

V-SYNC-Aware GPU DVFS Governor for Efficient Game Application Execution on Mobile Devices

En-Ming Huang

Google, Inc. / National Taiwan University
emsamuelhuang@google.com

Yu-Fu Kao

Google, Inc.
lono@google.com

Yan-Hong Lu

Google, Inc.
dinolu@google.com

Chun-Yi Lee

National Taiwan University
cylee@csie.ntu.edu.tw

Abstract—The optimization of energy consumption in mobile devices, particularly for resource-intensive applications such as mobile gaming, remains a critical and pervasive challenge. This paper presents a novel V-SYNC deadline-aware GPU dynamic voltage and frequency scaling (DVFS) governor designed to address this challenge by optimizing energy usage during gameplay. Our approach leverages V-SYNC events as deadlines for GPU frequency adjustments, minimizing energy wastage while ensuring timely frame rendering. To implement this strategy, we utilize the Perfetto software development kit for real-time system tracing and hardware counter capturing. This innovative approach enables fine-grained dynamic adjustment of GPU frequencies based on workload demands. The experimental results conducted on the Google Pixel 8 Pro demonstrate the efficacy of our proposed DVFS governor in achieving significant energy savings, with a reduction in per-frame energy consumption of up to 3%. Furthermore, our method enhances frame rate stability by reducing frame per second (FPS) variation by 37%, substantially improving the overall user experience. These experimental results substantiate the effectiveness of our methodology in achieving a balance between energy efficiency and performance in mobile gaming.

Index Terms—Mobile Devices, GPU, V-Sync Events, DVFS.

I. INTRODUCTION

Video games represent one of the most compute-intensive applications on mobile devices. Driven by advancements in processors and graphics application programming interfaces (APIs), the increasing complexity of mobile games enhances realism and visual appeal through features such as higher-resolution textures and ray tracing. These advancements necessitate substantial computational resources from both the Central Processing Unit (CPU) and the Graphics Processing Unit (GPU). However, mobile devices face unique constraints compared to their desktop counterparts, including limited power sources and the absence of active cooling mechanisms. Consequently, energy consumption significantly impacts battery life and can lead to thermal throttling, both of which detrimentally affect user experience [1]. Furthermore, the frame rate, measured in frames per second (FPS), plays a crucial role in determining the quality of the gaming experience. Higher FPS corresponds to smoother visual motion, thus contributing to increased user satisfaction. As a result, achieving an appropriate balance between energy efficiency and maintaining a high frame rate is crucial to meet user expectations in gaming.

Game applications on mobile devices typically operate on a game loop that continuously updates the virtual environment. Within each iteration, the scene is updated according to user input, and the frame is subsequently rendered by the GPU. The

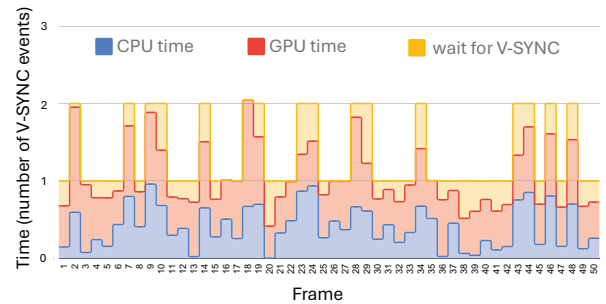


Fig. 1. Time distribution analysis for individual frames in a mobile game (Fortnite [2]), depicting CPU, GPU, and idle periods relative to V-SYNC events.

operating system then refreshes the screen to display the frame to the user. In an Android system, this screen update is triggered by a vertical synchronization (V-SYNC) periodic signal generated by a system process. Fig. 1 illustrates the time distribution of a typical Android game application. The rendering of each game frame encompasses three sequential stages: the CPU updates the game environment, the GPU renders the frame, and the system awaits a V-SYNC event to refresh the screen. This entire process necessitates the coordinated operation of the equipped CPU, GPU, and hardware I/O components.

The time distribution presented in Fig. 1 reveals a significant observation: 80% of the rendering workload concludes prior to the V-SYNC event, which results in underutilized CPU and GPU resources. To address this inefficiency, previous research [3]–[11] have extensively explored methods to decelerate the rendering speed to a predefined interval (e.g., 1/60 second for a 60Hz screen) by selecting lower operating frequencies and voltages through the dynamic voltage and frequency scaling (DVFS) function on electronic devices. This approach reduces idle time and energy consumption while delivering the same computational workload. Some previous strategies [3], [4], [6], [7], [11] focused on utilization and frequency metrics to predict and adjust the required frequency, while other researchers proposed to employ texture-directed [12] and system counter-based [13] techniques to estimate CPU and GPU execution times under different frequencies. In addition, some other works focused on the whole system, including the impact of multiprocesses and thermal throttling [14], [15], to ensure sustained performance. Nevertheless, these methods often adjust frequency at a coarse-grained level (e.g., once per 100 milliseconds). Moreover, although these previous approaches aim to adjust

CPU and GPU frequency to meet specific deadlines (e.g., 1/60 second) defined by themselves, they often overlook that the actual deadline is determined by the V-SYNC signal generated by the system process. Accounting for the V-SYNC signal as the deadline presents challenges due to the requirement for low-latency adjustments. For example, for a 60Hz refresh rate display, V-SYNC event occurs every 16.66 milliseconds, necessitating frequency adjustments at millisecond intervals.

