Power-Performance Modelling of Mobile Gaming Workloads on Heterogeneous MPSoCs

Anuj Pathania*, Alexandru Eugen Irimiea†, Alok Prakash†, Tulika Mitra†
*Karlsruhe Institute of Technology, †National University of Singapore
Corresponding Author: tulika@comp.nus.edu.sg

ABSTRACT

Games have emerged as one of the most popular applications on mobile platforms. Recent platforms are now equipped with Heterogeneous Multiprocessor System-on-Chips (HMPSoCs) tightly integrating CPUs and GPUs on the same chip. This configuration enables high-end gaming on the platform but at the cost of high power consumption rapidly draining the underlying limited-capacity battery. The HMPSoCs are capable of independent Dynamic Voltage and Frequency Scaling (DVFS) for CPUs and GPUs for reduction in platform's power consumption. State-of-the-art power manager for mobile games on HMPSoCs oversimplifies the complex CPU-GPU interplay. In this paper, we develop power-performance models predicting the impact of DVFS on mobile gaming workloads. Based on our models, we propose an efficient power management strategy and implement it on an Odroid-XU+E mobile platform. Measurements on the platform show that our power manager provides on average 20% increase in performance per watt when compared to the state-of-the-art.

Categories and Subject Descriptors

C.1.4 [**Parallel Architectures**]: Mobile processors

General Terms

Algorithms, Design, Management, Performance

Keywords

GPU, Power-Performance Model, Games

1. INTRODUCTION

Contemporary mobile platforms are equipped with Heterogeneous Multiprocessor System-on-Chips (HMPSoCs). HMPSoCs, powered by tightly coupled CPUs and GPUs, allow platforms to offer gaming performance comparable to that of desktop GPUs of the recent past. However, these games consume lot of power and rapidly drain the batteries of the underlying platforms [2]. Figure 1 shows the average power consumption of multi-core CPU and GPU for latest mobile games running on Odroid-XU+E [3] mobile platform containing Exynos 5 Octa (5410) HMPSoC [1][1]. Figure 1 confirms that games use CPU (for artificial intelligence and physics) and GPU (for 3D rendering) of the HMPSoC in tandem. Hence, both of them contribute towards the HMPSoC's total power consumption.

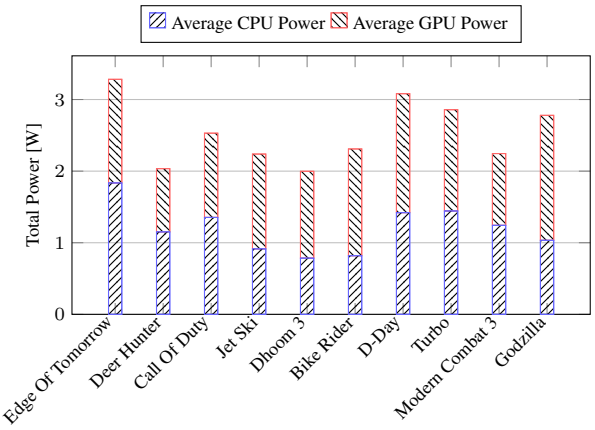


Figure 1: Average power consumption for latest mobile games.

A HMPSoC is capable of performing independent Dynamic Voltage and Frequency Scaling (DVFS) [19] for CPU and GPU. This capability enables power efficient execution of games, specially when their Quality of Service (QoS) is considered. The standard QoS metric for games is Frames per Second (FPS) [16].

Default run-time power management in these platforms is performed somewhat independently for CPU and GPU. CPU power management is performed by one of the standard Linux power manager [18]. *Conservative* power manager is the default manager for the CPU in our platform. GPU power management is handled by a closed source firmware. These power managers are completely agnostic to an application's QoS. Thus, they lack synergy and may waste power by overclocking the CPU and/or the GPU.

Authors in [5] proposed an FPS-aware power manager for CPUs in HMPSoCs specifically for gaming workloads. They predicted the future CPU workloads based on the past observations and then performed its DVFS. However, they ignored the accompanying GPU and thereby were not required to model the CPU-GPU dynamics.

In our previous work [4], we proposed an integrated CPU-GPU power manager to minimise power consumption of HMPSoCs, while achieving user-defined FPS ranges for operating games. It was based on a reactive online heuristic where CPU-GPU DVFS was

[1]The details of the experimental setup are described in Section 3.

DAC '15, June 07 - 11, 2015, San Francisco, CA, USA

http://dx.doi.org/10.1145/2744769.2744894.

performed based on their respective cost functions defined as utilisation $\times$ frequency. However, this manager assumed that FPS is independently proportional to the CPU- and GPU cost. Thus, each component was frequency scaled independently based on its current cost. This is a simplistic assumption because different games have different levels of sensitivity towards CPU and GPU vis-a-vis FPS. Some games are more CPU dependent, while others are more dependent on GPU. Either component can become a bottleneck during gameplay and increasing frequency of the non-bottlenecked component in hope of increasing FPS leads to power wastage. We rectify this shortcoming in this work by developing models that can identify the bottlenecked component. Power managers based on our models can increase the frequency of bottlenecked component, while keeping non-bottlenecked component frequency untouched or even reduce its frequency to further save power.

