Energy Efficient Smartphones: Minimizing the Energy Consumption of Smartphone GPUs using DVFS Governors

Thesis by

Enas Ahmad

In Partial Fulfillment of the Requirements

For the Degree of

Masters of Science

King Abdullah University of Science and Technology, Thuwal,

Kingdom of Saudi Arabia

(May, 2013)

The thesis of Enas Ahamd is approved by the examination committee

Committee Chairperson: Dr. Basem Shihada

Committee Member: Dr. Markus Hadwiger

Committee Member: Dr. Mohamed-Slim Alouini

ABSTRACT

Energy Efficient Smartphones: Minimizing the Energy Consumption of Smartphone GPUs using DVFS Governors

Enas Ahmad

Modern smartphones are being designed with increasing processing power, memory capacity, network communication, and graphics performance. Although all of these features are enriching and expanding the experience of a smartphone user, they are significantly adding an overhead on the limited energy of the battery. This thesis aims at enhancing the energy efficiency of modern smartphones and increasing their battery life by minimizing the energy consumption of smartphones Graphical Processing Unit (GPU). Smartphone operating systems are becoming fully hardware-accelerated, which implies relying on the GPU power for rendering all application graphics. In addition, the GPUs installed in smartphones are becoming more and more powerful by the day. This raises an energy consumption concern. We present a novel implementation of GPU Scaling Governors, a Dynamic Voltage and Frequency Scaling (DVFS) scheme implemented in the Android kernel to dynamically scale the GPU. The scheme includes four main governors: *Performance*, *Powersave*, *Ondmand*, and *Conservative*. Unlike previous studies which looked into the power efficiency of mobile GPUs only through simulation and power estimations, we have implemented our approach on a real modern smartphone GPU, and acquired actual energy measurements using an external power monitor. Our results show that the energy consumption of smart-

phones can be reduced up to 15% using the *Conservative* governor in 2D rendering mode, and up to 9% in 3D rendering mode, with minimal effect on the performance.

ACKNOWLEDGEMENTS

First and foremost, I would like thank my adviser Dr. Basem Shihada, for his indispensable guidance, and for his continuous help and rich feedback through out my thesis. Also, for his trust, light spirit and sense of humor.

I would also like to thank Dr. Markus Hadwiger, and Dr. Mohamed-Slim Alouini for being part my committee, and for all their time.

A special thanks goes to Ola and Hessa, for all the nights we spent, the tea we drank, and the conversations we shared. Could not have pulled through without your amazing company.

And finally, to my wonderful family. Everything I am, I owe to them.

"*And my success is not but through Allah. Upon him I have relied, and to Him I return.* -Hud(88).

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Introduction

1.1 Overview

In 1992 IBM announced ”Simon”, the first smartphone in the world [1]. What was phenomenal about ”Simon” was that it added capabilities other than making phones calls to mobile devices. Some of these capabilities were: a calendar, storing contacts, creating task lists, and the ability to use third party applications. Although it might sound very trivial compared with today’s phones, at that time it was considered a breakthrough in the mobile industry, and therefore ”Simon” was labeled as ”Smart”. Years later, smartphones have been emerging into our world in a rapid rate. According to Gartner [2], sales of smartphones witnessed an increase of 96% in the third quarter of 2010, which amounted to more than 81 million units sold in less than 3 months. However, smartphones have come a long way since 1992; modern smartphones are being designed with increasing features and capabilities. It is common today to see smartphones having Dual and even Quad core processors, their speed as fast as 1.7 GHz, and with up to 2 GB of RAM. Smartphones are also being equipped with a variation of network interfaces, such as WiFi, 3G, 4G, EDGE and Bluetooth. Moreover, high resolution screens and powerful GPUs are being installed in most smartphone devices.