To address the aforementioned challenge, this paper introduces the concept of “V-SYNC deadline”, which designates the V-SYNC event as the deadline for GPU job completion. We introduce a methodology for dynamic GPU frequency adjustment leveraging Android’s DVFS capability. Additionally, we showcase the ability to capture system trace events and hardware counters in real-time (e.g., at millisecond intervals) using Perfetto [16], Google’s performance tracing tool, allowing for frequency management with only 4 milliseconds of latency and without modifications to Android source code or GPU firmware. This facilitates future research in controlling the GPU frequency in a fine-grained manner. Our GPU DVFS governor, which is a module that determines the GPU operating frequency, initially captures the time when the GPU commences operation and estimates the appropriate frequency according to the time when V-SYNC occurs. In addition, we introduce a novel concept termed “delayed deadline”. This approach automatically extends the deadline to the second or third V-SYNC event if the time to the first V-SYNC event is insufficient for the GPU to complete the workload. This extension allows for the selection of a lower frequency, which reduces energy consumption while maintaining performance. To validate the proposed methodology, we conducted experiments on a Google Pixel 8 Pro device with an online first-person shooter game that features wide variations of moving scenes. The experimental results demonstrate that our proposed method achieves up to 3% GPU energy improvement per rendered frame with minimal performance impact. Moreover, our approach significantly reduces FPS variation by 37%, enhancing frame rate stability and user experience. The contributions are summarized as follows:

- 1) We propose a GPU DVFS strategy that utilizes the V-SYNC signal as a constraint for frequency optimization.
- 2) We introduce the concept of “delayed deadline”, enabling the selection of lower frequencies under constrained timing conditions to enhance the overall energy efficiency.
- 3) We demonstrate a real-time methodology for capturing system traces and hardware counters for a fine-grained DVFS controller, which has not been previously explored.

The remainder of the paper is organized as follows. Section II introduces the background and our motivation. Section III elaborates on the proposed methodology. Section IV presents the experimental results. Finally, Section V concludes the paper.

II. BACKGROUND AND MOTIVATION

This section introduces the GPU DVFS management system for Android devices and the workflow of common game applications, providing context for our proposed methodology.

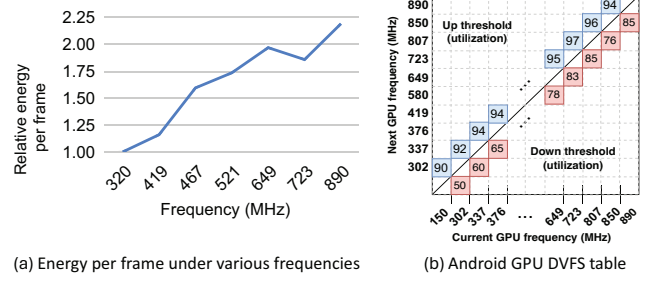


Fig. 2. (a) Energy per frame under various frequencies and (b) the default GPU DVFS table for the Google Pixel 8 Pro.

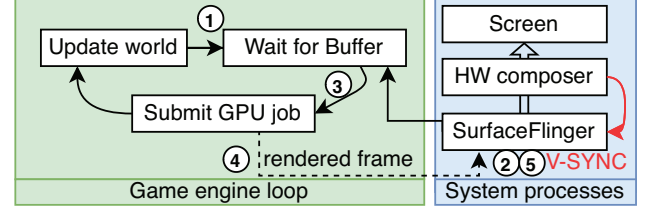


Fig. 3. An illustration of the conventional workflow for executing mobile game applications on the Android system.

A. Android GPU Frequency and Voltage Management

The GPU of an Android device typically supports multiple operating frequencies. The Google Pixel 8 Pro, for example, allows its GPU to function at thirteen distinct frequencies, spanning from 150 to 890 MHz. To ensure system stability, higher frequencies necessitate higher operation voltages, with the voltages for each frequency predefined by the vendor. The dynamic energy consumption can generally be modeled by $E = NumClocks \cdot CV^2$, where E represents energy consumption, $NumClocks$ denotes the number of clock cycles, C signifies capacitance, and V indicates voltage. Despite potentially completing workloads faster, higher frequencies result in increased energy consumption due to elevated voltages. Fig. 2 (a) illustrates a motivational experimental analysis of the required energy for rendering a frame under various GPU frequencies. It can be observed that, in general, higher operation frequencies consume more energy. As a result, selecting a proper lower frequency decreases overall energy consumption.

