Frame-based and Thread-based Power Management for Mobile Games on HMP Platforms

Nadja Peters, Dominik Füß, Sangyoung Park, Samarjit Chakraborty
Institute for Real-Time Computer Systems, Technical University of Munich, Germany
Email: {nadja.peters, sangyoung.park, samarjit}@tum.de, dominik.fuess@mytum.de

***Abstract*—Games belong to the most popular but power-hungry applications on smartphones. Gaming workloads exhibit highly variable and user-interactive behavior, which makes it hard to predict the workload. Modern MPSoC (multiprocessor system-on-chip) platforms are equipped with heterogeneous multi-processing (HMP) processors comprising performance-oriented and energy-efficiency cores in order to better exploit power-performance trade-offs among different types of applications. To minimize the energy consumption of games on HMP platforms, it is essential to precisely predict the gaming workload and perform joint thread-to-core allocation as well as dynamic voltage and frequency scaling (DVFS).**

In this paper, we propose a frame- and thread-based MPSoC power management strategy for games. We focus on the fact that gaming workload has high temporal correlation among frames and evaluate selected workload predictors on a per-frame basis. Moreover, we find that there are two categories of thread workloads, periodic and aperiodic, and hence, propose to use a hybrid workload predictor. Based on the per-thread predictions, the power manager allocates the threads among the heterogeneous cores in an evenly distributed fashion in order to minimize the operating frequency while keeping the frames-per-second (FPS) constraint. We implement the game power manager as an Android governor on a state-of-the-art platform based on the Exynos5422 SoC, which is also incorporated in the Samsung Galaxy S5 smartphone. Our measurement results show that we save on average 41.9% of energy compared to the Android default governor. Further, we have performed a user study to evaluate the user perception of our governor. The gaming experience was rated between *good* and *very good* for all games.

I. Introduction

Games are one of the most favored, but also one of the most power consuming applications for mobile devices. It is reported that 34% of total mobile time is spent on gaming while 22% is spent on messaging and social networks [1]. The gaming workload is characterized as highly variable, and user-interactive as opposed to other types of mobile applications. These characteristics make it hard for the CPU frequency governor in an Android system to perform appropriate power management, thereby impairing the battery lifetime.

A fine-grained workload estimation has more potential for power reduction than coarse-grained workload estimations. Some works have focused on the short-term temporal correlation in computational workload in gaming and proposed frame-based workload predictors [2], [3], [4]. These works made fine-grained frame-wise workload estimations to perform DVFS, and hence, minimize the power consumption while not violating the performance requirements. The performance requirements of games are usually defined by FPS. The user perceived quality is not affected as long as the FPS value stays above a certain threshold, e.g., 30 FPS [5]. Yet, the frame-wise prediction of gaming workload is a demanding task, as the workload characteristics differ among games and underlying hardware platforms.

This work was supported by the Bavarian Ministry of Economic Affairs and Media, Energy and Technology as part of the EEBatt project.

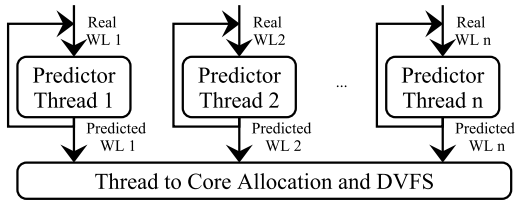

Fig. 1. Power management unit with workload prediction on a per-thread basis, thread to core allocation and DVFS.

To meet the dynamic computing demands, HMP SoCs are being embedded in state-of-the-art mobile devices. They comprise a number of heterogeneous cores to allow computationally demanding threads to run on the performance-oriented cores, and less demanding threads to run on the energy-efficient cores to save energy. This way, the system is able to respond to various computing demands in an energy-efficient manner. However, the problem of minimizing the power consumption of games by means of thread allocation to cores and DVFS without significant degradation in user perception is a demanding task. Previous works [2], [3] have looked into gaming workload as a whole, and performed power management for the whole application. Unlike in these works, multiple threads of one game could be distributed over multiple cores, enabling the cores to run on the minimum frequency that just fulfills the FPS requirement of the user. Consequently, it is essential that the gaming workload is estimated at per-thread and per-frame basis as shown in Figure 1. We observe that there are roughly two categories of thread workloads, periodic and aperiodic. Hence, we propose to make use of a hybrid predictor, a combination of the autocorrelation predictor (ACR), and the weighted moving average predictor (WMA), to handle different workload categories appropriately. However, even if we know the exact values of workload, finding the energy-optimal allocation and frequency is known to be computationally intractable [6]. Thus, a heuristic algorithm has to be developed to perform the actual power management.

978-1-5090-5142-7/16/$31.00 ©2016 IEEE

In this paper, we extend the previous frame-based power management techniques to HMP platforms to propose a frame-based and thread-based, predictive power manager for mobile games. The proposed framework is capable of performing thread allocation and DVFS simultaneously to meet the FPS requirement of the user while minimizing the power consumption. The contributions of this paper are summarized as follows.

- We characterize the gaming thread workloads and develop a thread-based and frame-based hybrid workload predictor to accurately predict the gaming workload.
- The hybrid predictor learns online whether the thread workload is periodic or aperiodic.
- We devise an algorithm to perform thread-to-core allocation and DVFS simultaneously based on the predictions.
- We implement the integrated power manager, which we refer to as *GameOptimized governor* in the following, on an Odroid-XU3 development board [7] and compare it to the default Android governors.
- We perform a user study to evaluate the user perception of our proposed power manager.

