Integrated CPU-GPU Power Management for 3D Mobile Games

Anuj Pathania, Qing Jiao, Alok Prakash, and Tulika Mitra
School of Computing, National University of Singapore
{pathania,jiaoqing,alok,tulika}@comp.nus.edu.sg

ABSTRACT

Modern system-on-chips (SoC) integrate CPU and GPU for immersive 3D gaming experience. These games require both the CPU and GPU to work in tandem, resulting in high power consumption. In the past, Dynamic Voltage Frequency Scaling (DVFS) has been exploited for embedded CPU to save power during game play; but it is only recently that embedded GPUs have attained DVFS capabilities that provide additional opportunities. In this paper, we propose a power management approach that takes a unified view of the CPU-GPU DVFS, resulting in reduced power consumption for latest 3D mobile games compared to an independent CPU-GPU power management approach.

Categories and Subject Descriptors

C.1.4 [**Parallel Architectures**]: Mobile processors; D.4.7 [**Organization and Design**]: Real-time systems and embedded systems

General Terms

Algorithms, Design, Performance

Keywords

Embedded GPU, Power Management, 3D Mobile Games

1. INTRODUCTION

Multiprocessor system-on-chips (MPSoC) in high-performance mobile platforms supporting consumer electronic devices have witnessed unprecedented advances over the past decade. Current generation mobile SoCs consolidate heterogeneous processing elements such as high-performance CPU, GPU, DSP blocks on a single chip. Figure 1 shows the simplified block diagram of the recent Samsung Exynos 5410 Octa SoC, that powers Samsung Galaxy S4 devices. Qualcomm's Snapdragon and AMD's A-series APU are other examples of platforms with integrated CPU and GPU on a single chip.

The integration of the powerful 3D graphics capable GPU with the CPU cores on the same chip enables sophisticated real-time 3D gaming experience for the consumers on such mobile platforms.

DAC '14, June 01 - 05 2014, San Francisco, CA, USA
Copyright 2014 ACM 978-1-4503-2730-5/14/06 ...$15.00
http://dx.doi.org/10.1145/2593069.2593151.

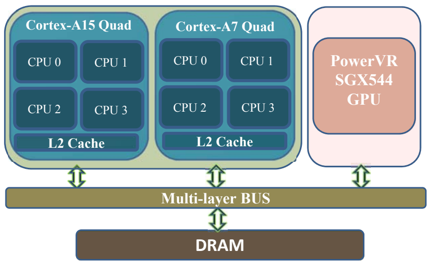


Figure 1: Exynos 5 Octa SoC simplified block diagram.

However, 3D games are highly demanding of computational resources as well as memory bandwidth on both the CPU and the GPU. This is because while the GPU supports 3D rendering of a scene, the CPU builds up the scene using complex game physics or smart artificial-intelligence based strategies. The compute- and memory- intensive nature of the 3D games translates to substantially high power consumption in the mobile platforms, resulting in poor battery life. Figure 2 shows the power consumption of the ARM Cortex-A15 CPU cluster and the PowerVR GPU on Exynos 5410 Octa SoC for a popular Android game *"Asphalt 7: Heat"* over 2-minute lifetime. The figure clearly shows that both the CPU and the GPU contribute to the power consumption during gaming. Thus, power management of both the CPU and the GPU is a first-class design priority in all high-end mobile platforms.

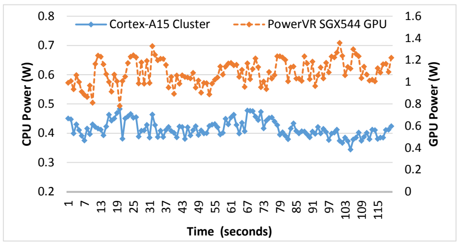


Figure 2: CPU-GPU power behavior for Asphalt 7 game.

In this work, we focus on system level runtime power management for integrated CPU-GPU system-on-chip devices. Modern operating systems, such as the Linux kernel in Android, include simple governors to perform power management through dynamic frequency and voltage scaling (DVFS) of the CPU. The latest integrated GPU cores, emerging in the embedded SoCs also offer DVFS capability. However, the GPU power management is typically achieved through GPU-specific firmware. In other words, there is no synergy between the CPU and GPU power management.

In this paper, we first argue through quantitative characterisation of a set of 3D gaming workloads that an integrated power management framework, where the CPU-GPU frequency levels are scaled in a synergistic fashion, is essential to achieve satisfactory user experience at minimal energy levels.