The default DVFS governor in the Google Pixel 8 Pro bases its frequency adjustments on GPU utilization metrics within defined time intervals. Specifically, the system monitors GPU utilization every 20 milliseconds. When utilization exceeds the upper threshold, the frequency increases to the next higher level. Conversely, if utilization falls below the lower threshold, the frequency decreases to the next lower level. Fig. 2 (b) details the operation of this strategy. For instance, at a current operating frequency of 337 MHz, the system elevates the frequency to 376 MHz if utilization surpasses 94%. Alternatively, the frequency reduces to 302 MHz if utilization drops below 60%. This dynamic adjustment mechanism enables the system to balance performance according to the GPU workload.

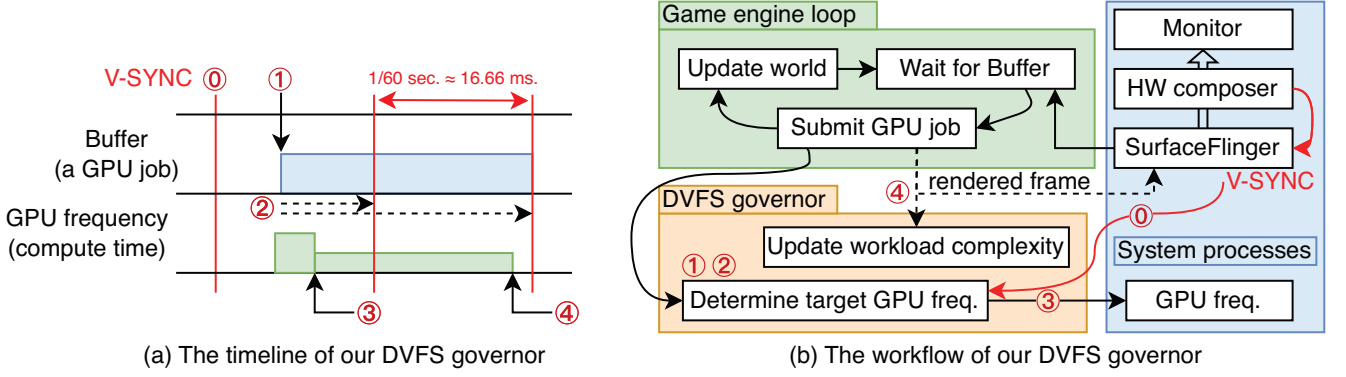


Fig. 4. An overview of the proposed methodology, which covers: (a) the Timeline, and (b) the Systematic Workflow among the game engine, system processes, as well as the proposed DVFS governor. The process encompasses the following four steps: (0) Initialization, (1) Monitoring, (2) Frequency Calculation, (3) Frequency Adjustment, and (4) Workload Completion and Update.

B. Android Game Application Workflow

A game represents a virtual environment simulated by a program and updated periodically through a game loop. Each iteration of the loop, termed as a ‘tick’, advances the virtual time and updates the status of game objects (e.g., velocity, position, lighting). In modern game applications, the CPU calculates and generates the game world, while the GPU renders the scene to a single frame. Once rendered, the frame is processed by SurfaceFlinger, which serves as a system process, and is then updated to the hardware composer and the screen.

Fig. 3 illustrates the overall workflow of a game application. The game loop initiates by updating scenes and awaits the V-SYNC event generated by the hardware composer. This event signals SurfaceFlinger to release the GPU buffer queue, at which point the GPU commences rendering upon the CPU finishes updating the game world. SurfaceFlinger then updates the rendered frame to the hardware composer during the subsequent V-SYNC event and releases the GPU buffer queue. However, if frame completion is delayed due to extended CPU and GPU processing time, a frame skip occurs. In this scenario, SurfaceFlinger refrains from updating the screen and awaits the next V-SYNC event. On the other hand, common Android systems employ a double-buffered scheme. This approach enables concurrent operation of the CPU and GPU: the CPU may update the environment for tick $n + 1$, while the GPU renders the frame for tick n . To recapitulate, the rendering process encompasses CPU-driven game world updates, GPU-based frame rendering, and SurfaceFlinger-executed monitor refreshes, triggered by the V-SYNC event. In this study, we concentrate on enhancing the energy efficiency of the GPU through the selection of an appropriate frequency while adhering to the V-SYNC deadline. The proposed methodology aims to optimize energy consumption without compromising the smoothness and responsiveness of the game application.

III. METHODOLOGY

This section presents the proposed methodology, termed the *V-SYNC deadline-aware GPU DVFS governor*. We begin with an overview and a description of our workflow, followed by the strategy for estimating the GPU frequency and workload.

A. Overview of the Proposed Methodology

Our methodology aims to propose a novel approach to adjusting GPU frequency on a per-frame basis. In contrast to previous works that employ coarse-grained frequency control, our approach estimates the target frequency based on historical workload complexity and a time-to-deadline metric. This fine-grained control enables more precise optimization of energy consumption in mobile gaming applications, and thus enhances both performance and efficiency. However, to achieve fine-grained DVFS control, it necessitates the DVFS governor to possess low latency in capturing system traces and hardware counters, a capability not addressed in the previous works.

