2017 IEEE 35th International Conference on Computer Design

Estimating the Limits of CPU Power Management for Mobile Games

Benedikt Dietich, Nadja Peters, Sangyoung Park, Samarjit Chakraborty
Technical University of Munich,
{benedikt.dietrich, nadja.peters, sangyoung.park, samarjit}@tum.de

***Abstract*—Games are one of the most popular and at the same time most computation intensive and energy consuming class of applications on mobile devices like smartphones and tablets. Dynamic voltage and frequency scaling (DVFS) is a common technique for reducing the processing power. However, highly variable and non-deterministic workload characteristics of mobile games mandate sophisticated workload prediction models to predict low-utilization phases of games during which the processor's frequency can be decreased to save energy. While prior works exhibit significant improvements, one main question is left open: How large is the gap between the developed techniques and the theoretically *optimal* power manager, i.e., a power manager which exactly knows the future workload and, hence, can select the optimal sequence of frequencies that minimizes the power consumption under given timing constraints. In this paper, we discuss that estimating the savings from such an optimal power manager is non-trivial due to the non-deterministic nature of games and the underlying system. In order to address this, we suggest a statistical model of the optimal power manager using which we estimate the potential savings of popular closed-source games. The results of our work have several implications: We reveal a significant gap between savings obtained from recently proposed game power managers and the theoretically optimum savings (up to 54.4 % energy savings are possible). Our work strongly motivates future research endeavors to minimize the gap between the optimum and the existing power managers.**

I. Introduction

Power consumption on mobile platforms is a major design concern. Although a vast amount of work has been put into optimizing the energy consumed by displays [2], memory [11], or the operating system in general [7], [12], there is only little work on application-specific power management, e.g., for games. On mobile devices, games account for a popular class of applications. According to [9], users spend around 32 % of time they use tablets and smartphones on gaming.

Power management for games is a challenging task due to the characteristics of the gaming workload. First, the gaming workload depends on the nature of the game that is being played. A first-person shooter exhibits a completely different workload behavior than a puzzle game. While the background in an endless running game might be constantly changing and needs re-rendering, the background of a card game will stay the same for a long period of time. Second, the gaming workload depends on the game state. For example, recent work [5] has shown that players spend a significant amount of time not only on actually playing the game, but as well in menu states or in loading states during which game content is fetched. Third, computation workload of games is non-deterministic and depends significantly on the interaction with the user. Unlike video rendering, which guarantees deterministic workload over multiple repetitions, gaming workloads are inherently user-interactive, and hence, difficult to reproduce. The frame rendering workload can be drastically different depending on the user's decisions. Considering all this, one of the major concerns in game power management has been to accurately estimate the workload, and apply power management techniques accordingly.

In the past, many successful attempts based on workload predictors have been made to reduce the CPU power consumption for games. The key to successful power management is to achieve remarkable power savings without noticeable performance degradations. A good gaming experience and smooth animations are provided once the frame rate is large enough, e.g., $\geq$30 FPS [3], [4]. Power management techniques [5], [6], [20], [18], based on the frame-wise prediction of workloads, performed dynamic voltage and frequency scaling (DVFS) to determine the minimum operating frequency at the start of each frame that can meet the minimum frames-per-second (FPS) constraint. These techniques have shown decent performance in saving power. Nevertheless, we do not yet know how much further reduction in power consumption could be achieved. In this paper, we quantify the theoretical upper bound for improvement.

There are multiple hardware components in a smartphone, which contribute to the total power consumption. Among them, we are interested in the power consumption of the CPU, which is a major power consumer as shown in Table I. The table gives the measured power consumption of a Samsung Galaxy Nexus for different applications. The phone contains the same processor as the PandaBoard that we used in further experiments. It has been modified to allow detailed measurements of the total and the processor's power consumption. While the CPU's power consumption only amounts to 19.4 % of the total power when running apps like Facebook, it can constitute up to 28.9 % when complex games like Shadowgun from Madfinger Games are played. The rest is consumed mainly by the display, network, and GPU. However, all the components except the CPU and GPU are irrelevant to the frame processing time, therefore, these are out of scope of this paper.

We propose a method to quantify the theoretical upper bound of power savings by an optimal CPU power manager. We rely on a statistical workload model rather than

1063-6404/17 $31.00 © 2017 IEEE
DOI 10.1109/ICCD.2017.10

TABLE I: Power consumption for different applications run on a Samsung Galaxy Nexus.

Application	CPU [mW]	Total [mW]	CPU power consumption [%]
Facebook	237.8	1226.0	19.4
Temple Run	377.3	1605.9	23.5
Jetpack Joyride	376.3	1481.6	25.4
Cut the Rope	349.2	1449.2	24.1
Shadowgun	467.9	1615.6	28.9

the workload profiles from individual game plays to tackle the non-deterministic and non-reproducible nature of gaming workload. The contributions of this work are: (i) We show that the non-deterministic nature of games and the underlying system highly complicates deriving a model that allows estimating the optimal power consumption for an individual game frame. (ii) We present a statistical model that allows estimating the optimal savings of previous game plays using recorded workload statistics. (iii) We, for the first time, compare the power savings of state-of-the-art autoregressive (AR) power managers to the savings of a theoretical, *optimal* power manager for closed-source games. (iv) Even though the AR-based governor outperforms Android's default governor, we show that there is still a significant gap to the power savings of the optimal power manager, strongly motivating future research towards closing this gap.