A CPU-GPU integrated power management framework has to identify the bottleneck (CPU or GPU) and take actions accordingly through the DVFS knobs. However, an interactive 3D game is a highly dynamic workload demanding multiple different CPU-GPU frequency settings over the lifetime of a game play. In this work, we develop an efficient integrated power management framework that perform DVFS at runtime to save power while providing users with a stable performance during game execution. We establish the superiority of our integrated management approach over independent CPU-GPU power management through quantitative evaluation with popular high-end mobile 3D games on a real platform.

Previous work has demonstrated [11] that the frames per second (FPS) is the key metric that contributes to the gaming experience. Current Android platforms attempt to run a game at the highest possible FPS level leading to quick battery drain. However, most of the games can be played quite satisfactorily at much lower FPS level. We can either allow the user to set the expected FPS for each gaming session or target FPS can be set transparently by the OS based on the remaining battery life and a simple user preference profile. Our power management framework could maintain an FPS level for majority of the execution and help save power.

The concrete contributions of this paper are the following.

- We perform 3D gaming workload characterisation on mobile SoCs with integrated CPU-GPU to analyse the power-performance behaviour. This analysis provides us with the insights required to design the power management solution.
- To the best of our knowledge, ours is the first work that explores CPU and GPU DVFS in tandem to provide a simple yet powerful power management approach for 3D mobile games on integrated CPU-GPU platforms.
- We implement our power management technique by modifying the Linux kernel on Android platform and evaluate its effectiveness with latest high-end mobile 3D games.

2. RELATED WORK

The power management for CPU-GPU heterogeneous system-on-chip architectures have so far primarily focused on the general-purpose computing applications [8] [9] and not on 3D gaming workload. Only recently [14] conducted a performance and power consumption characterisation of 3D mobile games on three mainstream mobile heterogeneous system-on-chips. However, they did not consider power saving strategies for 3D games.

Few works in the literature studied power management and performance characterisation of 3D games. [12] performed a detailed dynamic workload characterisation of 3D applications. [10] built a simulator based on hypothetical GPU architecture to analyse 3D graphic application performance bottleneck and power consumption. In [6], [3], the authors observe that 3D graphics applications show significant variation of workload with different configurations such as level of detail, resolution, texture mapping and lighting and are amenable to potential power saving by employing dynamic voltage and frequency scaling. However, they either employ CPU or an emulator of the GPU pipeline for evaluation. [4] [7] proposed DVFS technique based on workload prediction. However, none of them target the modern embedded GPU.

In contrast to the above works, we target a modern SoC with integrated CPU and mainstream embedded PowerVR GPU. We consider the 3D gaming workloads using both CPU & GPU and based on their characterisation, we propose our integrated power management strategy.

3. GAME POWER-PERFORMANCE CHARACTERISATION

We first characterize the power-performance behaviour of contemporary high-end 3D mobile games at different voltage-frequency levels for the CPU, GPU, and the memory. This application behaviour will lay the foundation of the design choices we make for our power management algorithm in the subsequent section.

Experimental setup. We use an Odroid-XU+E board [1] running Android 4.2.2v with Kernel 3.4.5v for our experiments. The board contains Exynos 5410 Octa chip with Quad Core ARM Cortex A15 and Quad Core ARM Cortex A7 CPU clusters along with PowerVR SGX544MP3 GPU and 2GB LPDDR3 RAM as shown in Figure 1. As the focus of this work is on CPU-GPU interaction, we only use the A15 cluster for our experiments. We plan to extend this work to consider migration of the CPU workload between the A7 and the A15 cluster, along with CPU-GPU DVFS in the immediate future.

The board offers DVFS capabilities for CPU, GPU and main memory. The CPU can be operated at nine frequency levels 800, 900, 1000, 1100, 1200, 1300, 1400, 1500 & 1600 MHz, while the GPU is capable of operating at six frequency levels 177, 266, 350, 480, 532 & 640 MHz. The memory subsystem has two operating frequency levels 400 & 800 Mhz. Thus, we have a total of 108 different combinations of DVFS configurations, rendering exhaustive exploration of the design space infeasible.

The board provides four power sensors, one each for the A7 cluster, the A15 cluster, the GPU, and the DRAM main memory. The power sensors can be sampled at 4Hz to obtain continuous power readings for the different on-chip components. We also sample the CPU utilisation, the GPU utilisation and the FPS (frames per second) directly from the kernel.