The experimental results show that up to 60.0% of energy can be saved for our power manager, while the user perception is considered good.

II. Related Work

Power management for mobile games has drawn attention rather recently as opposed to techniques for other multimedia applications. Prior works regarding fine-grained, i.e., per-frame, power management of mobile gaming is scarce due to the difficulty of precise workload prediction and closed-source nature of commercial games. A PID controller-based workload predictor has been proposed for 3D games, but it requires parameter hand-tuning and is subject to trade-off relationship between the controller stability and prediction accuracy [8]. Despite the hand-tuning, the PID-predictor diverges for different states of the game, e.g., the loading and the gaming phase as those two have significantly different workload behavior. To overcome this issue, auto-regressive (AR) and self-tuning least mean square (LMS) predictors have been proposed [3], [9], which have shown reasonable accuracy in workload prediction. These predictors estimate the workload per frame to perform DVFS in order to reduce power consumption. However, these predictors are only able to predict the frame workload as a whole, while per-thread prediction is required to perform thread-to-core allocation on HMP platforms. Moreover, these predictors are not able to distinguish between different types of thread workloads, e.g., periodic and aperiodic. Hence, their prediction accuracy is not guaranteed for all types of threads.

Power management utilizing both thread-to-core allocation and DVFS on an HMP platform has been proposed in [10]. The work introduces a heuristic online strategy based on a *thread-price* calculated from the CPU utilization. It allocates the threads to cores in a way that allows reducing the CPU frequency, and hence, the power consumption. Another work from the same group has proposed a coordinated CPU-GPU power management that could perform better than independent management for 3D games [11]. This work has been extended to use a regression-based predictor for the impact of DVFS on the game workload in [12]. Although these lines of works achieve decent power reduction compared to the default Android governor, the power management is mostly done in an inherently reactive way to workload changes, and applied on a coarse-grained manner, i.e., roughly once per-second. Compared to these works, our proposed power management offers more potential for power reduction as we are able to better exploit the per-frame workload changes by the predictive nature of our approach.

In [13], the authors introduce a game state and frame rate dependent DVFS policy. The work observes that there exists a bottleneck CPU frequency above which the frame rate does not increase anymore. Moreover, it changes the target frame rate itself based on the game state detection. The CPU frequency is scaled such that it meets the target frame rate.

In summary, we propose a predictive frame- and thread-based CPU power management scheme for games on HMP platforms, which integrates scheduling and DVFS. Compared to previous work, we identify the workload type of each thread, e.g., periodic or aperiodic, and apply the most suitable predictor for that thread. Based on the predicted workload of threads, we allocate the threads to the CPU cores such that the workload is evenly distributed over the cores at the minimum required CPU frequency to meet the target FPS. Rather than relying on the average FPS as a sole performance metric, we also perform a user study to evaluate our power manager.

III. Workload Characterization of Mobile Games

Previous work proved that frame-based workload prediction and DVFS for games achieves significant power savings [3], [8]. Game scene changes, e.g., the appearance of a new enemy, usually occur suddenly and last within a range of seconds. Hence, subsequent game frames inherit similar workload, which can be exploited for workload prediction-based power management. Also, mobile games are becoming more and more multi-threaded. Therefore, it is essential to understand the per-frame and per-thread workload characteristics of games to perform the proposed fine-grained power management.

We observe that there are two types of threads in a typical gaming process, periodic and aperiodic. Figures 2 and 3 show the workload over time for the two categories of threads. The workload was measured on an HMP platform based on the Exynos5422 processor running an Android operating system that will be described later. Figure 2 depicts a sample thread workload of the game Asphalt. The workload is aperiodic, but shows high temporal correlation among adjacent frames. Figure 3 shows the workload of a periodic thread in the game GTA III. There is not much temporal correlation among adjacent frames, but the workload is invoked every second. Once we understand the pattern, it becomes easier to predict the thread workload. The AR predictor used in [3] performs decently for the aperiodic category of threads, but it is not so efficient in predicting the periodic threads. This observation is the

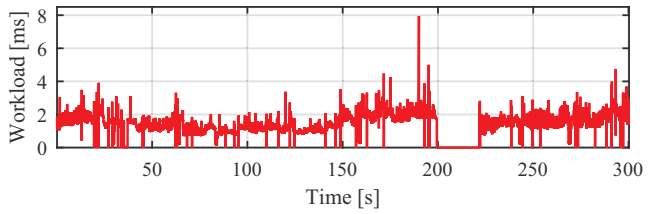

Fig. 2. Measured non-periodic workload of one thread from the game Asphalt.

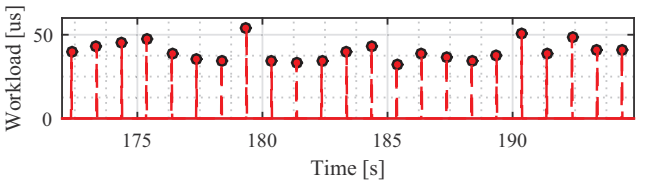

Fig. 3. Measured thread workload from the game GTA III which is reoccurring periodically. The workload is zero except for the indicated circles.

key to our proposed hybrid WMA predictor that successfully estimates workload for both categories of threads.