Further, the power manager in [4] also assumes perfectly linear relationship between FPS and CPU-GPU costs with slope equal to 1 (i.e., a 45-degree line). This assumption is not always true and leads to power inefficiency. Instead, we employ linear regression models to characterise the complex relationship between power, performance, utilisation, and frequency at various operating points. Our offline regression analysis with ten popular and compute-intensive mobile games identifies the appropriate slope for each relationship. We also integrate an on-line learning mechanism that further refines the parameters during gameplay, making the models more accurate progressively. We employ these models to design a power manager that strives to achieve a targeted FPS with minimal power consumption.

We implement our power manager in the Android OS of the Odroid-XU+E [3] mobile platform. Measurements on the platform show that our power manager provides on average 20% increase in performance per watt compared to the state-of-the-art [4].

2. RELATED WORK

Existing research have so far mostly focused on developing power models for General Purpose Computing on Graphics Processing Units (GPGPU)- [8] and gaming applications [13] for PC-class GPUs designed for high performance computing. Unlike GPGPU workload, a gaming workload is not interchangeable between CPU and GPU, and requires them to work in tandem. The embedded CPU-GPU architecture is also significantly different from its PC-class counter part [9].

A detailed characterisation of mobile games for PC-class- and embedded GPUs was performed in [20] and [7], respectively. Authors in [6] developed microbenchmarks that stressed the various stages of graphics pipeline of an embedded GPU to study its impact on power consumption of an underlying platform, but the work neither provided any mathematical formulation for the observed behaviour nor evaluated any mobile game.

Foremost power mangers ([14], [12] and [11]) for mobile games were based on information derived directly from the source code of the games. These managers are not practical as most of the present popular games are closed source. Recent FPS-aware power managers ([5], [4]) can operate with closed source commercial games and have been described in detail in Section 1.

3. EXPERIMENTAL SETUP

We use Odroid-XU+E mobile platform [3] for our experiments. The platform is built around Exynos 5 Octa (5410) HMPSoC [1] currently deployed in Samsung Galaxy S4 smartphones. HMPSoC integrates a CPU based on ARM's big.LITTLE [10] asymmetric multi-core architecture and PowerVR SGX544MP3 GPU on the same chip. The CPU contains a quad-core Cortex-A7 and a quad-core Cortex-A15 cluster, but only one cluster can operate at a time. We experiment with graphics intensive games for which the low power-performance A7 cluster is insufficient to support the expected level of performance. Thus, we only use the high power-performance A15 cluster, while the A7 cluster is powered down.

CPU Frequency (MHz)	CPU Voltage (V)
800, 900, 1000	0.9
1100, 1200, 1300	1.0
1400, 1500, 1600	1.1

Table 1: Cortex-A15 CPU frequencies and corresponding voltages.

GPU Frequency (MHz)	GPU Voltage (V)
177, 266, 350	0.9
480	1.0
532 ,640	1.1

Table 2: PowerVR GPU frequencies and corresponding voltages.

The A15 cluster can operate at nine different frequency levels, while the GPU can operate at six different frequency levels. Tables 1 and 2 show the available CPU and GPU frequencies along with their corresponding operating voltages, respectively.

We install Android 4.2.2v with Linux kernel 3.4.5v on the platform and select twenty compute-intensive (in terms of either CPU or GPU workload) popular mobile games for this work.

We perform DVFS by writing the desired frequency into the exposed configuration files of CPU and GPU drivers, which then switch the voltages accordingly. We measure FPS by intercepting the frame composition logs generated by Android's "Surface Flinger" service. The utilisations of CPU cores are obtained from the kernel activity logs. The utilisation of the GPU is obtained from a software counter provided by the GPU driver. The on-board Texas Instrument INA231 sensors allow us to obtain current measurements for CPU and GPU, independently.

4. POWER-PERFORMANCE MODEL

We propose a predictive power manager in Section 5, which estimates the performance (FPS) and power consumption of a gaming workload at different CPU-GPU DVFS combinations. To achieve this objective, we develop predictive *regression models*. We use a learning set of ten games to build the regression models. The models are evaluated using a disjointed set of ten test games.