Although all of these features are enriching and expanding the experience of a smartphone user, but they are significantly adding an overhead on the limited energy of the battery. Few years back, a mobile phone would last three to four days without a recharge, however, today's users are complaining of the fast drainage of their phones batteries and the need to recharge them on a daily basis if not more than once in a single day. According to a study done by [3], 80% of mobile phone users are searching for various measures to increase their battery lifetime. This shows that the short lifetime of the phone battery is becoming a serious concern for users. Thus, designing energy efficient smartphones while keeping up with these newer capabilities is now becoming vital.

1.2 Problem Statement

Due to the growing importance of this issue, researchers have been showing increasing interest in designing energy efficient smartphones. The first step to accomplish that would be to establish an in-depth characterization and analysis of the energy consumption of the different components of the smartphone. This analysis would help in understanding the main sources of energy drainage under different usage scenarios. Several work such as [4–9] have aimed to achieve this. The main components under measurement were: GSM, CPU, RAM, Wifi, 3G, GPS, and screen. The energy consumed by each of these components is calculated separately while keeping the others constant. Furthermore, some researchers have targeted the second step, which is to minimize and optimize the energy consumption of smartphones. Such work will be discussed in detail in the State of the Art section. Although these studies have proposed practical and employable solutions for almost every component in the smartphone including CPU, networks, GPS, and application-level efficiency, they have failed to do so for one of the main components of any modern smartphone, which is the GPU. Recall, GPUs are known for high computational and processing power.

In smartphones, GPUs are located on the same chipset as rest of the processing cores, which is referred to as SoC (System-on-a-Chip) architecture [10]. The addition of the GPUs to the SoC have provided a big leap in the graphical experience. Applications and user interface components in modern smartphones use very rich and high-quality graphics. Therefore, smartphone vendors are increasing the computational capabilities of their GPUs accordingly, to ensure the smoothness of the graphics rendering. The resolution of the smartphone displays has also been increasing steadily over the recent years. The resolution can range from 120 x 160 pixels, and up to 768 x 1280 pixels found in the Black Berry Z10. But higher display resolution means heavier load for the GPU in the rendering process. Moreover, operating systems such as Android are designed with full support of hardware acceleration [11]. The drawing operations and the rendering of both 2D and 3D graphics are all carried out by the GPU. All these significant advancements in the smartphone graphics raise a concern of the energy consumption of the GPU, which has not yet been practically investigated. Some few primary work have aimed to enhance the power efficiency of mobile GPUs, as it will be discussed later on. However, the problem of all these studies is that they are simulation-based and not tested or implemented on real mobile GPUs. Moreover, the power measurements in these studies are only approximations derived from theoretical energy models. We believe that simulating smartphone GPUs is not sufficient to reflect the variations, characteristics, and constraints of a real smartphone environment.

1.3 Selected Approach

Smartphone applications can vary from simple 2D web browsers or text based editors to high resolution 3D gaming. This variation encourages using dynamic scaling techniques for the GPU based on its real time utilization, similar to the ones used for the CPU. Energy can be saved by dynamically adjusting the GPU frequency and voltage to the current usage, instead of running it at its highest performance the entire time. Linux Kernel, which is used by the Android OS, defines what is famously known as ”CPUFreq Scaling Governors” [12]. Each of these governors provides a different scaling algorithm for the CPU that give a trade-off between performance and power. The main governors are: *Performance*, *Powersave*, *Userspace*, *Ondemand*, and *Conservative*. These CPU governors were the inspiration behind this study. We have adapted the same model of the CPUFreq governors to dynamically scale the smartphone GPU. The motivation is to enhance the power efficiency of the GPU, and provide it similar variation and flexibility that is offered to the CPU by those governors

Therefore, this work specifically addresses the energy consumption of the smartphone GPU, and proposes the novel implementation of the GPU Scaling Governors similar to the Linux-based CPUFreq Governors, in order to minimize the total energy consumption of the GPU in smartphones. Unlike previous work in this area, we have implemented and tested our approach on a real modern smartphone GPU, and acquired actual energy measurements using an external power monitor.