We present the characterisation results of the main game play scene from a top racing Android game *"Asphalt 7: Heat."* We choose a fast changing, 3D graphics intensive scene of a car racing along the track. The complexity of the scene provides us with an opportunity to demonstrate the full of range of possible power-performance behaviour.*We observe similar behaviour for many other high-end 3D games in the Android platform even though the actual power values and the FPS might differ from game to game. Indeed, we evaluate our power management framework with a large number of contemporary games in Section 5.*

To demonstrate the power-performance behaviour of the game across the 108 different DVFS configuration points, we need to reproduce the exact workload every time. But we are not aware of any existing mechanism that can record and replay the exact game play on an Android platform. We observe, however, that in Asphalt 7, once a track is loaded, the car can race along the track without any user input on a deterministic path. We play the car racing scene for 108 different configuration points where each game play lasts for two minutes and profile the power-performance behaviour. At this point, we are interested in the power-performance at different DVFS levels averaged over the lifetime of each game play.

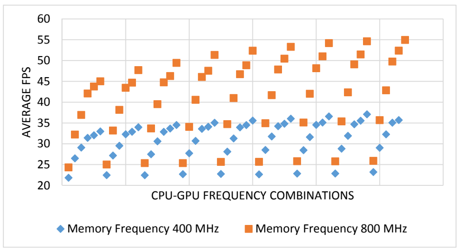


Figure 3: Impact of memory DVFS on FPS

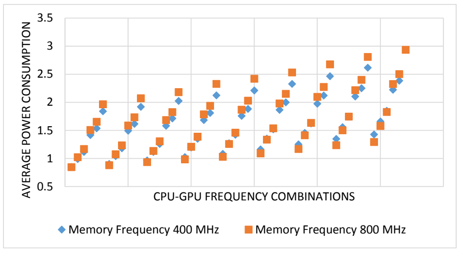


Figure 4: Impact of memory DVFS on total power

Impact of memory frequency scaling. We first investigate the impact of the memory subsystem frequency on the FPS and the total power as shown in Figure 3 & 4, respectively. Note that the memory bandwidth in integrated platforms is shared between the CPU and the GPU. The X-axis corresponds to different CPU-GPU frequency combinations. For each CPU-GPU frequency combination, we plot the average FPS (or average total power) at two different memory frequency levels. As expected, the game performance (in terms of FPS) for a given CPU-GPU frequency combination is always substantially higher at 800MHz memory frequency due to increased memory bandwidth compared to 400MHz frequency.

However, it is interesting to observe that the total power remains almost the same irrespective of the memory frequency. In fact, at certain CPU-GPU frequency combinations, reducing the memory bandwidth leads to increase in the total power consumption. This behaviour can be explained by considering the effect of memory DVFS on the individual units. When memory clock frequency is reduced, CPU utilisation increases as the CPU spends more time in active state waiting for memory responses, increasing its power consumption [2]. GPU utilisation, on other hand, is severely reduced as it receives less data to render from the CPU and also gets reduced bandwidth from the shared memory due to contention with the CPU. This reduced GPU utilisation decreases its power consumption. Memory, as expected, observes a reduction in power consumption as clock frequency is decreased. But the reduction in power consumption of the GPU and the memory cannot always compensate for the increased power consumption of the CPU, leading to increased total power with reduced memory frequency. This makes memory DVFS unattractive for saving power. Therefore, in the subsequent analysis, we set the memory frequency at 800MHz.

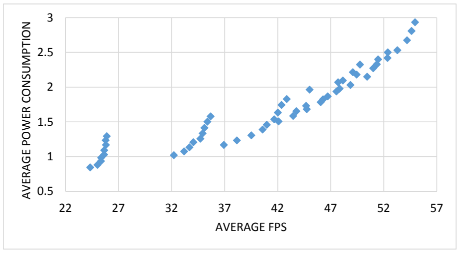


Figure 5: Average FPS vs Average Total Power

Power-Performance trade-off. Our goal in this work is to offer the expected gaming performance (FPS) at minimal power through DVFS. So we first analyse the relationship between power and performance. Figure 5 plots the average power and FPS for each of the CPU-GPU frequency combinations at 800MHz memory frequency. The obvious conclusion is that we can save significant energy if we play at reduced FPS. The interesting observation from this plot, however, is that we can achieve nearly the same level of performance with very different power profiles. The challenge therefore is to identify the appropriate frequency levels that offer the required FPS using minimal power. So we now need to understand how changing CPU-GPU frequency impacts performance and power individually.