We emphasize that the purpose of the proposed statistical approach is ***not to propose an actually performing power management technique*** as numerous other works have presented, but to assess the theoretical upper bound of the power management techniques, and to motivate further research in this domain. To the best of our knowledge, this is the first work to address the issue by adopting a statistical approach.

The remainder of this work is structured as follows: Section II gives an overview over related work in the area of game power management. Section III introduces existing game power management techniques, while Section IV explains why we chose a statistical approach over a frame-based approach to model the optimal power manager. Section V describes the hardware and software setup used for this study. In Section VI, we present the experimental results and compare the performance of existing techniques to the optimal power manager. We conclude the work in Section VII.

II. Related Work

In this section, we introduce recent work that has focused on game power management for Android. However, none of the works has tried to find the theoretical upper limit for DVFS by constructing a theoretically optimal game workload predictor, in the following referred to as *oracle* predictor.

Similar to videos, games are computed on a frame basis. While recent games often target 60 FPS, it has been shown that a good user experience can be obtained even with 30 FPS [4], [3], [20]. Lowering the frame rate is not only beneficial in terms of CPU but also in terms of GPU power consumption, as was pointed out in [14], [23]. In order to apply DVFS, it is crucial to estimate the frame workload accurately. However, the workload fluctuates considerably from frame to frame. This motivated to develop sophisticated workload predictors [6], [5], [20], which enabled DVFS-based power management, which outperforms the default Android governors. For example, in [6], it has been shown that the workload of neighboring frames correlates, which can be exploited for fine-grained power management. Nevertheless, the frame workload prediction accuracy is never 100%, and often under- or over-estimated. This leads to frame drops and sub-optimal power savings. Yet, it is difficult to analyze how close to optimal the state-of-the-art techniques are, and therefore, it is hard to judge whether more sophisticated techniques such as non-linear workload predictors are worth investigating.

With the emergence of multi-core CPUs and even heterogeneous multi-processing platforms that combine low-power and high-performance CPUs on a single chip, the multi-threading potential of games has also been subject of recent work [20], [18], [16]. The authors in [20] show that the workload of the threads of a game exhibit different patterns. [18] presents a detailed investigation about the actual workload distribution among the threads. Both works have shown significant improvement in power consumption by performing allocation and DVFS at the same time. However, no work has investigated the theoretical lower bound in terms of power consumption compared to the existing work.

Besides CPUs, GPUs have become an integral part of modern system-on-chips (SOCs), recently [13], [17], [1]. There exists an own line of research for the non-trivial relationship between CPU and GPU workload. The authors in [17] show that some games are more CPU bound while others have a performance bottleneck at the GPU side. There are also games with hybrid CPU-GPU dependencies. However, estimating the theoretical bound of CPU DVFS is already a difficult problem as we will see in the following sections, let alone the effect of GPUs. Therefore, we first focus on the theoretical bound of CPU DVFS techniques, and leave the analysis of CPU-GPU coordinated power management as a future work.

There has been decent work on game power management achieving high savings for CPU and GPU power consumption. Still, it remains unknown how much potential exists for future research on this topic. In this paper, we assess this gap by constructing a power model based on a multi-core platform and evaluating the theoretical minimum power consumption for a set of games using a statistical model of the workload.

III. Game Power Management Techniques

Before we introduce our method to accurately approximate the oracle game workload predictor, we provide background on existing frame-based power management techniques.

A. Frame-rate selection

The most important factor affecting the power consumption is the frame-rate. Most recent games target 60 FPS, although a good user experience can be already achieved at 30 FPS [3],

[4] The larger the frame rate, the more frames have to be rendered. To guarantee a specific frame rate, the CPU's frequency has to be chosen at the *beginning* of each frame such that the processing of the frame finishes within $1/FPS$, where FPS is the target frame rate. For each frame i, the minimum possible frequency $f[i]$ should be used that just avoids violating the deadline $1/FPS$ to optimize the power consumption. Recent work shows that not only the CPU but also the GPU power consumption can be significantly reduced by lowering the frame rate [14], [23].

Further, each game has specific game *states* such as the loading, game menu and the actual gaming state. Each of these states has different requirements and workload characteristics, e.g., the level selection menu can be rendered at a lower frame rate compared to the interactive gaming state. The state detection described in [5] is based on textures used during a frame. Each game has specific textures to render, e.g., menu buttons in the menu phase, a static picture during the loading state and main character textures in the gaming state. Based on work in [5], we not only provide power management limits for different target frame rates, but as well for individual game states as we show in Section VI.

B. Workload prediction

As mentioned before, the gaming workload is hard to predict and poses a challenge for the existing power management techniques, such as the Android default governors: Within different game genres and even within different games of the same genre, the workload is hardly repetitive. Moreover, the underlying operating system does not behave deterministically for multiple runs of the same game. Furthermore, the workload also depends on the current game state, as depicted in [5]. All of these factors make the prediction of the workload difficult. We will show in Section IV-A that the workload characteristics of games motivate the need to develop statistical models for estimating the upper bound for power savings.