1.4 Road Map

The thesis is organized as follows. First, the background and state of the art is presented in Chapter 2. Chapter 3 describes the methodology used in the study. Following that, Chapter 4 presents the result and the discussion of the new approach. And finally, the conclusion and future work are discussed in Chapter 5.

Chapter 2

Background and State of the Art

There has been a significant improvement in the lithium-ion batteries that are used in most smartphones nowadays [13]. Never the less, this improvement is obviously unable to keep up with the rapid evolution of the smartphone capabilities. Therefore, researchers are trying to enhance the usage of the components in a smartphone in order to make them more energy efficient. The following sections discuss some state of the art energy efficient methodologies in the smartphone applications, networks, memory, CPU, and GPU.

2.1 Energy Efficiency in Smartphone Applications

Smartphone operating systems give user-space applications wide freedom to use the different capabilities of the phone, such as GPS, Wifi, 3G, sensors, and camera. However, most application developers neglect the energy consequences of their applications, aiming only to provide more impressive features. The following sections present some state of the art techniques used to optimize the energy consumption of Always-on applications, Location-based applications, and finally methods for detecting energy-hungry bugs and malware in smartphones.

2.1.1 Always-On Applications

Many smartphone applications have online features and perform periodic updates of their status over remote network servers. A very common example of such application is the email. In most cases, a timer is associated with these kind of applications, and once it goes off a network connection is established. If several of "always-on" applications are running on the same device, and each with a timer going off at a different time, it will cause a high energy consumption and prevent the smartphone of going into a deep-sleep state. A novel solution for this problem is proposed by the authors of [14]. Their idea is to align those different timers in such a way they go off at almost the same time. This timer alignment will cause a parallel network access, which is proven to consume less energy than multiple separate accesses. They have implemented two different alignment algorithms: Array Based Alignment, and Statistical Clustering Based Alignment. Their experimental results show that energy consumption can be reduced by up to 60% with their experimented application set using timer alignment algorithms.

2.1.2 location-Based Applications

The ability to provide realtime positioning information might be one of the most significant features added to modern smartphones. This information is provided through multiple interfaces, such as: GPS, GSM and Wifi. Currently, a variety of applications and services are being based on this feature. But continuously fetching the current location might be one of the main reason behind absorbing the power of the phone battery. Therefore, many studies have been conducted in order to make the usage of location-based applications more energy efficient.

GPS is considered the most accurate positioning system compared with its alternatives, nerveless, it is the most power-hungry. Hence, a study in [15] proposed an enhancement to GPS-based positioning in smartphones, which is to make it rate-adaptive. The main idea of their design is to turn on GPS only as often as necessary without effecting the achieved accuracy. The decision to when to turn on GPS is cleverly made based on several techniques such as, Location-time history logs of the user, the estimation of user's movement using an accelerometer, communicating with neighboring devices via Bluetooth to reduce position uncertainty, and turning off GPS when it is unavailable (such as indoors) by using celltower-RSS blacklisting.

A similar approach is used by the authors in [16]. Their framework suggests to always use the most energy efficient positioning method (which is cellular network positioning) as a default approach, and only switch to the higher energy consuming methods such as Wifi or GPS when the accuracy of the previous reading is insufficient.

2.1.3 Detecting Energy-Greedy Bugs and Malware

With the wide spread of battery-powered devices, a new class of bugs have emerged called "Energy Bugs". Authors of [17] define the energy bugs or ebugs as any system error that causes a high or unexpected energy consumption of the system as a whole. They characterized the different type of energy bugs by mining 39k posts collected from 4 online forums and mobile bug repositories. After filtering the unrelated posts, they cluster the remaining into groups using k-mean. The clustering resulted in defining 4 main groups of ebugs. The first group is the hardware ebugs, such as faulty batteries, damage in the exterior hardware, or scratches and cuts on the SIM or SDcard. The second type is the software ebugs, classified in either OS or application-level related. The third group is bugs that result from external conditions, such as a weak wireless signal, or a very frequent wireless handover. And finally, the last group is the unknown bugs which came form posts that reported energy drainage problems, but the reasons behind them were still undiscovered.