Our methodology utilizes the Perfetto [16] software development kit to capture system events and GPU hardware counters in real-time, with a latency of one millisecond. Perfetto is an open-source profiling tool developed by Google, which enables the recording of traces and visualization of reports for Android and Linux devices. To facilitate real-time usage, we implement a small buffer to temporarily store system events generated by Perfetto in memory. Data is continuously extracted from Perfetto into our V-SYNC deadline-aware DVFS governor via a callback function, which is invoked upon the collection of new traces. Such a real-time data collection scheme is crucial for adjusting the GPU frequency on a millisecond scale.

B. Workflow of the V-SYNC Deadline-Aware DVFS Governor

The V-SYNC deadline-aware DVFS governor captures and integrates several Android system events and GPU hardware counters for real-time frequency adjustments. For instance, it monitors GPU frequency, the event when the GPU receives a job, and the V-SYNC event. Fig. 4 illustrates the timeline and workflow of our proposed GPU DVFS governor. Fig. 4 (a) depicts the timeline within the working region, while Fig. 4 (b) illustrates the interaction between the DVFS governor, game engine, and system processes. In Fig. 4 (b), all arrows pointing to the DVFS governor represent traces and counters captured by Perfetto. The detailed workflow proceeds as follows:

- 0) **Initialization:** Prior to the commencement of the GPU job, the DVFS governor continuously captures the V-

TABLE I
THE SYMBOL DEFINITIONS FOR WORKLOAD ESTIMATION.

Symbol	Definition
$Freq_i$	The GPU frequency utilized for frame i
$GPUPTime_i$	The GPU rendering time for frame i
$EstComplexity_i$	The estimated GPU workload for frame i
$Complexity_i$	The actual GPU workload for frame i

SYNC signal and records its timestamp. This process enables the prediction of future V-SYNC event timings.

- 1) **Monitoring:** The DVFS governor remains in a state of readiness for the initiation of GPU activity.
- 2) **Frequency Calculation:** Upon reception of the GPU job, the DVFS governor calculates the required frequency based on the time interval to the next V-SYNC event. In cases where the workload cannot meet the current deadline (e.g., requiring a frequency beyond the GPU's capability), our method introduces the concept of a delayed deadline. This approach designates the subsequent V-SYNC event as the new target deadline. Section III-C details the strategy for determining the GPU frequency.
- 3) **Frequency Adjustment:** Subsequent to the determination of the target frequency, the DVFS governor invokes the GPU firmware's API to adjust the GPU frequency. This process inherently incurs a latency of several milliseconds due to hardware constraints and limitations.
- 4) **Completion & Update:** After the completion of rendering, the DVFS governor recalculates and updates the GPU workload complexity, as elaborated in Section III-C.
- 5) **Repeat:** Steps 1 through 4 repeat for each rendering job.

C. GPU Workload Complexity and Frequency Estimation

This section elaborates on the methodology for predicting GPU workload complexity and estimating the required GPU frequency. The GPU workload complexity represents the computational demands of the GPU job, specifically the requisite computing power for task completion. Previous studies [3], [5], [7], [10], [17] have measured workload complexity by considering GPU utilization and frequencies, and estimated future workload using an exponentially moving average model [18] that updates according to historical workloads (i.e., Eq. (2)). Our approach employs a similar strategy but quantifies the workload complexity as the product of GPU frequency and GPU time (i.e., Eq. (1)). The equation is formulated as follows:

$$Complexity_i = Freq_i \cdot GPUPTime_i. \quad (1)$$

Future workload estimation adopts an exponential moving average model [18], with the equation symbols defined in Table I.

$$EstComplexity_{i+1} = (EstComplexity_i + Complexity_i)/2. \quad (2)$$

The method of representing complexity through GPU frequency and GPU time enables dynamic updates of workload complexity and facilitates the estimation of necessary GPU frequency to meet timing constraints with minimal delay.

While some research [4], [6] has applied machine learning techniques to estimate performance impacts more accurately, these approaches incur higher computational overhead. The product of GPU frequency and GPU time, which indicates the number of executed cycles, serves as a reliable representation of GPU workload complexity. The workload complexity of the subsequent frame is estimated by averaging $EstComplexity_i$ and $Complexity_i$, according to Eq. (2). As a result, $EstComplexity_{i+1}$ iteratively updates based on the workloads of historical frames. Utilizing the estimated workload and the timing conditions (i.e., the time to the next V-SYNC event), the required GPU frequency is calculated by dividing the estimated workload by the time remaining until the deadline, which can be expressed as the following equation:

$$Freq_{i+1} = \frac{EstComplexity_{i+1}}{time\ to\ deadline}. \quad (3)$$

If the required frequency exceeds the GPU's capability, our DVFS governor attempts to meet the workload requirement by selecting the preceding V-SYNC event as the new deadline. Specifically, the *time to deadline* in Eq. (3) is incremented by $\frac{1000}{screen\ refresh\ rate}$ (e.g., 16.66 ms for a 60Hz screen). This allows for deadline extension, ensures the selected frequency remains valid, and employs a lower frequency for energy conservation. Our methodology utilizes historical workload data and real-time conditions to predict GPU workload complexity, and therefore can dynamically adjust the GPU frequency in minimal time.