Although there are works implementing reactive power managers, we focus on prediction-based strategies in our paper. Prediction-based approaches exploit the correlation between closely-timed frames, which exhibit similar workloads for neighboring frames. In [6], it has been shown that autoregressive models provide the sufficient accuracy that guarantees significant power savings while maintaining a stable frame rate. An autoregressive moving average based predictor was derived that predicts the workload of future frames based on a linear combination of workloads of previous frames.

While AR-model based prediction significantly outperformed Android's default governor, it is not clear if more complex workload predictors, e.g., non-linear models, would provide a significantly better performance or not. Our results show that there is a significant gap between power savings obtained using our newly introduced oracle predictor for modern closed source games, and those obtained using state-of-the-art power management techniques from the literature.

At this point, we would again like to emphasize that the goal of this paper is *not* to propose a new power manager for gaming applications. In this work, we show for three popular games, each from a different genre, which CPU power savings are possible for the full range of viable target frame rates.

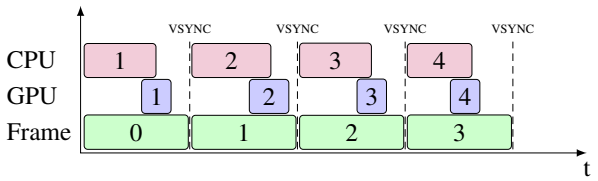

Fig. 1: Typical frame timing.

TABLE II: Average frame duration at different processing frequencies and workloads in ratio to the average frame processing time at the highest processing frequency (1.2 GHz).

Workload	350 MHz	700 MHz	920 MHz	1.2 GHz
Temple Run	2.25	1.31	1.08	1.00
Cut the Rope	1.77	1.26	1.14	1.00
Jetpack Joyride	2.69	1.53	1.01	1.00
CPU-bound	3.43	1.71	1.30	1.00

IV. Optimal Power Manager Model

The oracle power manager is based on the power and workload model described in this section. First, we describe that a frame-based workload model approach is not suitable for our power manager due to the non-deterministic nature of the gaming workload. Then, we derive a statistical workload model and approximate the power consumption of the CPU, including the DVFS overhead.

A. Frame-Based Model

In the case of videos, the frame content and the corresponding workload for de/encoding is deterministic. Hence, the optimal sequence of frequencies can be determined for all frames of the clip [19]. For games, on the contrary, the content depends on the user actions as well as the time it takes to process each frame: At the beginning of each frame processing, a game typically computes the time Δt that has passed since the last frame processing beginning. Δt is then used to update object positions, the physics engine and the artificial intelligence (AI). Hence, what exactly can be computed during a frame, heavily depends on Δt. If a game is played at different frequencies, the processing time of individual frames, Δt and consequently the game content will significantly differ from the previous run. This prevents a brute-force solution that is viable in case of video de/encoding.

Starting from Android 4.1, Project Butter [8] introduced the synchronization between the frame processing and the vertical synchronization signal (VSYNC) of the display. As depicted in Figure 1, the processing of the next frame i always starts with the arrival of a VSYNC signal. Due to this synchronization, in the ideal case, Δt always equals $1/r_{VSYNC}$, where r_{VSYNC} is the refresh rate of the display, typically 60 Hz. Based on this observation, we investigated the possibility to derive the

optimal sequence from a single, fixed frequency recording of a game play. We instrumented the operating system (see Section V) and recorded the number of cycles $c[i]$ required for each frame of a game play at the highest processing frequency. A simplified assumption is that the processing time linearly scales with the CPU's processing frequency. Hence, the optimal sequence of processing frequencies can be chosen by selecting the smallest frequency $f[i] \in \mathcal{F}$ for each frame, where $\mathcal{F}$ is the set of all available frequencies, that guarantees

$$\frac{c[i]}{f[i]} < \frac{1}{r_{VSYNC}}. \quad (1)$$

Assuming that we never violate this deadline, it should be possible to determine the optimal frequency for each frame. However, the linear relationship in Equation 1 typically does not hold true and in most cases is highly pessimistic. As presented in [21], the frame computation time is amongst others composed of the CPU computation time and the time the CPU waits for GPU, memory and I/O given by the following:

$$t_{frame}[i] = \frac{c_{CPU}}{f_{CPU}} + \frac{c_{Mem}}{f_{Mem}} + \frac{c_{IO}}{f_{IO}} + \frac{c_{GPU}}{f_{GPU}} + \ldots,$$

where c_{CPU}, c_{Mem}, c_{IO} and c_{GPU} are the workload in cycles and f_{CPU}, f_{Mem}, f_{IO} and f_{GPU} are the CPU, memory, I/O and GPU frequency respectively. Increasing the processor's frequency will only decrease the time it takes to process the CPU workload, given by the term c_{CPU}/f_{CPU} without affecting the times required for memory accesses, I/O operations and GPU processing. The linear Equation 1 only holds true for purely CPU-bound workloads. The relative performance is defined by

$$s = \frac{f_{CPU}}{f'_{CPU}} \times \frac{c'[i+1]}{c[i+1]} = \frac{t'[i+1]}{t[i+1]}$$