Moreover, the work in [18] have focused specifically on one type of application-level bugs, and that is No-Sleep energy bugs. Most of smartphone platforms or operating systems employ a strict sleeping policy. The sleeping policy sets the phone components in a low-power state while being idle. But on the other hand, applications are given some control over keeping these components awake to serve their needs. A No-Sleep bug would acquire what is referred to as a "wakelock" over one or more components, and then would not release this "wakelock" properly to let the component go back to a sleeping state. The authors propose a compile-time tool for automatically detecting such bugs. Their tool is based on the classic reaching definitions dataflow analysis problem, which detects the code path of a No-Sleep bug statically at compile time. The evaluation results show that their tool successfully detects all reported No-Sleep bugs, and around 30 new instances of unreported bugs.

In addition to energy bugs, some malicious softwares are also being written to target the battery of smartphones. Such softwares would run dummy intensive CPU operations or network transmissions in the background in order to drain the battery. A study in [19] proposed a detection framework for such energy-greedy anomalies using power signature matching. Their framework consists of two stages: a Power Monitor and a Data Analyzer. The power monitor's main job is to detect any abnormal power consumption, and then send these power samples to the data analyzer. Following that, the data analyzer, which can be placed on the same handheld or on a remote server, creates a power signature out of these samples and tries to match it with known malwares stored in the database. However, their work is unfortunately limited by the small number of publicly available worm samples for research purposes.

2.2 Networks

Data usage via mobile devices have dramatically increased in the last few years. Modern smartphones are being equipped with multiple radio interfaces for sending and receiving data. Moreover, the lower prices of such transmissions have encouraged the replacement of traditional voice calling and SMS with Voice over IP (VoIP) and wireless messengers. However, these continuous connections are made at the price of the limited battery power. Multiple studies have worked on reducing the power consumption of network communication in smartphones. The work in [20] proposed a framework to reduce the idle-listening (IL) power in Wifi communications, which according to them accounts to 60% of the total consumed energy of Wifi. Their framework *E-MiLi* minimizes the Wifi power by down-clocking the radio during IL, and restore it to the full clock rate only before transmission or after detecting a incoming packet. *E-MiLi* uses sampling rate invariant detection algorithm to ensure the ability of detecting packets accurately with a down-clocked radio. Moreover, it uses opportunistic down-clocking mechanism to balance the overhead of switching back and forth between clock rates, and to maintain the compatibility between the previous MAC and sleep protocols. Their evaluation shows that *E-MiLi* reduces the energy consumption up to 44% of 92% users of real wirless networks, with close to 100% accuracy in packet detection.

Authors in [21] focused on minimizing the high tail energy when connecting via GSM or 3G. They targeted two types of network applications: Delay-Tolerant, and Delay-Intolerant. To minimize the tail energy of delay-tolerant applications such as email or RSS feeds, a scheduling algorithm for transmissions is employed, which saves the total energy consumption while maintaining the applications' strict deadlines. As for the delay-intolerant applications such as web search, an aggressive pre-fetching of data is used after analyzing the statistics of the user behavior.

Finally, authors of [22] presented a protocol to reduce the energy consumption of VoIP using wireless wake-ups. The concept behind their prototype protocol *Cell2Notify* is to only turn on Wifi when the user is making a VoIP call, or if there is an incoming VoIP call for that user to receive. A wake-up signal is sent to the user from the *Cell2Notify* server using the cellular interface to notify them of an incoming VoIP call. The *Cell2Notify* client installed on the mobile phone would then turn on Wifi to receive the call. *Cell2Notify* saves energy by using the cellular network that is rarely turned off by the user, and consumes significantly less energy than Wifi when it is not used.