IV. EVALUATION

This section first introduces the hardware and experimental setups, followed by an evaluation of our method's effectiveness through an examination of overall performance and energy consumption. Our analysis includes a detailed breakdown of GPU frequencies and workload estimation to assess the impact of different configurations on performance and energy efficiency.

A. Experimental setup

Our experiments utilized the Google Pixel 8 Pro mobile phone, equipped with the Google Tensor G3 system-on-chip (SoC) and running Android version 14 (build ID: UQ1A.231205.015). The SoC incorporates a 7-core Arm Mali-G715 GPU, which features thirteen available frequencies ranging from 150 to 890 MHz. Due to GPU hardware limitations, a single frequency applies to all cores within the GPU. Our GPU DVFS governor interfaces with Perfetto [16], a profiling tool for Android systems, and manages the GPU frequency through the API provided by the GPU firmware. The API resides in the local directory `/sys/devices/platform/1f000000.mali`. The CPU frequency is managed by the Android system as it is not discussed in this paper. The screen refresh rate is set to 60Hz.

We selected Fortnite [2] as our benchmark game application, which utilizes the Vulkan graphics API. Vulkan is a more recent and advanced graphics API employed by newer games compared to OpenGL, as it offers higher performance and greater control over the GPU for developers [19], [20]. The Fortnite game is built on Unreal Engine 4, which supports a wide variety of scenes. This diversity in scene composition enables a

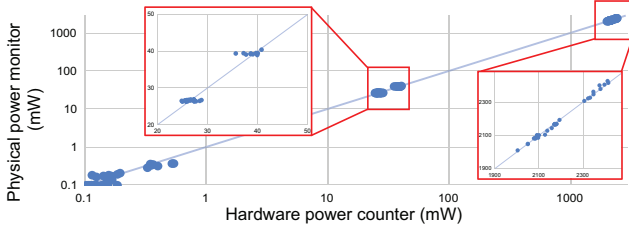


Fig. 5. The strong correlation observed between the hardware power counter and physical power monitor during a ten-minute system measurement period.

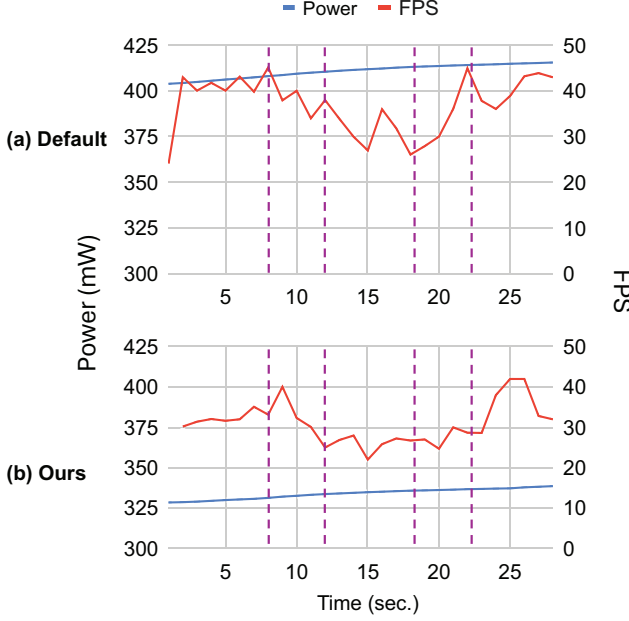


Fig. 6. Power and FPS comparison between (a) the Default and (b) our DVFS governor. The purple dotted lines represent transitions between different scenes. The experimental results indicate that the proposed methodology achieves up to 18.7% energy reduction. Although the average FPS decreases from 37 to 31, our method delivers a 3% improvement in energy-per-frame efficiency.

comprehensive test environment for our experimental purposes. The experimental scene includes character parachute jumping from the sky, movement in outdoor and indoor environments, and interactions with other moving objects (e.g., characters). In addition, particle effects and lighting conditions vary in each scene. We measured the frames-per-second (FPS) and energy consumption of the GPU over a 30-second period to assess the impact of different GPU frequencies on performance and energy efficiency. We collected GPU hardware counter data using Perfetto, while referencing the ARM Mali GPU reference guide [21] to interpret the counters and derive performance and energy consumption metrics. Fig. 5 depicts the strong correlation between the hardware power counter and the physical power monitor, particularly when power exceeds 10 mW.