and describes by how much the processing time of a task is prolonged if CPU frequency f' is used instead of f. As can be seen in Table II, the relative performance of games differs significantly from the relative performance of a purely CPU-bound workload. While the pessimistic purely CPU-bound model from Equation 1 would for example assume a slowdown of 3.43 if the frequency is scaled from 1.2 GHz to 350 GHz, in average the processing is only slowed down by a factor of 1.77 in the case of Cut the Rope which is one of the games that we investigated in our study. Hence, a power manager that has predicted the next frame's workload and has to choose among the available frequencies can scale the frequency more aggressively if it as well considers the relative performance. Table II only provides the average factor by which the duration of a frame is being prolonged if a smaller than the highest frequency is used. In reality, s will highly vary from frame to frame, since some frames might be GPU or memory bound, i.e., the CPU has to wait for a non-negligible amount of time for the GPU or memory, whereas other frames might be highly CPU-bound.

Non-deterministic nature and reproducibility of gaming workloads: In order to correctly evaluate the relationship between the CPU operating frequency and the workload considering these effects, it is required to replay the same workload at different CPU processing frequencies and measure the CPU workload. We have developed our own gaming benchmark, which animates a knight walking through a 3D landscape, to derive more accurate models that include I/O and GPU waiting times. The input events to this benchmark are generated automatically to guarantee the reproducibility of the workload. Figure 2 shows the frame-based workload of two consecutive benchmark runs using a constant CPU processing frequency. Even though the benchmark has always been started exactly one second after Android has been booted, significant differences can be observed already after 100 frames. Computations of the Android OS and other processes running in Android, which are not necessarily synchronized with the game, heavily disturb the reproducibility. Clearly, variations to this extent do not allow us to accurately tune a model describing the relative performance factor.

Due to these difficulties, we propose to use a statistical model for calculating the theoretical upper bound of a power management technique in the next section. This model does not allow determining the exact optimal sequence of frequencies, but provides an estimation of the optimal power manager's performance and hence is sufficient for our purposes.

B. Statistical Performance Model

We seek the minimum possible power consumption that can be obtained when, for each frame, the minimum possible frequency is used (considering the target frame rate). In Section IV-C, we show that the power consumption can be determined, once it is known to what percentage $P(f = f_x)$ each processing frequency $f_x \in \mathcal{F}$ has to be used for processing a game play under frame rate constraints. In the following, we describe how to determine these percentages $P(f = f_x)$ based on game workload recordings.

We have recorded the frame processing times of different game plays using fixed frequencies f_x. From these recordings, we computed probability distributions $P_{f_x}(X = t)$ of the frame processing time. Even though Temple Run randomly generates its game scenarios and hence the content significantly differs between consecutive runs, the two histograms depicted in Figure 3 are nearly identical. This could be observed for all available processing frequencies and games under test (Cut the Rope, Temple Run and Jetpack Joyride) if the games were played long enough. Game plays of 10 minutes turned out to be sufficient.

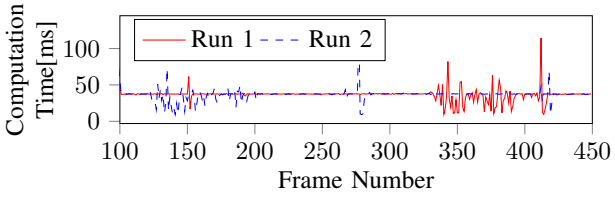

Fig. 2: Workload of two identical 3D Benchmark runs.

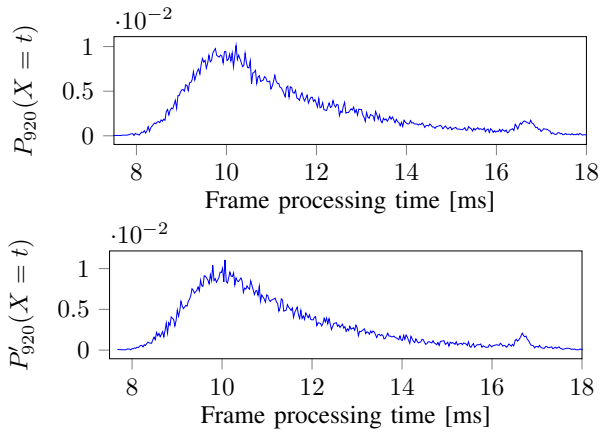

Fig. 3: Workload histogram of two game plays of Temple Run at 920 MHz each.

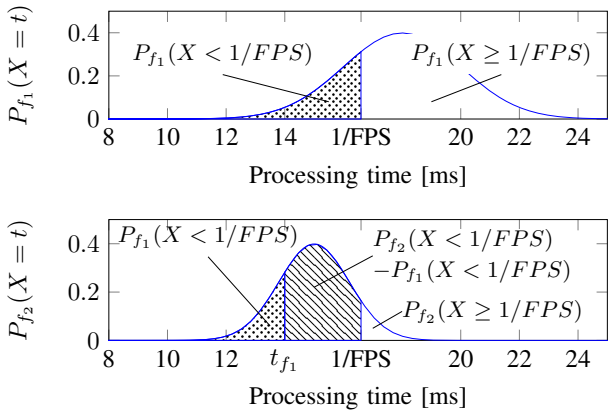

Fig. 4: Probability distribution of frame-based workloads for two different processing frequencies f_1 and f_2.