Impact of CPU-GPU frequency on performance. We focus on the impact of CPU-GPU DVFS on the gaming performance. It is not easy to isolate the effect of CPU or GPU frequency scaling as they are interdependent through their producer-consumer relationship. For example, reducing CPU frequency leads to less work for the GPU leading to reduced GPU utilisation and hence reduced FPS. Increasing the GPU frequency here will have no impact on the FPS. In this context, we cannot simply consider the relationship between the GPU frequency and the FPS. Instead, we need to consider both the frequency and the utilisation. So we employ the concept of *CPU Cost*[2] and *GPU Cost*. The CPU cost is defined as the product of the CPU utilisation and its frequency. The GPU cost is defined similarly w.r.t. GPU utilisation and its frequency.

Figure 6 plots the average FPS and average CPU Cost for the different DVFS configuration points. In this figure, we distinguish among the configuration points with different GPU frequency levels. For example, consider GPU frequency of 640 MHz (circular markers). At this GPU frequency level, we vary the CPU frequency and compute the average cost and FPS at each CPU frequency level (total of 9 points). We observe near-linear increase in FPS with increasing CPU Cost. In general, the correlation between the average CPU Cost and the average FPS is 0.94, indicating near-linear relationship between the two.

However, there are some exceptions. For example, consider the lowest GPU frequency at 177 MHz (square markers). In this case, the GPU is the bottleneck; therefore, as the CPU frequency is increased, FPS remains almost the same. Similarly, a restricted memory bandwidth, refresh rate restriction on maximum FPS (60 frames per sec), and possible internal FPS control by game developers can all contribute to exception cases where the linear relationship between CPU Cost and FPS may not hold good.

Figure 7 plots the average FPS and average GPU cost for different DVFS configuration points. The correlation between the average GPU Cost and the average FPS is quite high at 0.98. The effect of CPU bottleneck is much less apparent here because the scene imposes more demand on the GPU than the CPU.

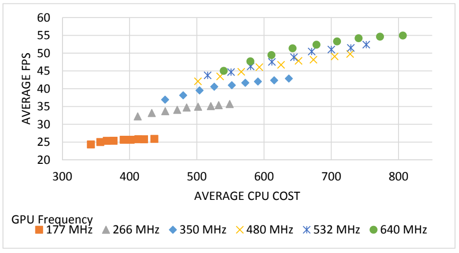


Figure 6: Average FPS vs Average CPU Cost

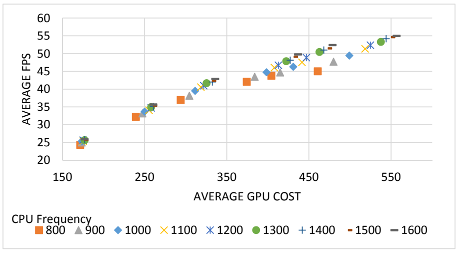


Figure 7: Average FPS vs Average GPU Cost

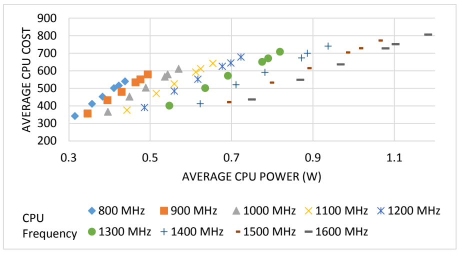


Figure 8: Average CPU Power vs Average CPU Cost

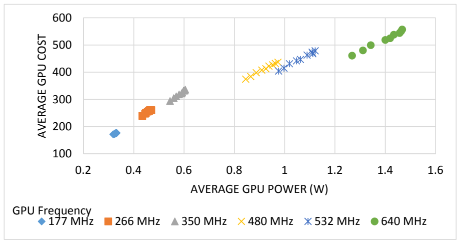


Figure 9: Average GPU Power vs Average GPU Cost

Impact of CPU-GPU DVFS on power. We have established that, in general, we need to increase the GPU and CPU costs to increase the FPS level. However, CPU cost is a product of utilisation and frequency. Thus, multiple different frequency levels can lead to the same CPU cost depending on the utilisation. Hence, we investigate the power behaviour of the design points with the same CPU cost.

Figure 8 plots the average CPU cost and the average CPU Power for different configuration points. For the same frequency level, higher CPU utilisation (due to increasing GPU frequency) leads to higher CPU power consumption. More importantly, for the same CPU cost, the power increases with increasing CPU frequency. Thus, it is beneficial to choose the design point with lower frequency and higher utilisation rather than higher frequency and lower utilisation to pay the required CPU cost. Figure 9 shows similar observations between average GPU cost and average GPU power.