Another thing to note is the performance requirement of games, usually measured in FPS. Most modern games target a frame rate of 60 FPS to guarantee a good gaming experience for the user. This means that a game frame is computed within 16.7 ms. Research has shown that lower frame rates of 30 FPS are barely noticeable for the user for shooter games [5]. Overachieving this goal requires a great amount of computational resources, and hence, battery energy, which is not desired in case of mobile devices. Some game developers have already taken this into account and run their games with only 30 FPS. We also adopt this result and target to achieve 30 FPS within our GameOptimized Governor.

In general, the GameOptimized governor is designed for applications that exhibit similar workload characteristics for subsequent frames. It learns the workload within a range of frames and can be applied to all applications that maintain a constant frame rate by initiating periodic frame buffer changes within the OpenGL library as explained in Section V-C.

IV. Frame-based and Thread-based Workload Prediction

Our GameOptimized governor relies on a precise workload prediction of each thread as it performs fine-grained per-thread and per-frame power management. Overestimation of the thread workload will hinder the power saving potentials as it forces the cores to run at higher frequency, while underestimation will degrade the user experience by violating the FPS constraint. The criteria for choosing the predictor are 1) the applicability for a wide range of games without the need of manual parameter tuning for every single game and 2) the applicability for the various types of thread workloads we have discussed in Section III.

We have compared a set of predictors that have been previously considered for estimating the per-frame gaming workloads, and a number of hybrid predictors. The predictors are the PID, the simple moving average (SMA) [14], WMA [14], linear least squares (LLS) [15], LMS [15], and the ACR [16] predictors. Moreover, we evaluate three hybrid predictors that combine the ACR predictor with the PID, the SMA and WMA predictors, respectively. We have not considered the AR predictors used in previous works as they are not suitable for thread-based workload prediction. AR predictors need to be trained with recorded workloads to achieve a good result. As there can be hundreds of game threads that exhibit very different workloads, it is neither feasible to obtain the optimal weights for each thread nor to obtain one set of weights for all threads. The chosen predictors are fed with measured thread workload of five popular games, Dragon Fly, Star Wars Galactic Defense, Blood and Glory: Legend, Grand Theft Auto 3, and Asphalt 8, from different genres that exhibit different workload characteristics.

Table I shows the accuracy of each predictor for each game. The results per game are shown in terms of error of prediction, err_{game}, which is defined as follows.

$$err_{game} = \frac{1}{M} \sum_{\forall tid} \frac{1}{N} \sum_{\forall n} |W_{tid,m}(n) - W_{tid,p}(n)|, \qquad (1)$$

where $W_{tid,m}[t]$ and $W_{tid,p}[t]$ are the measured workload and predicted workload of thread tid at time slot n, M is the number of threads in the game, and N is the total number of frames within the measurement period. Our results show

TABLE I
Averaged prediction error e_p in percent.

Predictor	Asph. 8	Drag. Fly	Glad.	GTA III	Star W.
PID	9.74	40.97	31.10	3.51	10.45
SMA	9.18	38.32	28.76	3.47	10.17
WMA	9.03	38.37	28.78	3.39	10.04
LLS	9.87	44.46	31.77	3.64	10.63
LMS	unstable	unstable	unstable	unstable	unstable
ACR	9.73	552.74	34.68	2.90	11.56
Hyb. PID	7.79	39.34	29.44	2.82	10.00
Hyb. SMA	7.51	37.35	27.72	2.82	9.67
Hyb. WMA	7.41	37.36	27.66	2.80	9.61

that the hybrid predictor that combines the autocorrelation and the WMA predictor has the smallest prediction error. Hence, we have chosen this predictor and implemented it within our GameOptimized governor. While the ACR predictor performs well for periodic workloads, the WMA predictor achieves good results for aperiodic workloads. However, applying either only the WMA or the ACR predictor to all of the threads results in sacrificing prediction accuracy for the periodic or aperiodic thread workloads, respectively. This can also be seen from Table I, where the hybrid predictors have a lower prediction error than both comprising predictors separately. In the subsequent sections, we first describe the two comprising predictors, the WMA and ACR predictors before we describe the proposed hybrid WMA predictor.

A. Weighted Moving Average Predictor

The WMA predictor is motivated by weighted moving average filters [15]. The thread workload W of a frame is

estimated by calculating the weighted average of the past N frames,

$$W(n+1) = \frac{\sum_{i=0}^{N-1} (N-i) \cdot W(n-i)}{\sum_{i=1}^{N} i}, \quad (2)$$

where $W(n)$ is the workload of the n-th frame in seconds. The weights are chosen to be larger for more recent data such that it has greater influence on the prediction than older data. The window size, N, can be tuned. Inspection of different values for N showed that $N = 14$ leads to a good prediction for the WMA. Figure 4 shows an example workload from the game Asphalt and the corresponding prediction of this signal by the WMA predictor. As shown in the figure, the predicted curve follows the mean of the highly fluctuating workload.

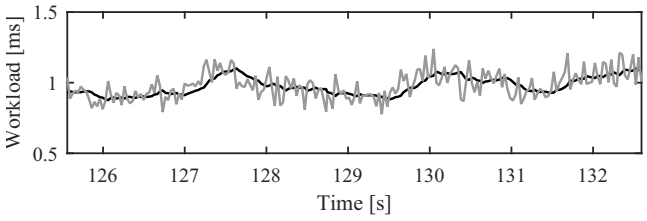

Fig. 4. Workload prediction with the WMA Predictor. The black line represents the predicted workload while the gray line shows the original signal.