Figure 4 shows two examples of probability distributions $P_{f_1}(X=t)$ and $P_{f_2}(X=t)$ recorded at the frequencies f_1 and f_2, respectively. Here, f_1 represents the smallest and f_2 the next higher of all available CPU clock frequencies.

The probability $P(f=f_1)$ that a frame can be finished within the deadline $1/FPS$ using frequency f_1 is given by

$$P(f=f_1)=P_{f_1}(X<1/FPS)=\int_0^{\frac{1}{FPS}} P_{f_1}(X=t)\,dX.$$

Further, the percentage of frames that will require a higher frequency than f_1 is given by $P_{f_1}(X\geq 1/FPS)$.
Like for P_{f_1}, the percentage of frames that can be computed in time using f_2 is given by $P_{f_2}(X<1/FPS)$. Frames that can be completed within time using f_1 will certainly finish if f_2 is used since increasing the frequency will not increase the processing time. Hence, the area defined by $P_{f_1}(X<1/FPS)$ is included in $P_{f_2}(X<1/FPS)$. Consequently, the percentage of frames that will require f_2 and fail for f_1 is given by

$$P(f=f_2)=P_{f_2}(X<1/FPS)-P_{f_1}(X<1/FPS).$$

The remaining percentages $P(f=f_3),\ldots,P(f=f_i)$ can be computed accordingly, once the probability distributions of all processing frequencies have been determined. Note, that in Figure 4, the area defined by $P_{f_1}(X<1/FPS)$ is depicted as the leftmost area for visualization purposes. The workload composition and the relative performance factor of the frames determine how this area is distributed within $P_{f_2}(X<1/FPS)$. In the next section, we explain how these probabilities can be used to compute the minimum possible power consumption of the oracle power manager.

C. Power Consumption Model

The average power consumption $\overline{P}$ can be approximated as follows: In addition to the frame processing time $t_x[i]$, we measure the power consumption $\overline{P}_{f_x}[i]$ of each individual frame. Based on these recordings, the average power consumption of frames at frequency f_x is given by

$$\overline{P}_{f_x}=\frac{1}{N}\sum \overline{P}_{f_x}[i].$$

The overall average power consumption can then be approximated by

$$\overline{P}=\sum_{\forall f_i\in F}\overline{P}_{f_i}\cdot P(f=f_i), \tag{2}$$

where $P(f=f_i)$ is the extent to which the frequency f_i has to be used to satisfy the processing deadline $1/FPS$. The accuracy of this approximation are evaluated in Section VI.

Typically, two metrics are used to compare the quality of game power managers: The average power consumption $\overline{P}$ and the percentage of frames D missing their deadline. Based on determined probability distributions, D can be determined, since $D = P_{f_{max}}(X\geq 1/FPS)$, i.e., the percentage of frames that can not be processed in time, even though the highest processing frequency f_{max} is used.

D. DVFS overhead

The overhead for dynamic voltage and frequency scaling has to be considered in terms of the time and energy that the voltage and frequency transition costs. As will be shown in Section VI, the overhead highly depends on the current and the target frequency of the processor. However, using the statistical model presented in Section IV-B it is only possible to estimate the percentage of frames that require a particular frequency and not the exact *order* of the processing frequencies, i.e., the optimal frequency sequence $\mathcal{S}$. Hence, it is not possible to consider the exact scaling overhead. In the worst case, we switch at each frame from the current frequency to the frequency which will cause the largest overhead in terms of switching time. This worst case can be modeled by reducing the deadline of $1/FPS$ by the corresponding scaling time. On the contrary, the best case assumes that only a minimum number of frequency switches occur. As will be shown in Section VI, there is only a small difference between the best and worst case. Before above described models are verified and applied, we will first explain the experimental setup.

TABLE III: Workload characteristics of the tested games.

Game	Workload $[\frac{cycles}{frame}]$	Deviation $\sigma\ [\frac{cycles}{frame}]$	Utilization Min	Utilization Max
Cut the Rope	5.87e+06	6.89e+03	13%	96%
Jetpack Joyride	6.67e+06	8.92e+03	12%	57%
Temple Run	1.07e+07	9.55e+03	24%	91%

V. Experimental Setup

In the following, we describe the hardware platform used and the required software modification in the Android OS and the Kernel source code.

A. Hardware Setup

We used the PandaBoard ES [15] for this study running a Linaro Android 4.3 and 3.2.0 Kernel. This board is based on the same processor as the Samsung Galaxy Nexus, namely an OMAP4460 processor, a dual core 1.2 GHz ARM Cortex A9 Mobile processor from Texas Instruments [22]. Attached to the PandaBoard is a 10" multitouch LCD display. The core of the OMAP4460 processor is the microprocessor unit (MPU) subsystem. It consists of the two Cortex A9 cores, each with a dedicated L1 instruction and data cache, a NEON and a VFPv3 (floating point) unit. The MPU clock domain can be configured to run at 350, 700, 920 and 1200 MHz. The frequency of both cores is scaled equally. We have modified the board to measure the power consumption of the MPU unit, separately.