Game dynamism. So far, we have focused on the averaged power-performance of a game play. However, gaming workloads can exhibit highly dynamic characteristics. We analyse the game dynamics to decide when and how to perform power management.

A game is composed of a set of scenes with which the user interacts and during game play *different scenes* can demand very different processing costs (CPU costs and GPU costs) depending on the scene's complexity. At the same time, the complexity also varies *within a scene* due to user interactions and game dynamics.

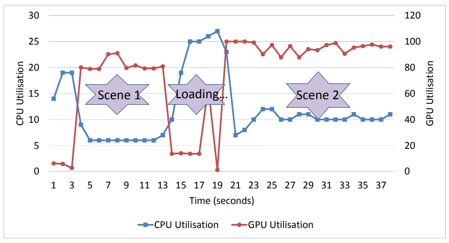


Figure 10: Scene change detection using CPU-GPU utilisation

We notice that scene changes do not happen often during game play and can be easily detected from CPU-GPU utilisation. Figure 10 plots the CPU and GPU utilisation as the sample racing game proceeds from the welcome screen to car selection to the racing track. Before a scene is rendered by the GPU, the CPU has to first create the scene that requires substantial computational resources. During this time, the GPU generally stays idle. So a scene change can be detected by concurrent sharp increase and decrease in the CPU and GPU utilisation, respectively. Similarly, sharp decline in CPU utilisation combined with sharp increase in GPU utilisation indicates completion of the scene creation by the CPU. In Figure 10, three scene changes are detected and marked on the timeline.

Scene changes do not happen often because the game cannot be played when a scene is loading; sometimes developers are forced to show a loading screen. To avoid such disruption, the scene environment is often large and created at once. In general, a user spends substantial amount of time (in minutes) in each main game scene. For example, in our illustrative racing game, once the track (scene) is loaded, the car makes multiple laps lasting for several minutes around the track without any scene change. We observe this behaviour in multiple games and draw an assumption that the main scenes in games can be expected to be long running.

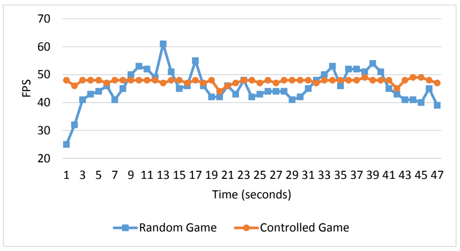


Figure 11: Impact of user interaction on FPS during a scene.

The dynamism within a scene, however, is completely unpredictable due to random user interaction and closed-source nature of commercial games. In the literature, several DVFS techniques have been proposed for games based on workload prediction [4, 7]. We first investigate whether workload prediction can be effective for modern high-end interactive 3D games. Figure 11 plots the instantaneous FPS over time for two runs of the same scene with the same DVFS settings. In the random run, the racing game is played aggressively with car bumping into fences and other cars, while in the controlled run the car is driven carefully in the center of the road without any collisions. Clearly, the FPS is very stable in the controlled run; but varies without any pattern in the random run. Unlike videos, it is very difficult to predict such interactive behaviour during game play. Only the after-effect of such an interaction can be observed through the resulting FPS. Therefore, we develop a reactive rather than predictive power management approach.

4. INTEGRATED POWER MANAGEMENT

We now proceed to present our power management algorithm for 3D mobile games derived on the observations in the previous section. The proposed algorithm exploits CPU-GPU DVFS capabilities to achieve the target FPS range with minimal power.

As mentioned earlier, we design a reactive power management technique due to the difficulty in predicting the future workload in a highly interactive 3D game. Moreover, unlike previous work [5] that proposes per-frame DVFS, we perform frequency scaling, if necessary, only at per-second granularity. This is because frequent DVFS (2400 DVFS per minute for 40 FPS) may lead to hardware failure due to thermal cycling [13].

We also observe that for highly dynamic and demanding scenes, maintaining the FPS at a fixed value (e.g., 30 FPS) may lead to frequent DVFS and hysteresis. Instead, we define performance as an FPS range (e.g., 30–35 FPS). We attempt to maintain the performance within the FPS range averaged over a sliding window of 5 sec. The 5 sec window is chosen as game players cannot notice any observable difference in performance even if the instantaneous FPS varies within this 5sec interval.