B. Autocorrelation Predictor

As stated in Section III, some thread workloads show periodic behavior. Especially for these particular threads whose workload is zero most of the time, the WMA predictor does not result in satisfying prediction accuracy. However, such workloads are highly correlated to a shifted version of themselves. In other words, the workloads exhibit a high autocorrelation, which can be exploited in order to achieve better prediction. The ACR predictor is capable of exploiting the repetitions within a thread workload. The autocorrelation $ACorr(W, \tau)$ of the workload W at lag τ

$$ACorr(W, \tau) = \frac{ACovar(W, \tau)}{ACovar(W, 0)}, \quad (3)$$

is obtained by normalizing the autocovariance $ACovar(W, \tau)$ given by [16],

$$ACovar(W, \tau) = \sum_{i=0}^{N-\tau-1} (W(i+\tau) - \overline{W}) \cdot (W(i) - \overline{W}), \quad (4)$$

where $\overline{W}$ is the arithmetic mean of the workloads W. If $abs(ACorr(W, \tau))$ is close to 1, the workload W is highly correlated with itself, shifted by τ samples. By calculating the autocorrelation for $\tau = \{1, 2, ..., \Upsilon\}$, the lag with the highest autocorrelation τ_{Max} can be identified. The new workload is then estimated as

$$W(n+1) = W(n - \tau_{Max}). \quad (5)$$

Using this predictor, a perfectly periodic signal can be precisely predicted after a certain settling time. The total time it takes to learn the signal depends on the period of the signal. A perfectly periodic signal was created to test the autocorrelation predictor. Figure 5 shows an example in which it takes two periods for the ACR predictor to learn the periodic signal.

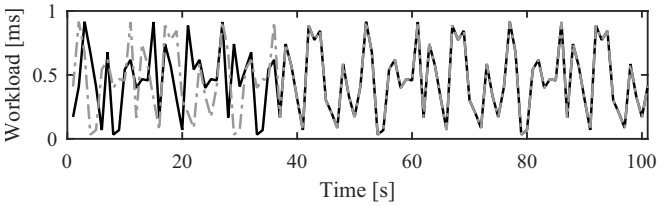

Fig. 5. Workload prediction with the ACR Predictor on a test data set with perfect repetitions. The solid line represents the predicted workload while the dashed line shows the original signal.

C. Hybrid WMA

The ACR predictor performs well if the predicted signal is highly autocorrelated. However, the predictor is not applicable for signals with low autocorrelation. For this reason, the ACR predictor was combined with the WMA predictor. After calculating the autocorrelation at lags τ for $\tau = \{1, 2, ..., \Upsilon\}$, the absolute maximum of the obtained autocorrelations is considered. If this maximum is above a threshold Θ, the prediction is done using the ACR predictor, otherwise the WMA predictor is used.

For the hybrid WMA predictor, we need to tune the parameters Υ, N, and Θ. Υ is the maximum number of lags to find whether the signal exhibits autocorrelation or not. N is the number of averaged data points for the WMA predictor. Θ is the threshold that indicates whether the ACR or the WMA predictor should be used. We have estimated the parameters experimentally by applying the predictor with varying parameter sets to the workload of the five games mentioned above. Then, we chose the results with the least prediction error and calculated their averages. In the following, we exemplify how we have determined the parameter N. For each game and thread, we have predicted the workload using the WMA predictor varying the parameter N = $\{1, 2, ..., 100\}$. For each run, we calculated the workload prediction error $err_{game,N}$. Then, we determined the minimum $err_{game,min}$ for each game and averaged all minimum errors to obtain the resulting parameter N. The approaches for Υ and Θ are similar, so we will not elaborate on this due to space reasons. Finally, the resulting values are N = 14, Υ = 20 and Θ = 0.34.

V. Game Power Management

In this section, we provide an overview of our proposed game power management based on the predictor we have stated above. First, we provide an overview of the complete framework. Next, we explain the underlying HMP platform and software components we have implemented. Finally, we present a heuristic algorithm that allocates threads to CPU cores and performs DVFS based on the workload prediction.

A. Overview of Frame- and Thread-based Power Management

The proposed frame-based and thread-based power management execution flow is summarized as follows:

1) Detect whether the running application is a game or not
2) Detect the beginning of a new frame
3) Predict the workload of each running thread
4) Allocate the threads to CPU cores according to the predicted workload
5) Set the CPU frequency to meet the desired frame rate

The proposed power management policy is implemented mainly in the Android governor. First, the governor needs to distinguish between the game processes and other application processes as we propose a power management technique tailored for mobile games. This is done by maintaining a hash table of known games. Second, the governor is notified about the beginning of a new frame. As our whole framework is based on per-frame power management and accurate prediction of the thread workloads, it is important to detect the precise point in time when the processing of a new frame begins. This is accomplished by modifying the OpenGL library, as described below. Third, we predict the workload of each thread required to process the frame. We implement the predictor described in Section IV. Fourth, we apply a heuristic algorithm to allocate threads to each core and set the operating frequency as will be described in Section V-D. The basic idea of the algorithm is to prefer the A7 CPU and distribute the threads' workloads as evenly as possible among all cores as long as the FPS constraint is not violated.