Although newer architectures are available on the market, this does not impact our contribution. It is true that the presented games will pose only a small workload on newer architectures. However, there are many recent games that will exploit the higher processing power of these architectures. The methodology that we present in this paper - the power model and the oracle predictor - can be applied to newer hardware and software architectures with only slight modifications.

B. Software Setup

All popular Android games are closed-source. Hence, it is not possible to instrument the games directly to acquire information on, e.g., frame timing that is needed to estimate the CPU energy consumption. Instead, we have modified the Android OS to gather workload statistics and power measurements of the closed-source games on a frame-by-frame basis. As done in previous works [5], [20], we have instrumented the Embedded Graphics Library (EGL) [10] and its `eglSwapBuffers()` function for timing measurements. This function is called by the game to mark the end of the current frame. Moreover, we have developed a kernel driver that can read the power consumption and the performance counter values per frame.

C. Game Selection

For this study, we used three popular Android games. Zeptolab's Cut the Rope is a 2D puzzle game. Halfbrick's Jetpack Joyride and TempleRun from Imangi Studio, are endless runner games. Jetpack Joyride is 2D while Temple Run is a 3D game. The workload, the standard deviation and the CPU utilization of the three games at the frequency of 1.2 GHz are given in Table III. Temple Run imposes the highest workload and variation on the CPU. The large utilization range shows that all of the games are amenable to DVFS.

VI. Experimental Results

Before presenting the results from the statistical model, we discuss overheads and validate the power model presented in Section IV.

A. Frequency Scaling Overhead

As described in Section IV-D, the overhead for dynamic voltage and frequency scaling has to be considered in terms of the time that is spent on scaling and the energy that the voltage frequency transition costs. To measure both, the `cpufreq` Kernel driver has been instrumented to toggle general-purpose I/O (GPIO) pins of the PandaBoard at entry and exit points of related functions. The voltage and the logic states of the GPIO Pins were sampled using a National Instruments PXI-6124 card and a sampling rate of 2 MS/s. The frequency transitions were initiated using the `userspace` governor.

We have instrumented three functions in the OMAP-specific frequency scaling Kernel driver to measure the overhead for scaling the voltage, the frequency and the operating system based overhead. The results in terms of energy and time of the corresponding parts are shown in Figure 5. As can be seen, the overhead highly depends on the original and the target frequency. Especially, switching to the lowest available processing frequency, i.e., 350 MHz is expensive in terms of time and energy and can consume up to 873.9 us and 116.3 uJ. The maximum observed variation of the average time and energy overhead was 2.58 % and 4.72 %, respectively.

As discussed, the statistical model is unable to determine the exact overhead. However, the resulting power consumption assuming worst and best case (maximum and minimum number of voltage and frequency transitions) differed at maximum only by 4.93 % for the game Cut the Rope. The percentage of frames missing the deadline at maximum deviated by 0.69 %. Hence, we consider the time and power errors by considering exact DVFS overhead as negligible, and hence do not account for ranges in the following, but instead only provide the worst case results.

Instrumenting Android, performing the workload prediction, logging data and detecting game states come with an additional overhead. As has been shown in [5], these overheads can be neglected compared to the workload of a game frame.

B. Power Model Validation

We have performed the following experiment to evaluate the power model described in Section IV-C. We first played each game using the `userspace` governor at every available frequency and computed the average power consumption $\overline{P}_{f_x}$ based on these recordings. Next, we played each of the three games using the `game state-specific` governor described in Section V and [5] and recorded the frequencies

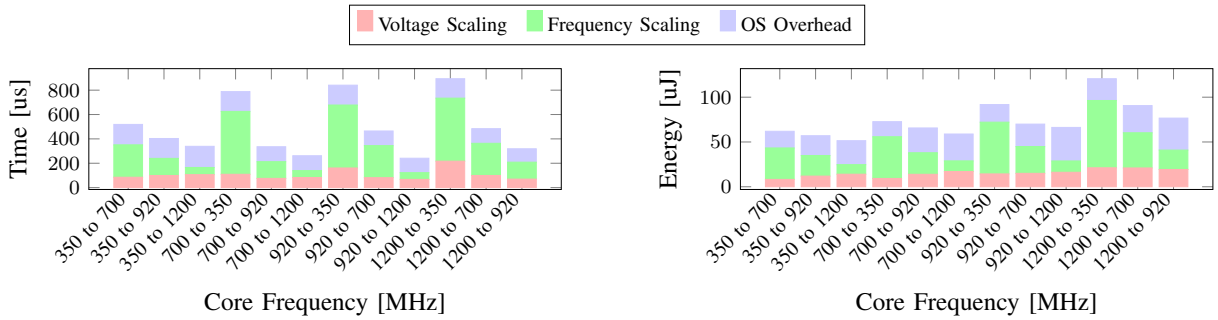

Fig. 5: Frequency scaling overhead in terms of time and energy measured for the TI - OMAP4460.

chosen as well as the real average power consumption of each frame. Based on these recordings, we computed the probability for each frequency to be used during the game play $P(f = f_i)$ and the resulting average power consumption according to Equation 2. The maximum deviation of the estimated power consumption was observed for the game Cut the Rope in the gaming state with 6.01 %. For Jetpack Joyride the maximum deviation was observed in the menu state with 2.96 % and for Temple Run in the gaming state with 1.26 %. We have assumed that these estimation errors are tolerable.