The proposed algorithm can be summarized as follows. The algorithm begins with the lowest CPU and GPU frequency at the start of a scene. In case the desired FPS range cannot be met at the lowest CPU-GPU frequency, the current CPU and GPU costs are evaluated. Using the current costs, algorithm then extrapolates the estimated CPU-GPU frequency that is sufficient to achieve the desired FPS range; the process is repeated till the target is met. Once the target FPS range is achieved, algorithm tries to maintain this FPS by only varying CPU frequency, as given the high sensitivity of FPS to GPU frequency, changing GPU frequency will cause current FPS to move out of the target range.

4.1 Algorithm

We first construct a cost-performance model for integrated CPU-GPU system that is later used to explain the proposed algorithm.

Cost Model at Current Setting. Let the tuple (c, g) represent frequency setting combination when CPU and GPU are set at frequency level c and g, respectively. Let Q_{min} and Q_{max} represent the maximum and minimum values for the target FPS range. $UC^{(c,g)}$, $UG^{(c,g)}$ and $Q^{(c,g)}$ represent the observed averaged CPU utilisation, GPU utilisation and FPS, respectively at frequency setting (c, g). We sample the utilisation and FPS once per second.

As defined earlier, CPU and GPU cost paid at a particular CPU-GPU frequency combination is the product of their respective frequency and utilisation. The current CPU and GPU costs can be considered the payment required to generate the current FPS $Q^{(c,g)}$.

We define $PC^{(c,g)}$ and $PG^{(c,g)}$ as the price paid by CPU and GPU to produce unit FPS at (c, g) frequency setting. $PC^{(c,g)}$ and $PG^{(c,g)}$ together represent the minimum cost required to produce unit FPS at the current settings.

$$PC^{(c,g)} = \frac{UC^{(c,g)} \times c}{Q^{(c,g)}} \qquad PG^{(c,g)} = \frac{UG^{(c,g)} \times g}{Q^{(c,g)}}$$

Extrapolation. Let (c, g) be the current frequency setting that achieves an FPS less than the minimum (or more than the maximum) specified FPS range. To achieve higher (or lower) FPS, the CPU-GPU frequency must be increased (or decreased).

Let Q be the target FPS that we wish to achieve. Let OC & OG be the expected CPU and GPU cost that are sufficient to achieve the target FPS Q. We can estimate OC & OG based on the near-linear relationship observed between CPU (GPU) cost and FPS (see Figure 6 and Figure 7).

$$OC = PC^{(c,g)} \times Q \qquad OG = PG^{(c,g)} \times Q$$

As we increase (or decrease) frequency, the CPU-GPU utilisation values typically drop (or rise). So the current utilisation values can serve as upper (or lower) bound for utilisation values expected at higher (or lower) frequencies. Thus, the maximum (or minimum) expected CPU-GPU cost at a higher (or lower) CPU and GPU frequency (c', g') represented by $\overline{OC}^{c'}$ & $\overline{OG}^{g'}$ can be computed as:

$$\overline{OC}^{c'} = c' \times UC^{(c,g)} \qquad \overline{OG}^{g'} = g' \times UG^{(c,g)}$$

When we need to improve the FPS, we set the target $Q = Q_{max}$ and look for higher frequency levels. We choose the lowest CPU frequency level c', such that $\overline{OC}^{c'} \geq OC$. Similarly, we choose the lowest GPU frequency g' such that $\overline{OG}^{g'} \geq OG$. In other words, we choose the minimum CPU-GPU frequency level where we expect the required cost to be satisfied with maximum utilisation. When we need to lower the FPS, we set $Q = Q_{min}$ and look for lower frequency levels. We again choose the lowest CPU-GPU frequencies that just satisfy the target cost. If the performance is still outside the target FPS range after DVFS, we continue our extrapolation with the new utilisations and FPS values. We employ a conservative cost model and always strive to run at the bare minimum CPU-GPU frequency level to save as much power as possible.

Maintenance Mode. Once the desired FPS range is achieved through extrapolation, it is essential to maintain the FPS within the target range. The FPS is highly sensitive to GPU frequency scaling (see Figure 6). Thus, we avoid scaling the GPU frequency and instead rely on CPU frequency scaling to keep the instantaneous FPS within range. When we observe that the instantaneous FPS sampled at one second interval falls outside the range, we increase (or decrease) the CPU frequency to bring it back within the desired range. As mentioned earlier, instantaneous FPS sampled at a 1 sec interval may fall outside the range; but it does not have a major impact on user experience. However, if the average FPS over 5sec sliding window falls outside the target range even with CPU frequency scaling, then we need to alter the GPU frequency setting. In such scenarios, the algorithm starts performing extrapolation again.