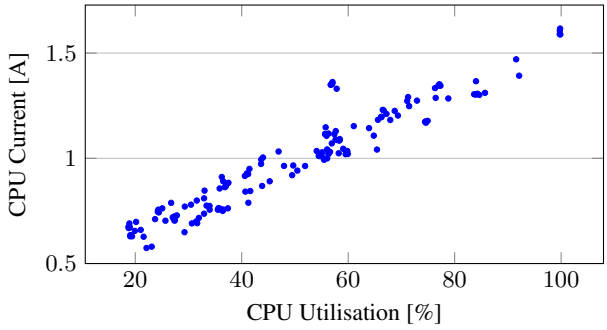
4.1 Power Model

Let CPU (or GPU) frequency be represented by $\mathbf{F_C}$ (or $\mathbf{F_G}$). $\mathbf{V_C^{F_C}}$ (or $\mathbf{V_G^{F_G}}$) represents CPU (or GPU) voltage at frequency level F_C (or F_G). CPU (or GPU) voltages corresponding to CPU (or GPU) frequencies can be obtained from Table 1 (or Table 2). Let $\mathbf{I_C^{(F_C,F_G)}}$ (or $\mathbf{I_G^{(F_C,F_G)}}$) represent CPU (or GPU) current drawn at CPU-GPU frequency combination (F_C, F_G). CPU and GPU power, represented by $\mathbf{P_C^{(F_C,F_G)}}$ and $\mathbf{P_G^{(F_C,F_G)}}$ is given by Equation (1) and (2), respectively.

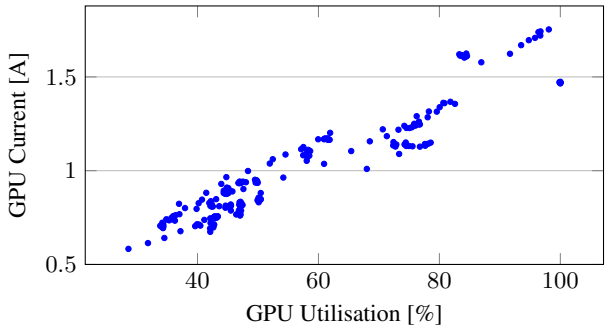
$$P_C^{(F_C,F_G)} = V_C^{F_C} I_C^{(F_C,F_G)} \quad (1)$$

$$P_G^{(F_C,F_G)} = V_G^{F_G} I_G^{(F_C,F_G)} \quad (2)$$

The current drawn by the CPU (or GPU) also depends upon frequency level of GPU (or CPU) because of their performance coupling. For example, at higher frequencies CPU may generate more work for GPU leading to a higher current draw for GPU.



(a) CPU current drawn at different CPU utilisation



(b) GPU current drawn at different GPU utilisation

Figure 2: Current drawn for different utilisations at max. frequency

The dynamic power of a component can be modelled as ACV^2f, where A is Activity Factor, C is capacitance, V is voltage, and f is frequency. The leakage power is estimated as VI_{leak}, where I_{leak} is the leakage current. Therefore, the total power can be modelled as $P = ACV^2f + VI_{leak}$ [17]. C and I_{leak} are platform-specific constants. As power equals voltage times current, dividing the total power P by voltage V gives Equation (3) for the current drawn I.

$$I = ACVf + I_{leak} \tag{3}$$

Due to lack of software-visible performance counters in our platform, we are restricted to using the CPU utilisation level as an approximate representation of a workload's activity factor.

Figures 2(a) and 2(b) plot the current drawn at different utilisation levels for CPU and GPU, respectively. For Figure 2(a) (or 2(b)), we keep CPU (or GPU) at maximum frequency but vary workloads and GPU (or CPU) frequency to obtain different CPU (or GPU) utilisations and current draws. The Figure 2(a) (or 2(b)) shows a near-linear relationship of current drawn by CPU (or GPU) with its utilisation and establish that utilisation can faithfully model its activity factor A. Let $\mathbf{U_C^{(F_C,F_G)}}$ (or $\mathbf{U_G^{(F_C,F_G)}}$) represents CPU (or GPU) utilisation at CPU-GPU frequency combination (F_C, F_G). Therefore, based on Equation (3), CPU and GPU current draws can be modelled using Equation (4) and (5), respectively.

$$I_C^{(F_C,F_G)} = \alpha_1 V_C^{F_C} U_C^{(F_C,F_G)} F_C + \alpha_2 \tag{4}$$

$$I_G^{(F_C,F_G)} = \beta_1 V_G^{F_G} U_G^{(F_C,F_G)} F_G + \beta_2 \tag{5}$$

Here α_1 (or β_1) denotes the capacitance C and α_2 (or β_2) denotes the leakage current I_{leak} of CPU (or GPU). We fix the values of $\alpha_1, \alpha_2, \beta_1$ and β_2 through linear regression for our platform.

4.2 Performance Model

A gaming workload's performance (FPS) depends upon frequencies and utilisations of both CPU and GPU [4]. Figure 3(a) (or 3(d)) plots FPS against different CPU (or GPU) frequencies for a fixed GPU (or CPU) frequency. Figure 3(b) (or 3(e)) plots CPU (or GPU) utilisation against different CPU (or GPU) frequencies for a fixed GPU (or CPU) frequency. Figure 3(c) (or 3(f)) plots CPU (or GPU) utilisation against different GPU (or CPU) frequencies for a fixed CPU (or GPU) frequency.