B. Overall Performance and Energy Consumption

Fig. 6 presents a comparison of power consumption and FPS between the Android default governor and our proposed V-SYNC deadline-aware DVFS governor. The figure illustrates FPS variations over time due to changing game scenes, such

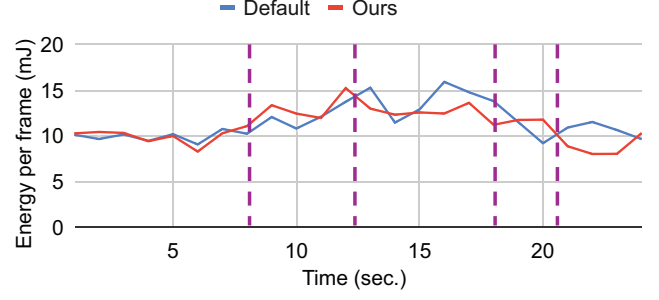


Fig. 7. Energy-per-frame comparison between the default governor and our proposed DVFS governor. On average, our approach achieves a 3% energy improvement compared to the default governor.

as the character's transition from outdoor to indoor environments. Our approach achieves up to a 19% reduction in GPU energy consumption. However, the average FPS decreases from 37 to 31. This reduction in FPS can be attributed to the default governor's operation at higher frequencies, which results in increased energy consumption but enables more consistent task completion before deadlines. Conversely, our DVFS governor, when estimating that a job cannot be finished within the first V-SYNC deadline, opts not to operate at higher frequencies. Instead, it extends the deadline to the next V-SYNC event and selects a lower frequency to reduce energy consumption. Moreover, as game rendering involves concurrent CPU and GPU operations, adjustments to the GPU frequency impact GPU time and utilization, which subsequently affect the DVFS governor for the CPU within the Android system. This interdependency indicates that while our method effectively reduces energy consumption, it faces challenges in accurately estimating workloads, predicting appropriate frequencies, and accounting for the overall impact on the heterogeneous system. Considering that our predictions are based on estimates, this represents the optimum performance achievable under current constraints. With more accurate sensor information, architectural support for acquiring better metrics, and enhanced CPU-GPU coordination, further improvements could be realized.

Fig. 7 further analyzes the energy consumption per frame by examining the energy-to-FPS ratio, calculated by dividing the energy consumed by the FPS. On average, our proposed V-SYNC deadline-aware GPU DVFS governor improves energy efficiency by 3% per frame. This improvement demonstrates that even with a lower FPS, our proposed methodology can achieve superior energy efficiency for each rendered frame.

C. Frequency Breakdown and User Experience Comparison

Fig. 8 presents a histogram of the selected frequencies under the default governor and our DVFS governor. The statistics are based on the average frequency for each rendered frame. The default governor, which adjusts the frequency based on utilization by selecting the next or previous frequency choice as shown in Fig. 2 (b), tends to operate at frequencies that maintain relatively high GPU utilization. As a result, nearly 70% of the frames' average GPU frequency remains within the 430 to 530 MHz range. In contrast, our proposed methodology operates in a more fine-grained manner, selecting appropriate frequencies

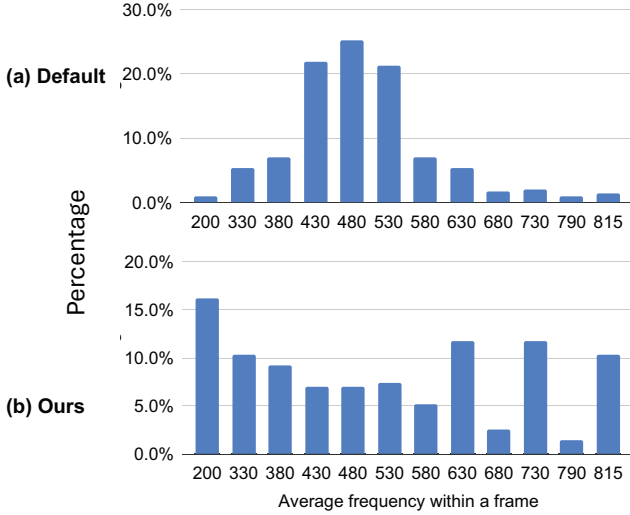


Fig. 8. The frequency breakdown: (a) the default governor, and (b) ours.

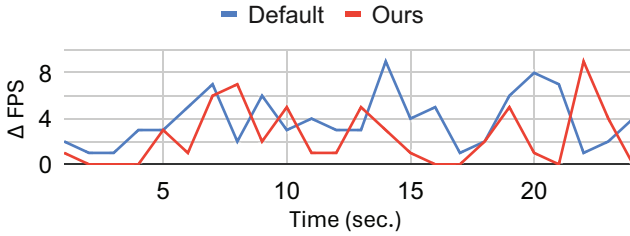


Fig. 9. The FPS fluctuation comparison between (a) the default governor and (b) our proposed V-SYNC deadline-aware GPU DVFS governor. The average delta FPS values are 3.8 for the default and 2.4 for our GPU DVFS governor.

according to the time remaining until the V-SYNC deadline. This approach allows for more precise adjustments tailored to specific workload requirements. The result, as revealed in Fig. 8, is a broader distribution of selected frequencies, which contributes to an enhanced user experience for mobile gamers.