Scene Transitions. The extrapolation and maintenance modes continue alternatively till a scene change is detected (Figure 10). At this point, the CPU is immediately set to run at maximum frequency and the GPU at minimum frequency to quickly complete the loading phase. Once the scene loading is finished, the extrapolation and maintenance are triggered again starting from the minimum CPU-GPU frequency levels. This strategy improves the user experience as waiting time is reduced during scene loading.

5. EXPERIMENTAL RESULTS

In this section we compare the proposed integrated approach against the independent Linux CPU-GPU power management solution used in Android platforms. We implement our integrated power management framework in the Linux kernel on Android platform. The independent power management approach consists of an on-demand governor for the CPU implemented in Linux kernel and a custom firmware-controlled DVFS management for the GPU that works independently. The independent power management approach in Linux kernel does not take into account the gaming performance and hence cannot respond to FPS.

Apart from Asphalt 7 mentioned earlier, we select several other high-end popular Android games: *Anomaly 2, Call of Duty: Final Strike, Need for Speed Most Wanted, Real Football 2013, AVP. Evolution*. To compare our integrated approach with Linux, we should ideally be able to record and replay identical game play. However, as mentioned earlier, we are not aware of any available record and replay mechanism for 3D games on Android platform. To overcome this limitation, we requested volunteers to play the main level of the games with repetition of their game-play as far as possible. For each game, a volunteer played it 5 times on both off-the-shelf Linux and our integrated approach. The results presented are averaged across 5 runs. We first present detailed results for *Asphalt 7* followed by experimental results for the other games.

Figure 12 shows the total power consumption of the proposed integrated approach against Linux for various FPS target ranges. The minimum and maximum lines represent the lower and upper bounds of power consumption with minimum and maximum frequency settings. The power consumption for Linux, minimum, and maximum remains unchanged across FPS range settings. The figure shows that the Integrated approach can provide significant performance-power trade-off capability. To compare the power consumption by the proposed approach against Linux, we observed the performance (FPS) achieved by these approaches as shown in 13. Comparing Figure 12 with Figure 13. it is evident that

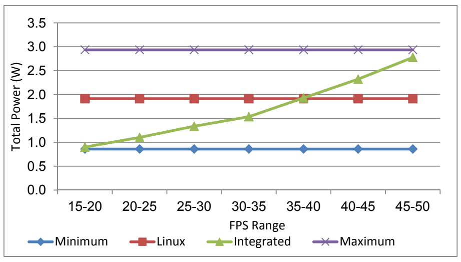


Figure 12: Total power for Asphalt 7 at different specified ranges for Linux and integrated approach.

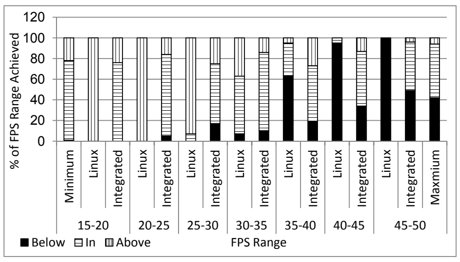
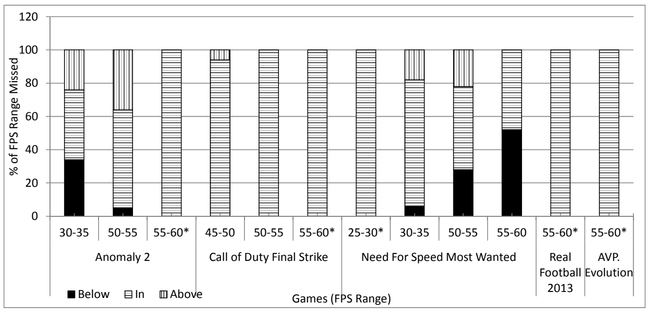
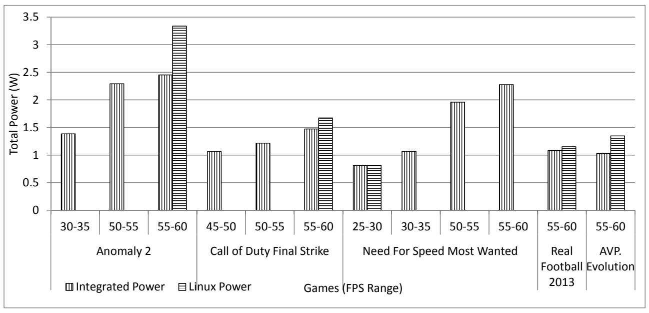