Figure 3(a) (or 3(d)) shows that increasing CPU (or GPU) frequency causes increase in FPS for most of the games. However, there are certain exceptions. For example, we observe that the FPS for *Deer Hunter* workload remains fixed at 60 FPS. Android limits its FPS to the refresh rate of the display, which is 60 FPS in our platform. Some games also employ internal FPS control that limits the highest achievable FPS for them. In case of *Deer Hunter*, FPS reaches the limit (60 FPS) at the lowest CPU-GPU frequency combination. Therefore, there is no scope for any further increase.

In Figure 3(a), we observe that *Bike Rider* workload does not respond to increase in CPU frequency. This is because the GPU utilisation of this workload is nearly always at 100% (Figure 3(f)) making it GPU bound. GPU bound games will not respond to CPU DVFS. Similarly in Figure 3(d), *Edge of Tomorrow* workload does not respond to GPU DVFS at 480 MHz or higher. The CPU utilisation for this workload is 100% throughout (Figure 3(c)) making it CPU bound. For this workload, below the GPU frequency of 480MHz, both CPU and GPU are bottlenecks and it responds to increase in GPU frequency. But beyond 480MHz, CPU becomes the only bottleneck and FPS is unresponsive to any further increase.

In summary, *a game's FPS increases with increase in a component's (CPU or GPU) frequency except when (a) the game reaches the maximum FPS defined either by platform OS or by the game developer, or (b) the other component becomes a bottleneck.*

We assume that we can measure the FPS $\mathbf{Q^{(F_C,F_G)}}$ and utilisation values $U_C^{(F_C,F_G)}$, $U_G^{(F_C,F_G)}$ at the current CPU-GPU frequency combination (F_C, F_G). Given the observations from the characterisation study, we now proceed to estimate the FPS at a higher CPU and GPU frequency level F'_C and F'_G, respectively. The linear relationship between FPS and CPU-GPU frequencies can be modelled as follows.

$$Q^{(F'_C,F_G)} = Q^{(F_C,F_G)} + \gamma_1(F'_C - F_C) \tag{6}$$

$$Q^{(F_C,F'_G)} = Q^{(F_C,F_G)} + \gamma_2(F'_G - F_G) \tag{7}$$

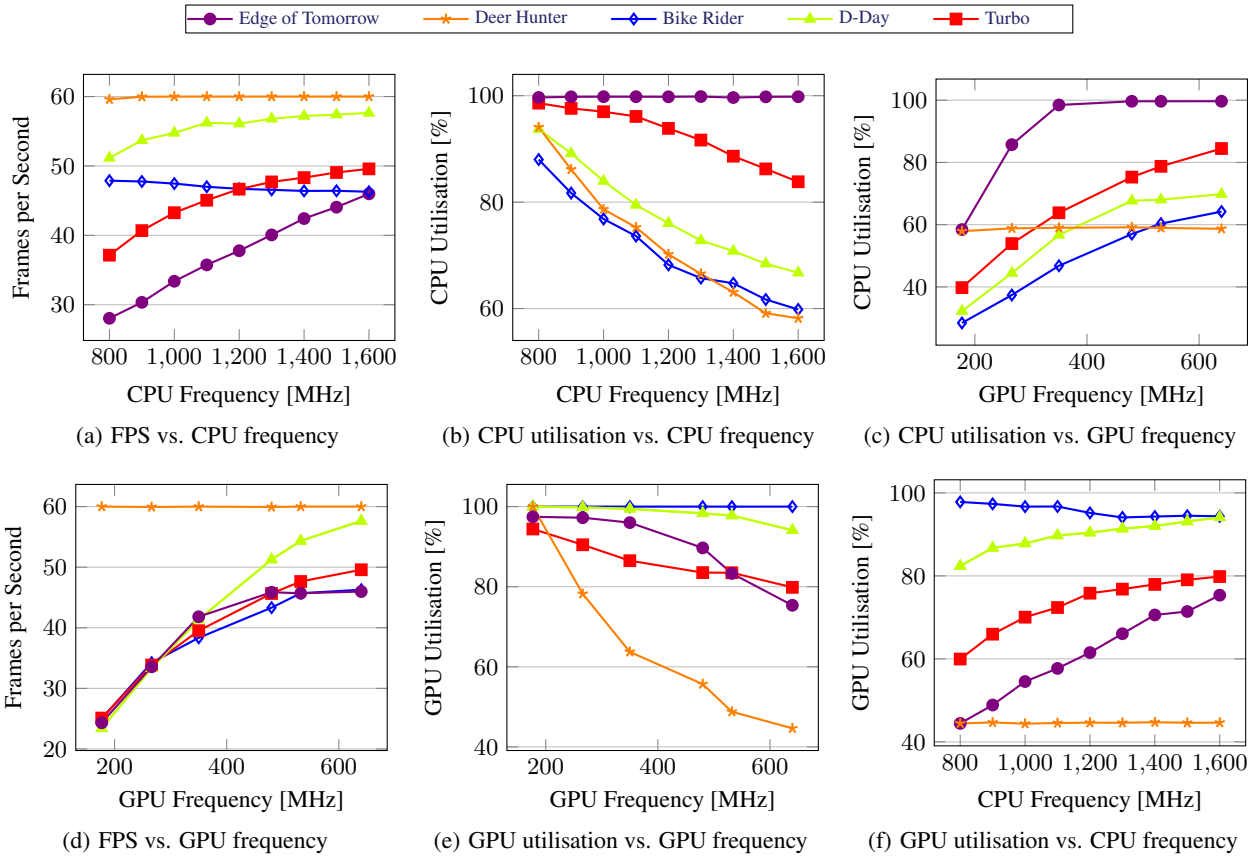
To model the FPS limit, let $\hat{\mathbf{Q}}$ be the maximum FPS a game can attain. We can obtain $\hat{Q}$ for a game by setting both CPU and GPU together at their respective highest frequencies during the execution. The following equations model the FPS bottleneck.