Fig. 9 illustrates the delta FPS (i.e., frame rate fluctuation) for each second during game execution and compares the default Android governor with our proposed DVFS governor. The average delta FPS of the default governor is 3.8, while our method achieves 2.4. This represents a significant 37% improvement in frame rate stability. Such enhancement in stability is crucial for user experience. A lower delta FPS translates to more consistent frame delivery, which reduces perceivable stutters or jitters in gameplay. This consistency provides a smooth, immersive gaming experience on mobile devices. It is worth noting that the authors in [22] have previously established that frame rate variation exerts a more significant influence on players' gaming experience than average frame rate alone. Our methodology takes into account the start time of GPU jobs, which varies in each frame due to the CPU executing both game applications and system processes. This adaptive approach enables better handling of real-world scenarios where system resources are shared among multiple tasks. By more accurately matching GPU frequency to the actual workload demands in a fine-grained manner, our method is able to achieve superior power efficiency while simultaneously enhancing user experience.

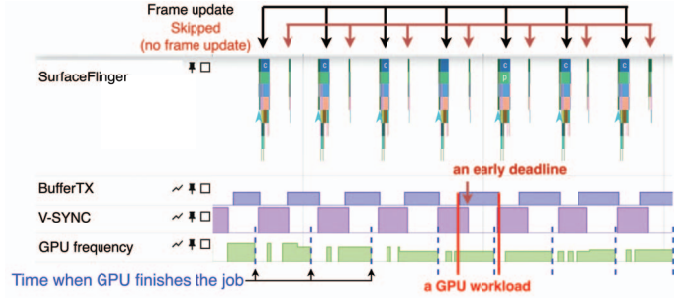


Fig. 10. Profiling of GPU workload & V-SYNC events during game execution.

D. Workload Estimation

For more detailed insights into our DVFS governor's interaction with the Android system, we profiled the game application using Perfetto. The screenshot in Fig. 10 illustrates this profiling. The red and black bold arrows represent the occurrence of the V-SYNC event, which happens at the positive and negative edges of the V-SYNC signal. The V-SYNC event occurs every 16.66 milliseconds, corresponding to a screen refresh rate of 60Hz. The blue counter, BufferTX, represents the buffer size of the GPU job. A counter value greater than zero indicates an incoming GPU job, which reduces to zero when the V-SYNC event occurs and the workload completes. The blue dotted lines at the bottom of the figure indicate the completion of GPU workload, as the GPU frequency drops to zero and the GPU becomes idle. In Fig. 10, we highlight a scenario of an early deadline. Under such conditions, our DVFS governor disregards the early deadline and selects a lower frequency, which ensures workload completion before the next deadline. The blue dotted lines demonstrate our DVFS governor's effective selection of an appropriate frequency, which enables the GPU to complete the workload close to the deadline and minimizes idle time. The maximum time to the V-SYNC event in Fig. 10 is approximately four milliseconds. This observation also reveals that the GPU often receives the job close to the deadline, as indicated by the increase in BufferTX. This further suggests that CPU jobs compress the available time for the GPU, as the CPU must create the scene before passing the job to the GPU.

V. CONCLUSION

This paper introduces a novel V-SYNC deadline-aware GPU DVFS governor designed to optimize energy consumption in mobile gaming applications. Our approach implements fine-grained GPU frequency adjustments aligned with V-SYNC events. This alignment ensures timely frame rendering while minimizing energy wastage. By leveraging the Perfetto profiling tool, we enable real-time system tracing and hardware counter capturing on Android devices. This innovation allows our governor to dynamically estimate workload complexity and adjust frequencies with a remarkably fine-grained 1 ms response time. Experimental results on the Google Pixel 8 Pro demonstrate the efficacy of our methodology, which achieves up to 19% reduction in overall GPU energy consumption and a 3% improvement in energy efficiency per frame. Although we observed a slight decrease in FPS, our proposed methodology

significantly reduced FPS variation by 37%, which substantially enhances frame rate stability and user experience. These experimental results highlight the potential of our V-SYNC-aware DVFS governor to optimize mobile gaming performance while maintaining energy efficiency and user gaming experience.

ACKNOWLEDGEMENT

The authors gratefully acknowledge the support from the National Science and Technology Council (NSTC) in Taiwan under grant numbers MOST 111-2223-E-002-011-MY3, NSTC 113-2221-E-002-212-MY3, and NSTC 113-2640-E-002-003. The authors would also like to express their appreciation for the support from Google Inc.