C. Performance of the Optimal Power Manager

As described in Section IV, the optimal power manager chooses for each frame the lowest possible processing frequency that still guarantees the frame computations to complete in time, resulting in a minimum power consumption. The optimal power manager only exists in theory. In practice, the future workload and required processing frequency for a frame have to be predicted. In recent literature [6], it has been shown that the inter-correlation between workloads of consecutive frames can be leveraged to predict the frame workload of future frames, as described in Section III. It was shown that this approach works very well for multiple games and significantly outperforms state-of-the art power managers.

To determine the gap between practice and theoretical optimum, we played each game using the AR-based predictor and evaluated the statistical model described in Section IV-B.

Figure 6 shows the power consumption and percentage of frame deadline misses for the optimal power manager, using the oracle predictor and the results obtained with the AR-based predictor. In addition, Table IV summarizes theoretical possible power savings expressed as percentage compared to the AR-based power manager. Clearly, it can be seen that for high frame rates in particular, there is a significant performance gap of the AR-based predictor to the theoretical optimum. In the case of Temple Run, the power consumption could be reduced by additional 41.8 % during the gaming state and 54.4 % for the menu state (at 50 FPS) if the future workload of frames was exactly known. This considerable potential in terms of power savings points out the need for more research directed to accurate game workload predictors.

TABLE IV: Power consumption gap expressed as percent between the oracle and the AR-model based power management at different frame rates.

Game	State	20	30	40	50
Temple Run	Menu	15.49	25.32	47.62	54.37
	Gaming	0.99	-3.32	6.35	41.74
Jetpack Joyride	Menu	6.87	6.30	24.11	33.17
	Gaming	1.29	-4.23	-6.45	22.80
Cut the Rope	Menu	-1.59	6.19	20.87	18.12
	Gaming	2.06	4.29	22.57	25.74

As can be seen in Table IV, the gap between savings obtained using the oracle power manager and the AR-based power manager is increasing significantly for most games with higher target frame rates. Higher target frame rates will require higher processing frequencies resulting in a significantly higher power consumption. However, at higher processing frequencies the average relative performance increase is less as shown in Table II. The AR-model based predictor is not aware but instead assumes a simple linear scaling according to the pessimistic Equation 1. Consequently, it over-estimates required frequencies particularly at high frame rates resulting in the large gap at high target frame rates.

In some cases, existing techniques outperform the optimal power manager in terms of power consumption, e.g., in the case of Jetpack Joyride (at 40 FPS) the AR-based governor saves 6.45 % more power in the gaming state than the optimal power manager. However, in all of these cases the number of frames missing their deadline is higher compared to the optimal power manager.

Reducing the target frame rate has a considerable impact on the power consumption for both, the AR-based and the optimal power manager. For example, a reduction from 50 to 40 frames per second already reduces the power consumption by 44.9 % for the AR-based predictor in the case of Temple Run, while further reducing the frame rate to 30 frames per second yields power savings of 23.5 %. This finding highly motivates a user study to identify the actually required target frame rates that satisfies the user.

VII. Conclusion

In this work, we have shown the potential for improvement in power savings for mobile games compared to existing

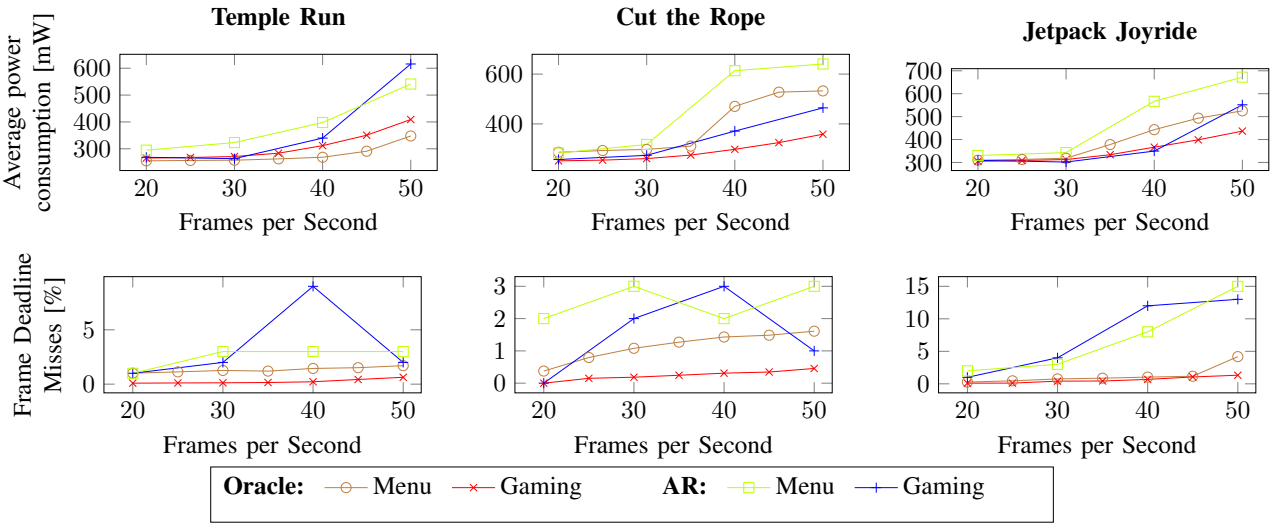

Fig. 6: Average power consumption and frame deadline misses for different target frame rates using the oracle predictor.