B. HMP Hardware Platform

The underlying hardware platform, on which we evaluate the proposed power management technique, is an Odroid-XU3 board. It features an Exynos5422 MPSoC that is also part of the Samsung Galaxy S5 smartphone [7]. This heterogeneous MPSoC is based on the ARM big.LITTLE architecture that consists of two different CPU clusters. One is a performance-optimized Cortex-A15 quad-core CPU (A15) and the other one is a power-optimized Cortex-A7 quad-core CPU (A7). We refer to a CPU cluster with a set of CPU cores as *CPU* while we refer to a single CPU core of a CPU cluster as *core*. The CPU voltage and frequency can be adjusted independently per CPU but not per individual core. The A7 supports a frequency range from 1.0, 1.1,$\cdots$ to 1.4 GHz while the A15 supports a range from 1.2, 1.3,$\cdots$ to 2.0 GHz. In addition, a Mali-T628 GPU and 2 GB of LPDDR3 memory are integrated into the Exynos5422 SoC. The operating system distribution of our setup is an Android Kitkat 4.4.4 with a Linux kernel version 3.10.9. The setup features so called *HMP scheduling* that allows to allocate tasks to all big and little cores simultaneously. The HMP scheduler prefers the small cores, and only migrates threads to the big cores if the CPU utilization rises above a certain threshold.

C. Software Architecture

As mentioned before, our power management technique is implemented as an Android CPU governor. Figure 6 shows the relevant software entities, which are the *GameOptimized governor*, and the *modified OpenGL library*.

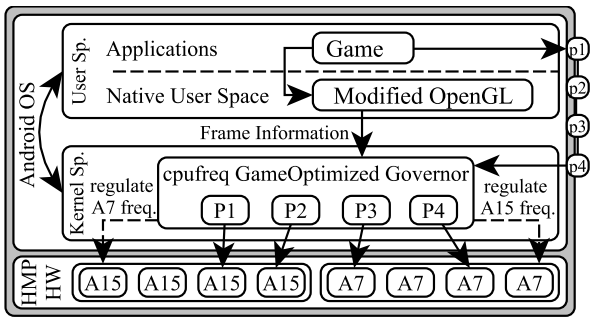

Fig. 6. System architecture with modified OpenGL library and combined governor/scheduler unit within the GameOptimized governor.

1) GameOptimized Governor: The GameOptimized governor, the key part of our power management technique, performs workload prediction, thread allocation and DVFS. It is implemented as a Linux kernel governor, which is a part of the *cpufreq* module. It contains a set of workload predictors that predict the workload for each game thread.

2) Modified OpenGL Library: The GameOptimized governor needs to be aware of the game context such as whether a game is being played, and the start time of the frame rendering. This is achieved with the help of the modified OpenGL library. The Android games we look into are downloaded from the Google Play Store. They are closed-source, and hence, we cannot instrument the source code to detect when a frame has been processed. Similar to the approach in [9], we calculate the frame rate by modifying the *eglSwapBuffers()* function of the OpenGL library. Usually, the frame buffer of a graphics unit consists of two buffers, a front and a back buffer. The currently displayed frame is stored in the front buffer while the next image is rendered into the back buffer. The function eglSwapBuffers() swaps the two buffers and the new frame is shown on the display. After this call, the processing of the next frame begins. We calculate the frame rate by measuring the time between two such function calls. This information is passed to the GameOptimized governor via an *ioctl()* call.

Game detection is also done in the modified OpenGL library. The process ID of the calling process is determined using the system's *getpid*() function. Via the process ID, the name of the process is read from the kernel's /proc file system. Next, a hash function is used to identify the process. All hashes of known games are calculated once and stored in a list.

D. Thread Workload Prediction-Based Core Allocation and Frequency Selection

In this section, we present our strategy for the thread-to-core allocation and the DVFS as well as the underlying hardware and software models. For the models, we take a cue from [10].

1) Hardware Model: Our hardware platform comprises two CPUs P_x where $x \in \{big, little\}$ with different performance characteristics described in Section V-B. Each P_x incorporates N_x cores where each core is denoted as $P_{x,i}$ with $i = 1, 2, ..., N_x$. The cores operate at the frequency f_x that ranges

from $f_{x,min}$ to $f_{x,max}$. Each core of each CPU provides a maximum capacity $C_{x,i}$. The capacity is defined as the number of CPU cycles that can be executed on one CPU core $P_{x,i}$ at the frequency $f_{x,j}$ within a given time. The time is dependent on the target frame rate FR_T. As introduced in [10], we define a *Migration Factor* that represents the performance difference between the big and the little cores. While one core of P_{little} can execute a number of n instructions, one core of P_{big} can execute $MigrationFactor_{big} \cdot n$ instructions within the same time. Experimentally, we have found that $MigrationFactor_{big} = 1.7058$ is a suitable number for the given platform. $MigrationFactor_{little}$ is normalized to 1. Hence, the maximum capacity of one core is calculated as

$$C_{x,i,max} = MigrationFactor_x \cdot \frac{f_{x,j}}{FR_T}, \tag{6}$$

while $C_{x,i}$ is the currently available capacity of one core and FR is the target frame rate. It needs to be considered that $C_{x,i}$ is further influenced by other threads executing on the CPUs, for example, threads spawned by the operating system. Therefore, we add an integral controller I_t to the calculation of the maximum available capacity for the game threads with

$$I_t = \sum_{1}^{n} \alpha \cdot (t_F - t_T), \tag{7}$$

where t_T is the target time for a frame to compute, t_F is the actual computation time, n is the total number of computed frames and α the integral gain of the controller. As α highly influences the speed at which the current CPU frequency adopts to frame misses, we introduce an α_{up} and an α_{down}. Experimentally, α_{up} was tuned for a too low frame rate, while α_{down} was tuned for a too high frame rate. We found that $\alpha_{up} = 0.2$ and $\alpha_{down} = 0.1$ are suitable for our application. Finally, I_t is converted to the actual control value I_c, which is measured in CPU cycles, and subtracted from the maximum available capacity $C_{x,i,max}$.