$$Q^{(F'_C,F_G)} = \min\left(\hat{Q}, (Q^{(F_C,F_G)} + \gamma_1(F'_C - F_C)\right) \tag{8}$$

$$Q^{(F_C,F'_G)} = \min\left(\hat{Q}, Q^{(F_C,F_G)} + \gamma_2(F'_G - F_G)\right) \tag{9}$$

Next, in order to model the bottleneck in the other component, let $\hat{\mathbf{U}}_\mathbf{C}$ and $\hat{\mathbf{U}}_\mathbf{G}$ be the maximum CPU and GPU utilisation for a game, respectively. Theoretically maximum value for utilisation is 100% but in practice we may observe a lower upper bound because of memory access latency or memory bandwidth saturation. We can obtain $\hat{U}_C$ (or $\hat{U}_G$) for a game by setting CPU (or GPU) at the lowest frequency while keeping GPU (or CPU) at the highest frequency. Utilisation bottlenecks are described by following equations.

$$Q^{(F'_C,F_G)} = \begin{cases} \min\left(\hat{Q}, (Q^{(F_C,F_G)} + \gamma_1(F'_C - F_C)\right) & \text{if } U_G^{F_C,F_G} \neq \hat{U}_G \\ Q^{(F_C,F_G)} & \text{otherwise} \end{cases} \tag{10}$$



(a) FPS vs. CPU frequency (b) CPU utilisation vs. CPU frequency (c) CPU utilisation vs. GPU frequency

(d) FPS vs. GPU frequency (e) GPU utilisation vs. GPU frequency (f) GPU utilisation vs. CPU frequency

Figure 3: Effects of CPU (or GPU) DVFS on FPS, CPU- and GPU utilisations of games, when GPU (or CPU) is kept at maximum frequency.

$$Q^{(F_C,F'_G)} = \begin{cases} \min\left(\hat{Q}, Q^{(F_C,F_G)} + \gamma_2(F'_G - F_G)\right) & \text{if } U_C^{F_C,F_G} \neq \hat{U}_C \\ Q^{(F_C,F_G)} & \text{otherwise} \end{cases} \quad (11)$$

Figure 3(b) (or 3(e)) shows that CPU (or GPU) utilisation decreases with increase in its frequency but with some exceptions. The workload *Edge of Tomorrow* is CPU bound with 100% utilisation (Figure 3(b)) and increasing the CPU frequency merely helps it to do more processing to achieve higher FPS and the utilisation still remains at 100%. The GPU is also a bottleneck for this workload upto 350MHz frequency (Figure 3(e)) but beyond that GPU utilisation decreases with increasing GPU frequency. The workloads *Bike Rider* and *D-Day* are mostly GPU bound and remain at 100% utilisation even with increase in GPU frequency.

In summary, *the utilisation of a component decreases with increase in frequency unless the component is a bottleneck with close to 100% utilisation*. The linear relationship between a component's utilisation and its frequency can be modelled as follows.

$$U_C{}^{(F'_C,F_G)} = \begin{cases} U_C{}^{(F_C,F_G)} + \alpha_3(F'_C - F_C) & \text{if } U_C^{F_C,F_G} \neq \hat{U}_C \\ U_C{}^{(F_C,F_G)} & \text{otherwise} \end{cases} \quad (12)$$

$$U_G{}^{(F_C,F'_G)} = \begin{cases} U_G{}^{(F_C,F_G)} + \beta_3(F'_G - F_G) & \text{if } U_G^{F_C,F_G} \neq \hat{U}_G \\ U_G{}^{(F_C,F_G)} & \text{otherwise} \end{cases} \quad (13)$$