REFERENCES

- [1] N. Thiagarajan, G. Aggarwal, A. Nicoara, D. Boneh, and J. P. Singh, "Who killed my battery? analyzing mobile browser energy consumption," in *Proc. Int. Conf. on World Wide Web*, 2012, p. 41–50.
- [2] E. Games, "Fortnite," <https://www.epicgames.com/fortnite>, 2024, version 28.10, Platform: Android.
- [3] A. Pathania, Q. Jiao, A. Prakash, and T. Mitra, "Integrated CPU-GPU power management for 3d mobile games," in *Proc. of Design Automation Conf. (DAC)*, 2014, pp. 1–6.
- [4] J.-G. Park, N. D. Dutt, and S.-S. Lim, "ML-Gov: a machine learning enhanced integrated CPU-GPU DVFS governor for mobile gaming," in *Proc. IEEE/ACM Symp. on Embedded Systems for Real-Time Multimedia*, 2017, p. 12–21.
- [5] Y. Choi, S. Park, S. Jeon, R. Ha, and H. Cha, "Optimizing energy consumption of mobile games," *IEEE Trans. on Mobile Computing*, vol. 21, no. 10, pp. 3744–3756, 2022.
- [6] J.-G. Park, N. D. Dutt, and S.-S. Lim, "An interpretable machine learning model enhanced integrated CPU-GPU DVFS governor," *ACM Trans. Embedded Computer System*, vol. 20, no. 6, 2021.
- [7] X. Li and G. Li, "An adaptive CPU-GPU governing framework for mobile games on big.LITTLE architectures," *IEEE Trans. on Computers*, vol. 70, no. 9, pp. 1472–1483, 2021.
- [8] A. Pathania, S. Pagani, M. Shafique, and J. Henkel, "Power management for mobile games on asymmetric multi-cores," in *Proc. IEEE/ACM Int. Symp. on Low Power Electronics and Design (ISLPED)*, 2015, pp. 243–248.
- [9] C.-Y. Hsieh, J.-G. Park, N. Dutt, and S.-S. Lim, "Memory-aware cooperative CPU-GPU DVFS governor for mobile games," in *Proc. IEEE Symp. on Embedded Systems for Real-Time Multimedia*, 2015, pp. 1–8.
- [10] P.-K. Chuang, Y.-S. Chen, and P.-H. Huang, "An adaptive on-line CPU-GPU governor for games on mobile devices," in *Proc. Asia and South Pacific Design Automation Conference (ASPDAC)*, 2017, pp. 653–658.
- [11] Y. Choi, S. Park, and H. Cha, "Graphics-aware power governing for mobile devices," in *Proc. Annual Int. Conf. on Mobile Systems, Applications, and Services*, 2019, p. 469–481.
- [12] B. Sun, X. Li, J. Song, Z. Cheng, Y. Xu, and X. Zhou, "Texture-directed mobile GPU power management for closed-source games," in *Proc. IEEE Int. Conf. on High Performance Computing and Communications, Proc. IEEE Int. Symp. on Cyberspace Safety and Security, Proc. IEEE Int. Conf. on Embedded Software and Syst (HPCC,CSS,ICSS)*, 2014, pp. 348–354.
- [13] Z. Cheng, X. Li, B. Sun, J. Song, C. Wang, and X. Zhou, "Behavior-aware integrated CPU-GPU power management for mobile games," in *IEEE Int. Symp. on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2016, pp. 439–444.
- [14] A. Prakash, H. Amrouch, M. A. Shafique, T. Mitra, and J. Henkel, "Improving mobile gaming performance through cooperative CPU-GPU thermal management," in *Proc. of Design Automation Conf. (DAC)*, 2016, pp. 1–6.
- [15] C. Lin, K. Wang, Z. Li, and Y. Pu, "A workload-aware DVFS robust to concurrent tasks for mobile devices," in *Proc. Int. Conf. on Mobile Computing and Networking*, 2023, pp. 1–16.
- [16] Google, "Perfetto - system profiling, app tracing, and trace analysis," <https://perfetto.dev>, 2024, accessed: 2024-07-08.
- [17] A. Pathania, A. E. Irimiea, A. Prakash, and T. Mitra, "Power-performance modelling of mobile gaming workloads on heterogeneous MPSoCs," in *Proc. of Design Automation Conf. (DAC)*, 2015.
- [18] S. Roberts, "Control chart tests based on geometric moving averages," *Technometrics*, vol. 42, no. 1, pp. 97–101, 2000.
- [19] S. I. Gunadi and P. Yugopuspito, "Real-time GPU-based SPH fluid simulation using Vulkan and OpenGL compute shaders," in *Proc. Int. Conf. on Science and Technology (ICST)*, 2018, pp. 1–6.
- [20] O. Ferraz, P. Menezes, V. Silva, and G. Falcao, "Benchmarking Vulkan vs OpenGL rendering on low-power edge GPUs," in *Proc. Int. Conf. on Graphics and Interaction (ICGI)*, 2021, pp. 1–8.
- [21] ARM, "Arm Mali-G715 performance counters reference guide," <https://documentation-service.arm.com/static/66716955e98d6a2ff18646c8>, 2024, accessed: 2024-07-08.
- [22] S. Liu, A. Kuwahara, J. J. Scovell, and M. Claypool, "The effects of frame rate variation on game player quality of experience," in *Proc. CHI Conf. on Human Factors in Computing Systems*, 2023.