2) Software Model: A game consists of N_t threads T_k where $k = 1, 2, \cdots, N_t$. Each thread puts a workload W_k on the system that is measured in CPU cycles. The workload W_k of each thread takes up capacity on one CPU core $P_{x,i}$.

3) Power Consumption Characteristics of the Hardware: As described in Section V-B, our hardware platform comprises the power-efficient A7 CPU and the performance-oriented A15 CPU. Our measurements have shown that the A7 consumes significantly less energy than the A15 for the same workload, although the A15 computation time is significantly lower. Hence, it is more efficient in terms of energy consumption to shift as much workload as possible to the A7 and only switch to the A15 when the required FPS cannot be met. However, it is not energy-efficient to process the workload at the maximum frequency as it results in higher energy consumption.

Furthermore, we have run a measurement set, which shows that it is more power-efficient on our platform to distribute the threads over all available A15 cores and lower the frequency rather than utilizing fewer cores at a higher frequency. This is mainly due to the impossibility to turn off single A15 cores and the resulting high idle leakage currents of these cores. Table II shows the power consumption of the A15 at different frequency levels and different numbers of utilized cores. The load of each utilized core is approximately 70%. Running one core at 1.9 GHz (case 1) and running four cores at 1.2 GHz (case 2) consumes approximately the same amount of power. However, if we compare the workload of both cases to the workload of one core at 1.2 GHz, the workload of case 1 is only 58% larger, while the workload of case 2 is 300% larger. Hence, in this extreme case, we can execute 5 times the amount of work on multiple cores in case 2 consuming the same amount of power as in case 1. Based on these observations, we implement a strategy that aims to prefer the A7 if the FPS requirement is not violated. Further, we distribute the workload as even among the cores as possible to keep the CPU frequency as low as possible.

TABLE II
POWER CONSUMPTION OF THE A15 AT A UTILIZATION OF APPROX. 70%.

Volt.	Freq.	Idle	1 Core	2 Cores	3 Cores	4 Cores
1.0 V	1.2 GHz	0.30 W	0.70 W	1.07 W	1.46 W	1.80 W
1.0 V	1.3 GHz	0.34 W	0.79 W	1.21 W	1.65 W	2.05 W
1.0 V	1.4 GHz	0.37 W	0.85 W	1.35 W	1.80 W	2.26 W
1.0 V	1.5 GHz	0.42 W	0.98 W	1.51 W	2.03 W	2.57 W
1.1 V	1.6 GHz	0.48 W	1.13 W	1.76 W	2.37 W	2.99 W
1.1 V	1.7 GHz	0.55 W	1.30 W	2.01 W	2.74 W	3.50 W
1.1 V	1.8 GHz	0.66 W	1.48 W	2.30 W	3.20 W	4.05 W
1.2 V	1.9 GHz	0.73 W	1.72 W	2.74 W	3.85 W	4.92 W
1.3 V	2.0 GHz	0.93 W	2.23 W	3.49 W	4.93 W	6.49 W

4) Thread to Core Allocation: The strategy we use to distribute the game tasks to the CPU cores is shown in Algorithm 1. It is executed once every frame. First, frequencies of both CPUs P_x are set to the minimum value and the capacity $C_{x,i}$ of each core is reset. Then, the workload of the next frame is predicted for each game thread. Next, we iterate through the threads T_k and assign them to CPU cores. We begin with the A7 cores at the minimum frequency level $f_{little,min}$. To prevent re-allocation overhead, we first check whether the previously assigned core offers enough capacity for the current thread. If not, we assign it to the core with the most available capacity. However, if none of the cores at the current frequency level provides enough capacity $C_{x,i}$ for the workload W_k of thread T_k, we increase f_{little} to the next higher frequency level. If we cannot find a suitable core on the A7 CPU, we reset its frequency to the previous level. Then, we repeat the same procedure for the cores of the A15 CPU. If we cannot find a suitable core on the A15, we assign the thread to the A15 core with the minimum workload.

VI. EVALUATION AND USER STUDY

We have evaluated the GameOptimized governor by comparing it to the two most popular Android default governors, Interactive and Ondemand. Moreover, we performed a user study to test the users' perception of the implemented governor. The results show that we save on average 41.9% of energy compared to the Interactive and 31.2% compared to the Ondemand governor with only a small degree of perceptible performance loss.

```
Initialize f_x = f_{x,min} for x ∈ {big, little}
C_{x,i} = C_{x,i,max} − I_c at f_{x,min} and i = 1..N_x
Predict workload W_k for each thread T_k where k = 1..N_t
for k = 1 to N_t do
    if W_k == 0 then
        Do not change allocation of T_k
    else
        Set x = little
        while f_x <= f_{x,max} do
            Check previous allocation and if needed iterate
            through all cores at f_x to find core with max(C_{x,i})
            if max(C_{x,i}) > W_k then
                Allocate T_k to C_{x,i}
                C_{x,i} = C_{x,i} − W_k
                Continue with next thread T_k
            else
                Increase f_x to the next level
            end if
            if f_little == f_{little,max} then
                Set f_little to previous value and restart
                while-iteration with x = big
            end if
            if f_big == f_{big,max} then
                Allocate T_k to P_{big,i}, with max(C_{big,i})
                Continue with next thread T_k
            end if
        end while
    end if
end for
```

Algorithm 1. Thread to core allocation and frequency selection strategy.