The utilisation of the CPU (or GPU) is also impacted by change in GPU (or CPU) frequency even if the CPU (or GPU) frequency is kept constant. Figure 3(c) shows an increase in CPU utilisation with increase in GPU frequency as long as there is a parallel increase in FPS (Figure 3(d)). The increase in FPS forces CPU to process additional frames at the same frequency, thereby increasing its utilisation. Similar observations can be made for GPU utilisation with increase in CPU frequency in Figure 3(f). We conceptualise the impact of cross-component frequency variations on utilisations using the following equations.

$$U_C{}^{(F_C,F'_G)} = U_C{}^{(F_C,F_G)} + \alpha_4(Q^{(F_C,F'_G)} - Q^{(F_C,F_G)}) \quad (14)$$

$$U_G{}^{(F'_C,F_G)} = U_C{}^{(F_C,F_G)} + \beta_4(Q^{(F'_C,F_G)} - Q^{(F_C,F_G)}) \quad (15)$$

4.3 Evaluation of the Models

We evaluate our models with a set of twenty games, where ten games form the learning set on which the regression models are built upon, while the remaining ten games form the test set. Tables 3 and 4 show the average predictive errors for the games in learning- and test set across all different frequency combinations, respectively. The average errors in predicting CPU and GPU power consumptions are 6.42% and 7.86%, respectively. In comparison, the regression based predictive models presented in [13], which also predict GPU power for ten benchmarks, have an average error of 12.56%. The average error in predicting performance (FPS) is 3.87%. We believe ours is the first work to predict FPS for games.

We are limited by the availability of only high-level utilisation counters in our platform. Some of the games we test show substantial variability that cannot be explained just with their CPU and GPU utilisations, and requires knowledge of lower level hardware activity. We plan to build more accurate models once detailed counters become available in the future.

	Edge of Tomorrow	Deer Hunter	Call of Duty	Jet Ski	Dhoom 3	Bike Rider	D-Day	Turbo	MC3	Godzilla
P_C	5.43%	5.14%	6.70%	7.51%	6.22%	6.62%	5.19%	3.94%	6.62%	5.53%
P_G	7.03%	9.57%	7.06%	5.32%	10.26%	6.69%	6.98%	6.68%	6.69%	7.05%
Q	2.39%	0.08%	2.11%	3.63%	3.48%	11.41%	3.32%	13.91%	11.41%	1.30%

Table 3: Prediction errors for CPU power, GPU power, and FPS in learning set

	Farmville	Contract Killer	RoboCop	Dark Meadow	Revolt	AVP	Asphalt	I, Gladiator	Call of Dead	B&G
P_C	4.96%	4.70%	9.49%	6.16%	4.59%	5.63%	6.37%	4.86%	5.60%	13.98%
P_G	5.23%	15.12%	11.96%	11.65%	14.28%	6.16%	5.80%	3.96%	6.83%	5.84%
Q	0.26%	0.21%	5.23%	5.48%	4.58%	0.02%	0.09%	7.64%	4.90%	1.02%

Table 4: Prediction errors for CPU power, GPU power, and FPS in test set

5. POWER MANAGEMENT

We now utilise the power-performance models to propose a power manager for mobile games running on HMPSoCs. *The objective of the manager is to achieve the target FPS with minimal total power consumption using CPU-GPU DVFS.*

In this work, we aim for maximum FPS, but our manager is capable of targeting any FPS. It has been already shown in [4] that considerable power saving can be achieved by reducing FPS in games. Further, it was observed in [4] and [5] that for same FPS, it is more power efficient to run at higher utilisation and lower frequency than lower utilisation and higher frequency. Power models for our platform also make the same prediction for both CPU and GPU irrespective of the games. Therefore, we design a power manager that tries to achieve maximum FPS with CPU and GPU in state of highest utilisations.

A game is composed of series of playable scenes. Power management is initialised at the start of every scene in the game. Start of a scene can be detected by changes in rendered textures [5] or CPU-GPU utilisation patterns [4].

When a new scene starts, we take three samples (each of one second duration) to obtain the game specific upper bound constants $\hat{U}_C$, $\hat{U}_G$ and $\hat{Q}$. Sampling for these constants has minimal impact on user's experience because a scene lasts quite long [4] and hence the sampling cost can be amortised over the scene duration. On the other hand, this sampling greatly enhances the accuracy of the algorithm and enables us to avoid any prior offline profiling.

Initially, in the model we use coefficients obtained from our learning set. This results in some prediction inaccuracy to begin with, however the effect on a user's experience is negligible because even with error, we are often very close to the target. As the scene progresses, we continue to sample instantaneous values to get new data. This data is then used online to further refine the coefficients using regression and make the models tailored to the scene being currently rendered. The runtime regression has very low overhead of approximately 0.08% additional CPU utilisation. Gaming, being a dynamic workload, can change in complexity and thereby requiring very different DVFS setting. The updating of our model in background also helps us to adapt to these changes.