power managers. The non-deterministic nature of gaming workload mandates us to take a stochastic approach to evaluate the optimal power savings. Even though this work performs a theoretical study *without* implementing an actual power manager, our work clearly shows that there is significant room for improvement in addition to what the state-of-the-art techniques offer. In the future, we plan to verify our models for more recent architectures, for example heterogeneous multi-processing platforms, which have become very popular, recently. Moreover, we plan to extend our work to include the GPU power consumption into our model. We believe that model-based estimation studies as presented in our work will help to understand where optimization potential is available and will provide motivation to adopt even more sophisticated techniques to further reduce the gaming power consumption.

References

[1] E. Ahmad and B. Shihada. Green smartphone GPUs: Optimizing energy consumption using GPUFreq scaling governors. In *IEEE 11th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, 2015.

[2] B. Anand, K. Thirugnanam, J. Sebastian, P. G. Kannan, A. L. Ananda, M. C. Chan, and R. K. Balan. Adaptive Display Power Management for Mobile Games. In *International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2011.

[3] K. T. Claypool and M. Claypool. On frame rate and player performance in first person shooter games. *Multimedia Systems*, 13(1):3–17, 2007.

[4] M. Claypool, K. pool, and F. Dama. The Effects of Frame Rate and Resolution on Users Playing First Person Shooter Games. In *Multimedia Computing and Networking (MMCN)*, 2006.

[5] B. Dietrich and S. Chakraborty. Lightweight graphics instrumentation for game state-specific power management in Android. *Multimedia Systems*, 20(5):563–578, 2014.

[6] B. Dietrich, D. Goswami, S. Chakraborty, A. Guha, and M. Gries. Time Series Characterization of Gaming Workload for Runtime Power Management. *IEEE Transactions on Computers*, 64(1):260–273, 2015.

[7] B. Donyanavard, T. Mück, S. Sarma, and N. Dutt. Sparta: Runtime task allocation for energy efficient heterogeneous manycores. In *International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2016.

[8] C. Haase and R. Guy. *For Butter or Worse - Smoothing Out Performance in Android UIs (http://youtu.be/Q8m9sHdyXnE)*.

[9] S. Khalaf. Apps Solidify Leadership Six Years into the Mobile Revolution, 2014.

[10] Khronos Group. EGL (www.khronos.org/egl).

[11] H. Kim, N. Agrawal, and C. Ungureanu. Revisiting Storage for Smartphones. *ACM Transactions on Storage*, 8(4):1–25, 2012.

[12] E. Kofman and R. de Simone. A formal approach to the mapping of tasks on an heterogenous multicore, energy-aware architecture. In *ACM/IEEE International Conference on Formal Methods and Models for System Design (MEMOCODE)*, 2016.

[13] X. Ma, Z. Deng, M. Dong, and L. Zhong. Characterizing the Performance and Power Consumption of 3D Mobile Games. *Computer*, 46(4):76–82, 2013.

[14] K. W. Nixon, X. Chen, H. Zhou, Y. Liu, and Y. Chen. Mobile gpu power consumption reduction via dynamic resolution and frame rate scaling. In *6th Workshop on Power-Aware Computing and Systems (HotPower)*, 2014.

[15] Pandaboard.org. Pandaboard ES - System Reference Manual. 2011.

[16] A. Pathania, A. E. Irimiea, A. Prakash, and T. Mitra. Power-performance modelling of mobile gaming workloads on heterogeneous MPSoCs. In *52nd Annual Design Automation Conference (DAC)*, 2015.

[17] A. Pathania, Q. Jiao, A. Prakash, and T. Mitra. Integrated CPU-GPU power management for 3d mobile games. In *51st Annual Design Automation Conference (DAC)*, 2014.

[18] A. Pathania, S. Pagani, M. Shafique, and J. Henkel. Power management for mobile games on asymmetric multi-cores. In *IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, 2015.

[19] T. Pering, T. Burd, and R. Brodersen. The Simulation and Evaluation of Dynamic Voltage Scaling Algorithms. In *International Symposium on Low power electronics and design (ISLPED)*, 1998.

[20] N. Peters, D. Füß, S. Park, and S. Chakraborty. Frame-based and thread-based power management for mobile games on hmp platforms. In *IEEE 34th International Conference on Computer Design (ICCD)*, 2016.

[21] D. Snowdon, S. Petters, and G. Heiser. Accurate On-line Prediction of Processor and Memory Energy Usage under Voltage Scaling. In *International Conference on Embedded Software (EMSOFT)*, 2007.

[22] Texas Instruments. OMAP4460 Multimedia Device, OMAP4460 Datasheet, 2012.

[23] Y. Yan, S. He, Y. Liu, and L. Huang. Optimizing power consumption of mobile games. In *7th Workshop on Power-Aware Computing and Systems (HotPower)*, 2015.