A. Energy Consumption and Frame Rate Evaluation

The main objective of the GameOptimized governor is to lower the power consumption of the two CPUs on the Odroid-XU3 board during the game play. To evaluate the governor, we chose twelve games of different genres. We played each game three rounds using 1) our GameOptimized governor, 2) the Interactive governor, and 3) the Ondemand governor. Each round had a duration of 10 minutes. For comparable results and synchronized measurements the board was rebooted before each measurement. To avoid that changes of the GPU frequency interfere with the frame rate, the GPU frequency was fixed to the maximum possible value of 543 MHz.

Figure 7 shows the total CPU energy consumption (A7 and A15) for the twelve test games for each of the three governors. The GameOptimized governor can achieve noticeable energy savings for all of the games, up to 60.0% compared to the Interactive governor and 58.5% compared to the Ondemand governor. In general, the savings highly depend on the type of the game and hence, the workload that is generated. Games like I, Gladiator, Fruit Ninja, Dragon Fly and Sonic Jump do not generate high workload. Consequently, both default governors whose power management strategy is highly workload dependent, do not ramp up the frequency of the A15 as they do for example for GTA 3 or Interstellar. For those less resource demanding games, the energy consumption using the default governors is anyway comparatively small. Hence, the power savings for the GameOptimized governor are smaller than for highly resource demanding games. Figure 8 shows the average FPS for the twelve test games for each

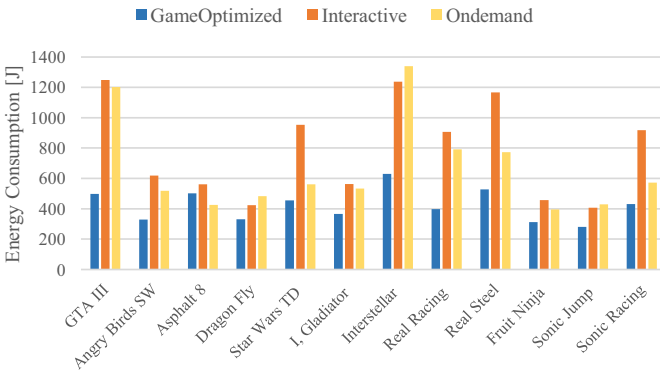

Fig. 7. Total energy consumption of both CPUs for the twelve test games for the game power manager and the Android default governors.

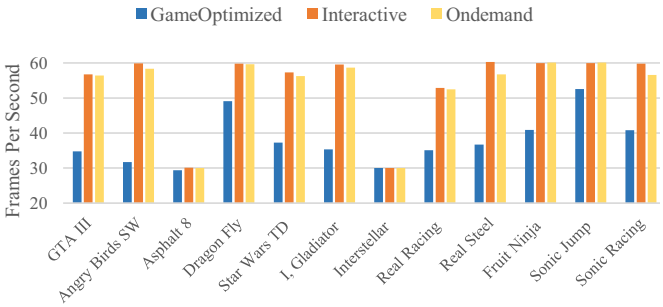

Fig. 8. Average frame rate for the twelve test games for the game power manager and the Android default governors.

of the three governors. As described in Section V-A, our implementation is designed in a way that it targets a frame rate of 30 FPS to guarantee a good user experience. The figure shows that the average frame rate for all games is between 30 and 40 FPS. For Dragon Fly and Sonic Jump, we can observe that the achieved frame rate is higher than for the other games, approximately 50 FPS. This effect is caused by the above mentioned low resource demand of those games. Even by applying the GameOptimized governor's thread allocation scheme and frequency down scaling, one frame is processed faster than within 33 ms. Consequently, the frame rate clamps at a higher value. A solution to this problem is to apply a more aggressive power management technique, for example delay the call to the eglSwapBuffers() function. Another effect that becomes visible in Figure 8 is the frame limitation, which is by default incorporated in some of the games. For Interstellar and Asphalt 8, the frame rate is already limited to 30 FPS. The game Interstellar is a good example for showing the energy saving efficiency of the GameOptimized governor. Although the frame rate stays the same as for the other two governors, the energy consumption is halved. Hence, we can claim that the energy savings we achieve are not only due to the frame rate reduction, but because of the applied thread to core allocation and DVFS scheme.

B. User Study

In the previous section, we have shown that that the GameOptimized governor achieves high energy savings. Due

to the reduced target frame rate of 30 FPS, it needs to be ascertained whether a decrease in the user experience is perceivable or not. For this reason, a user study with ten participants was conducted. The goal of this study was to obtain a realistic rating of the gaming performance of the GameOptimized governor compared to the Android default Interactive governor. For the study, we have chosen five games, a subset of the twelve games presented in Section VI-A: GTA III, Angry Birds Star Wars, Interstellar, Fruit Ninja and Sonic Racing. The reduction of the game number from twelve to five leads to a duration of the study of about one hour per person.