5.1 Power Management Algorithm

Now we present our power management algorithm. There are two possibilities at current frequency combination (F_C, F_G). We can be either below the maximum FPS ($Q^{(F_C,F_G)} < \hat{Q}$) or at it ($Q^{(F_C,F_G)} = \hat{Q}$).

Meeting Performance Demand. If $Q^{(F_C,F_G)} < \hat{Q}$, then either CPU is the bottleneck ($U_C^{(F_C,F_G)} = \hat{U}_C$) or GPU is the bottleneck ($U_G^{(F_C,F_G)} = \hat{U}_G$). We need to increase the bottlenecked component's frequency to increase FPS. Let the required frequency combination be (F'_C, F'_G), where $F'_C \geq F_C$ and $F'_G \geq F_G$.

If CPU is the bottleneck, we choose F'_C using Equation (16), derived from Equation (10).

$$F'_C = \frac{\hat{Q} - Q^{(F_C,F_G)}}{\gamma_1} + F_C \tag{16}$$

This increase in FPS forces GPU to do more work, increasing its utilisation. Also, it may happen that we may fail to achieve $\hat{Q}$ even after increasing CPU frequency because of an intermediate GPU bottleneck. The estimated U_G at $\hat{Q}$ would be given by the Equation (17), based on Equation (15).

$$U_G^{(F'_C,F_G)} = U_G^{(F_C,F_G)} + \beta_4(\hat{Q} - Q^{(F_C,F_G)}) \tag{17}$$

GPU will become a bottleneck if $U_G^{(F'_C,F_G)} > \hat{U}_G$ and in that case, we have to increase the GPU frequency to F'_G given by Equation (18), derived from Equation (13).

$$F'_G = \frac{U_G^{(F'_C,F_G)} - \hat{U}_G}{\beta_3} + F_G \tag{18}$$

Similarly, if GPU was the bottleneck to begin with, then we would have evaluated F'_G first using Equation (11). We would have checked for a possible CPU bottleneck using Equation (14) and if required set it to a higher frequency F'_C, found using Equation (12).

Saving Power. Now, if $Q^{(F_C,F_G)} = \hat{Q}$, then we are achieving maximum FPS, but we may be wasting power by being at higher frequency combination than required.

While achieving FPS, if CPU or GPU is not operating at its maximum utilisation ($U_C^{(F_C,F_G)} \neq \hat{U}_C$) or ($U_G^{(F_C,F_G)} \neq \hat{U}_G$) then we can save power by reducing CPU or GPU frequency to F''_C or F''_G, respectively. We can obtain F''_C and F''_G from the Equation (19) and (20), derived from Equation (12) and (13), respectively.

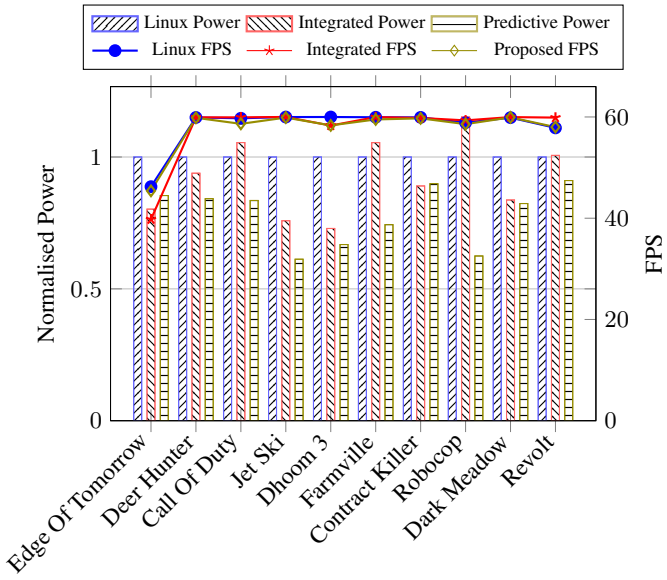
$$F''_C = \frac{\hat{U}_C - U_C^{(F_C,F_G)}}{\alpha_3} + F_C \tag{19}$$