Figure 13: The achieved FPS (below, within, or above the specified range) for Asphalt 7 with Linux and integrated approach.

for most FPS ranges (e.g., 30–35), we attain similar performance as Linux with significantly lower power consumption. This is achieved by operating closer to the possible minimal CPU-GPU cost and lowest CPU-GPU frequency necessary. We begin to consume more power than Linux as we try to achieve higher performance than Linux can offer.



(a) FPS (below, within, or above the specified range) with integrated approach.



(b) Total power consumption at different target FPS range.

Figure 14: Linux vs Integrated Power Management for Games

The Linux power management approach does not consider target FPS range for DVFS and hence the achieved FPS is distributed over a wide spectrum. On the other hand, the proposed integrated approach responds to the target range of FPS and succeeds to keep FPS within this range for longer durations. The discrete nature of CPU-GPU DVFS limits the ability to achieve a stable FPS range for all FPS values. This can be observed in Figure 13 where the proposed approach performs better for some ranges more than others.

Figures 14a and 14b show the achieved FPS and corresponding power consumption during game-play for a set of latest Android games. These games are selected from different genres based on their popularity. The Linux approach typically keeps the performance at the highest FPS range of 55-60 for most of the games tested except for *Need for Speed (NFS) Most Wanted* for which it stays at the lowest FPS range of 25–30. In Figure 14b, we plot the power consumption for Linux only at that achieved FPS range.

The integrated approach can achieve the same FPS range as Linux at significantly less power. Moreover, for games like *Anomaly 2* and *Call of Duty: Final Strike*, the integrated approach allows additional power saving at a lower, albeit acceptable, FPS range. On the other hand, for games like *NFS*, the integrated approach can improve performance significantly compared to Linux with additional power. Games like *Real Football 2013* and *AVP Evolution* achieve the highest FPS range at very low frequency settings, thereby leaving us with no scope for further FPS reduction. Still we manage to save power compared to the Linux.

The overhead of our approach is 2%, mostly due to the extraction of FPS information during game play. We plan to reduce the overhead through optimizations of the current naive implementation.

6. CONCLUSION

In this paper, we presented an integrated CPU-GPU power management approach for 3D mobile games. Empirical results on a real CPU-GPU platform with the latest Android games, show that we are able to provide the user with flexible power-performance tradeoff options that can reduce gaming power consumption by up to 58%. In comparison to the independent power management approach in current Android platforms, the proposed integrated approach is able to reduce power consumption of games by up to 26% for comparable FPS range.

7. ACKNOWLEDGEMENT

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

8. REFERENCES

[1] ODROID_XU–E. http://hardkernel.com/main/main.php.
[2] Y. Bai. Memory Characterization to Analyze and Predict Multimedia Performance and Power in an Application Processor. In *Marvell White Paper*. 2011.
[3] B. C. Mochocki et al. Power Analysis of Mobile 3D Graphics. In *DATE*, 2006.
[4] B. Dietrich et al. LMS-based low-complexity game workload prediction for DVFS. In *ICCD*, 2010.
[5] B. Dietrich et al. Managing power for closed-source android os games by lightweight graphics instrumentation. In *NetGames*, 2012.
[6] G. Yan et al. Games are up for DVFS. In *DAC*, 2006.
[7] G. Yan et al. A Hybrid DVS Scheme for Interactive 3D Games. In *RTAS*, 2008.
[8] H. Wang et al. Workload and Power Budget Partitioning for Single-chip Heterogeneous Processors. In *PACT*, 2012.
[9] I. Paul et al. Coordinated energy management in heterogeneous processors. In *SC13*, 2013.
[10] J. W. Sheaffer et al. A flexible simulation framework for graphics architectures. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, 2004.
[11] M. Claypool et al. The effects of frame rate and resolution on users playing First Person Shooter games. In *In Electronic Imaging*, 2006.
[12] T. Mitra et al. Dynamic 3d graphics workload characterization and the architectural implications. In *MICRO*, 1999.
[13] T.S. Rosing et al. Power and Reliability Management of SoCs. *VLSI*, 2007.
[14] X. Ma et al. Characterizing the Performance and Power Consumption of 3D Mobile Games. 2013.