other hand, our technique maintains a stable energy consumption profile. As shown in Figure 5, the energy profile of the proposed work is at all times confined between the under-conservative and the over-conservative discharge profiles. Hence, providing an optimal and seamless performance, while meeting the target lifetime. This is reflected through Table IV that shows a better performance, less QoS variation and a longer battery lifetime. As compared to BUSQ3 [2], our work is showing an improvement of 9.4% and 15.8% on the graphics and physics sections of 3DMark, respectively, with a total performance improvement of 11.9%.

Additionally, our work has only 3.6% performance variation, which is 15× less performance variation as compared to BUSQ3. This is demonstrated through Figure 6, that shows the normalized physics and graphics scores of 8 runs of 3DMark, divided by a 10 minutes break after the fourth run. The figure shows that the proposed work offers a seamless and stable performance on CPU and GPU side, as compared to the other techniques, whose normalized score varies between 0.4 and 1.

VI. Conclusion

This paper investigated a workload- and user-aware battery lifetime management technique for Mobile SoCs. The proposed technique manages both the CPU and the GPU to maximize performance while meeting a target battery lifetime. During an offline analysis we design a Workload-aware governor by identifying a set of canonical workload phases using cluster analysis. We build an energy prediction model based on the canonical phase composition of the user usage history. During runtime, a frequency optimization procedure uses the energy prediction model to find the optimal set of frequencies per canonical phase. Finally, the Workload-aware governor identifies the phase of the current workload and uses the frequencies per canonical phase to scale the frequency. The proposed technique achieves up to 11.9% better performance and it decreases the performance variation by 10× while meeting the user desired battery lifetime.

VII. Acknowledgement

This research was supported in part by a Samsung GRO award and NSF grant number 1730003.

References

[1] K. Yan, X. Zhang, and X. Fu, "Characterizing, modeling, and improving the qoe of mobile devices with low battery level," in *International Symposium on Microarchitecture (MICRO)*. IEEE, 2015, pp. 713–724.

[2] W. Lee, R. Panda, D. Sunwoo, J. Joao, A. Gerstlauer, and L. K. John, "Buqs: battery-and user-aware qos scaling for interactive mobile devices," in *23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2018, pp. 64–69.

[3] B. K. Donohoo, C. Ohlsen, and S. Pasricha, "Aura: An application and user interaction aware middleware framework for energy optimization in mobile devices," in *29th International Conference on Computer Design (ICCD)*. IEEE, 2011, pp. 168–174.

[4] X. Li, G. Yan, Y. Han, and X. Li, "Smartcap: user experience-oriented power adaptation for smartphone's application processor," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2013, pp. 57–60.

[5] N. Banerjee, A. Rahmati, M. D. Corner, S. Rollins, and L. Zhong, "Users and batteries: interactions and adaptive energy management in mobile systems," in *International conference on ubiquitous computing*. Springer, 2007, pp. 217–234.

[6] L. Yang, R. P. Dick, G. Memik, and P. Dinda, "Happe: Human and application-driven frequency scaling for processor power efficiency," *Transactions on mobile computing*, vol. 12, no. 8, pp. 1546–1557, 2012.

[7] J.-G. Park, C.-Y. Hsieh, N. Dutt, and S.-S. Lim, "Cooperative cpu-gpu frequency capping (co-cap) for energy efficient mobile gaming," *UCI Center for Embedded and Cyber-physical Systems TR*, 2015.

[8] J.-G. Park, N. Dutt, H. Kim, and S.-S. Lim, "Hicap: Hierarchical fsm-based dynamic integrated cpu-gpu frequency capping governor for energy-efficient mobile gaming," in *Proceedings of the International Symposium on Low Power Electronics and Design*, 2016, pp. 218–223.

[9] Y. Choi, S. Park, and H. Cha, "Graphics-aware power governing for mobile devices," in *Proceedings of the 17th Annual International Conference on Mobile Systems, Applications, and Services*, 2019, pp. 469–481.

[10] C.-Y. Hsieh, J.-G. Park, N. Dutt, and S.-S. Lim, "Memory-aware cooperative cpu-gpu dvfs governor for mobile games," in *13th Symposium on Embedded Systems For Real-time Multimedia (ESTIMedia)*. IEEE, 2015, pp. 1–8.

[11] N. Peters, S. Park, D. Clifford, S. Kyostila, R. Mcllroy, B. Meurer, H. Payer, and S. Chakraborty, "Phase-aware web browser power management on hmp platforms," in *Proceedings of the International Conference on Supercomputing*, 2018, pp. 274–283.

[12] F. Xu, S. Yang, Z. Zhou, and J. Rao, "ebrowser: Making human-mobile web interactions energy efficient with event rate learning," in *38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2018, pp. 523–533.

[13] U. Goel, S. Ludin, and M. Steiner, "Web performance with android's battery-saver mode," *arXiv preprint arXiv:2003.06477*, 2020.

[14] E. Poyraz and G. Memik, "Using built-in sensors to predict and utilize user satisfaction for cpu settings on smartphones," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 3, no. 1, pp. 1–25, 2019.

[15] S. Z. Selim and M. A. Ismail, "K-means-type algorithms: A generalized convergence theorem and characterization of local optimality," *Transactions on pattern analysis and machine intelligence*, no. 1, pp. 81–87, 1984.