$$F''_G = \frac{\hat{U}_G - U_G^{(F_C,F_G)}}{\beta_3} + F_G \tag{20}$$

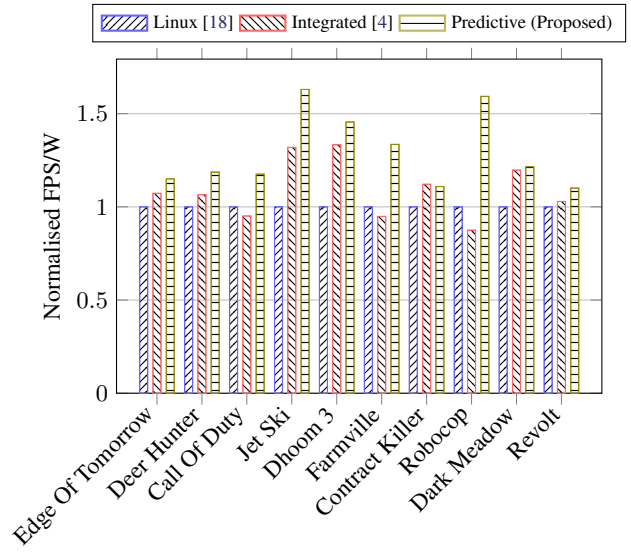
5.2 Comparative Evaluation

We test our manager on all the games but report results for ten games, with five games each chosen from our learning- and test set, due to space limitations. We compare our proposed manager (*Predictive*) with the default onboard CPU-GPU power managers (*Linux*) [18] and the state-of-the-art (*Integrated*) power manager for mobile games [4]. Please refer to Section 1 for discussion on these comparative baselines.

Figure 4(a) plots the average FPS achieved and corresponding



(a) Power Comparisons



(b) FPS/Watt Comparisons

Figure 4: Results of operating games with different managers.

normalised power consumption while running the games with three different power managers on Odroid-XU+E platform [3]. The three managers achieve almost equivalent performance in all games but with different power consumptions. As the performance is almost identical, FPS/Watt metric clearly quantifies the power-efficiency of the different managers. Figure 4(b) plots the respective normalised FPS/Watt. Figure shows that on average, our *Predictive* manager is able to achieve 29% and 20% higher FPS/Watt compared to *Linux* [18] and *Integrated* [4] managers, respectively.

6. CONCLUSION

In this work, we characterised the relationship of mobile gaming workload performance and power consumption with CPU-GPU DVFS on a real-world mobile platform with a HMPSoC. Based on our observations, we developed power-performance models using linear regression that captured the complex dynamics involved. Next, we used our models to design a power manager. We implemented the proposed manger on Odroid-XU+E [3] platform. Measurements on the platform show that our manager provides on average 20% higher power-efficiency compared to state-of-the-art [4]. The improved power efficiency will allow mobile platforms to operate games for longer durations.

7. ACKNOWLEDGEMENT

This work was partially supported by Singapore Ministry of Education Academic Research Fund Tier 2 MOE2012-T2-1-115.

8. REFERENCES

[1] Exynos 5 Octa (5422). www.samsung.com/exynos/

[2] M. Shafique, et al. Dark silicon as a challenge for hardware/software co-design. In *ISSS*, 2014.

[3] M. Hahnel and H. Hartig. Heterogeneity by the numbers. In *HotPower*, 2014.

[4] A. Pathania, et al. Integrated CPU-GPU power management for 3D mobile games. In *DAC*, 2014.

[5] B. Dietrich and S. Chakraborty. Lightweight graphics instrumentation for game state-specific power management in Android. In *Multimedia Systems*, 2014

[6] J. Park, et al. Quality-aware mobile graphics workload characterization for energy-efficient DVFS design. In *ESTIMedia*, 2014.

[7] X. Ma, et al. Characterizing the performance and power consumption of 3D mobile games. In *IEEE Computer*, 2013

[8] J. Leng, et al. Gpuwattch: Enabling energy optimizations in GPGPUs. In *ISCA*, 2013.

[9] A. Maghazeh, et al. General purpose computing on low-power embedded GPUs: Has it come of age? In *SAMOS*, 2013

[10] P. Greenhalgh. Big.little processing with Arm Cortex-A15 & Cortex-A7. An *ARM White paper*, 2011.

[11] B. Anand, et al. "Game action based power management for multiplayer online game." In *MobiHeld*, 2009.

[12] D. Zhang-Jian, et al. Power estimation for interactive 3D Game using an efficient hierarchical-based frame workload prediction. In *APSIPA ASC*, 2009

[13] X. Ma, et al. Statistical power consumption analysis and modeling for GPU-based computing. In *HotPower*, 2009.

[14] Y.Gu. Power management for interactive 3D games. *PHD Thesis*, 2008.

[15] K. Moiseev, et al. Timing-aware power-optimal ordering of signals. In *TODAES*, 2008.

[16] M. Claypool, et al. The effects of frame rate and resolution on users playing first person shooter games. In *Electronic Imaging*, 2006.

[17] T. Mudge. Power: A first class design constraint for future architectures. In *HiPC*, 2000.

[18] V. Pallipadi and A. Starikovskiy. The ondemand governor. In *Linux Symposium*, 2006.

[19] G. Semeraro, et.al. Energy-efficient processor design using multiple clock domains with dynamic voltage and frequency scaling. In *HPCA*, 2002.

[20] T. Mitra and T.Chiueh. Dynamic 3d graphics workload characterization and the architectural implications. In *MICRO*, 1999.