For each game, there are two phases, the training phase and the actual playing phase. During the training phase, the participant can try the game for an unlimited amount of time to get used to the game play and the controls. In the playing phase, the participant plays every game twice for three minutes, once with the GameOptimized governor (measurement 1) and once with the Interactive governor (measurement 2). The governors are picked in random order, hence, the user does not know which governor is currently active. The Odroid-XU3 board is rebooted before each measurement. After one measurement, the user is asked to rate the game play. The possible grades are in a range from one to six, where one is the best (no lags or glitches) and six is the worst (game is not playable).

Table III shows the rating results from all participants for all games. We can see that a small performance decrease was notable for the GameOptimized governor for four of the five games. Especially Fruit Ninja, which is a highly interactive game with a lot of fast animations, was rated worse than the other games. Still, the participants considered the gaming experience as *good* and *very good* for most of the measurements. Moreover, the grading difference between measurement 1 and measurement 2 is not more than 1 step apart for almost all cases. After the measurements, we showed the participants the amount of energy (in percent) that could be saved by applying the GameOptimized governor. All of them agreed to compromise a little performance for the energy savings that can be achieved with the GameOptimized governor.

TABLE III
GRADINGS IN THE USER STUDY FOR THE GAMEOPTIMIZED GOVERNOR (G) AND THE INTERACTIVE GOVERNOR (I) PER PARTICIPANT (P).

Game	GTA III		Angry B.		Interstel.		Fruit N.		Sonic R.	
Gov.	G	I	G	I	G	I	G	I	G	I
P1	2	1	2	1	1	1	2	1	1	1
P2	1	1	2	1	1	1	2	2	1	1
P3	2	2	1	1	1	1	1	1	1	1
P4	1	1	2	1	2	2	1	1	1	1
P5	1	1	1	1	1	1	1	1	1	1
P6	2	2	2	3	1	1	4	3	2	2
P7	1	2	1	1	1	1	1	1	1	1
P8	1	1	2	1	1	1	2	1	1	1
P9	2	1	2	2	1	1	2	1	2	2
P10	2	1	1	1	1	1	3	1	3	1
∅	1.5	1.3	1.6	1.3	1.1	1.1	1.9	1.3	1.4	1.2

VII. CONCLUDING REMARKS

In this work, we have implemented a predictive, thread- and frame-based game power manager on an HMP SoC, the Odroid-XU3 board, for Android. Compared to previous works that have looked either into workload prediction or thread to core allocation, we combine the advantages of a frame-based workload predictor with an energy-aware thread to core allocation on the power-efficient little CPU and the performance-oriented big CPU. We predict the workload of all game threads per frame and use this information to distribute the threads over the CPU cores such that we can minimize the CPU frequency, and hence, save energy. The predictor differentiates between periodic and aperiodic workload. We evaluate our power manager in a user study, which reveals that our power manager can save on average 41.9% of total energy consumption while still maintaining a *good* and *very good* user experience. For future work, we plan to combine the CPU governor with a frame-based GPU governor.

REFERENCES

[1] C. Klotzbach, "Enter the matrix: App retention and engagement," *Yahoo Flurry Analytics*, 2016.

[2] Y. Gu and S. Chakraborty, "Control theory-based DVS for interactive 3d games," in *45th Annual Design Automation Conference*, 2008.

[3] 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*, vol. 64, no. 1, pp. 260–273, 2015.

[4] Y. Gu and S. Chakraborty, "Power management of interactive 3d games using frame structures," in *21st International Conference on VLSI Design*, 2008.

[5] M. Claypool, K. Claypool, and F. Damaa, "The effects of frame rate and resolution on users playing first person shooter games," in *Multimedia Computing and Networking*, 2006.

[6] H. Aydin and Q. Yang, "Energy-aware partitioning for multiprocessor real-time systems," in *International Parallel and Distributed Processing Symposium*, 2003.

[7] Hardkernel co., Ltd., "Odroid-XU3," http://www.hardkernel.com, 2015.

[8] B. Dietrich, S. Nunna, D. Goswami, S. Chakraborty, and M. Gries, "Lms-based low-complexity game workload prediction for DVFS," in *IEEE International Conference on Computer Design*, 2010.

[9] B. Dietrich and S. Chakraborty, "Lightweight graphics instrumentation for game state-specific power management in Android," *Multimedia Systems*, vol. 20, no. 5, pp. 563–578, 2014.

[10] A. Pathania, S. Pagani, M. Shafique, and J. Henkel, "Power management for mobile games on asymmetric multi-cores," in *International Symposium on Low Power Electronics and Design*, 2015.

[11] A. Pathania, Q. Jiao, A. Prakash, and T. Mitra, "Integrated CPU-GPU power management for 3d mobile games," in *51st Annual Design Automation Conference*, 2014.

[12] 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*, 2015.

[13] Z. Cheng, X. Li, B. Sun, C. Gao, and J. Song, "Automatic frame rate-based DVFS of game," in *International Conference on Application-specific Systems, Architectures and Processors*, 2015.

[14] S. W. Smith, *Digital Signal Processing*. California Technical Publishing, 1999.

[15] S. Haykin, *Adaptive Filter Theory*. Prentice Hall, 2014.

[16] P. Stoica and R. Moses, *Spectral analysis of signals*. Prentice Hall, 2005.