2.3 Memory

Although energy efficiency in smartphone memories is one of the least tackled fields in the literature, there are some interesting work that have been done in this area by the authors of [23]. They studied the applicability of memory energy optimization methods used in computer systems, such as Power Aware Virtual Memory and Ondemand Mechanisms, on a simulated smartphone memory. They started by collecting memory traces of some of the most famous smartphone applications and simulated two types of memories: Mobile Memory and Phase Change Memory. Following that, they applied the previously mentioned two mechanisms on their simulations. After studying the efficiency and performance of each of them, they proposed a hybrid approach that takes advantage of both types of memories, and according to their results can save up to 98% of memory energy when compared with the traditional smartphone memory technologies.

2.4 CPU

Smartphone CPUs are catching up with its fellow desktop and laptop processors, in terms of speed, power, and number of processing cores. But the limited energy of the phone battery creates a heavy burden upon this fast evolution. A research group at the university of California (UC) San Diego is working on a new smartphone processor prototype they call GreenDroid [24]. The novel idea of GreenDroid, is to employ the unused dark silicon found on chips and turn it into Conservation cores (c-cores). C-cores are application-specific, energy efficient processing cores. GreenDroid would combine the use of general-purpose processors with application-specific and very efficient co-processors for energy saving. They proved that their design would reduce the energy consumption up to 91% of the code that is targeted by the c-cores.

In addition, many researchers have been working on finding the most optimal dynamic voltage and frequency scaling (DVFS) framework for the CPU in terms of energy saving. The researchers in [25] noticed that running CPUs at the lowest frequency possible does not always save energy. This is because decreasing the CPU frequency increases the total execution time, which also effects the active time of the memory. Therefore, they based their DVFS algorithm on predicting the critical speed of the CPU, at which it will save the most energy. Predicting this speed is done at the execution time using a correlation equation between the memory access rate and the CPU frequency.

On the other hand, the work in [26] suggests a new CPUFreq governor, which takes into account the foreground applications in addition to the CPU instantaneous utilization. They found that by considering the running foreground applications some unnecessary jumps for high frequencies can be avoided. After implementing their proposed governor *appspace*, they test it against the Ondemand Linux based governor. Their results show that *appspace* saves more energy than the Ondemand, while maintaining the smoothness of the user experience.

2.5 GPU

The introduction of GPUs to mobile and handheld devices have enabled rendering and displaying high quality graphics. But the design of such GPUs should consider the energy consumption as a priority factor even before the performance [27]. Some recent studies have targeted the energy efficiency of GPUs in mobile devices. The work in [28] analysis the trade off between energy and arithmetic precision of mobile graphics processor, by focusing on the vertex transformation stage in the graphics pipeline. The goal is to measure the energy savings that can be achieved by lowering the accuracy of the arithmetic operation to an acceptable level. Their energy model is based on approximating the energy usage of the number of signal transitions per operation. Using the ATTILA GPU simulation framework [29], their results show that around 23% of the energy can be saved by lowering the precision of the arithmetic operations, while maintaining the quality of the rendered image.

Moreover, since dynamic voltage and frequency scaling (DVFS) is one of the traditional power optimization techniques for general processors, researchers investigated the applicability of this method on GPUs as well. A primary study in [30] shows that 3D interactive gaming is amendable to DVFS. They use the open source Quake II game engine [31] for calculating their measurements, which proved that 3D games have variations in their frames processing workload. Such variation enables the use of DVFS, which according to them could save significant amount of energy.

Authors of [32] built their approach upon this conclusion. They present a quantitative study of the power consumption of mobile 3D games using three embedded processors to simulate the different stages of a mobile graphics pipeline. For measuring the power consumption, they employ an instruction-level energy model, which is based on actual measurements of the current drawn by a commercial processor. The trace-driven simulation approach they used is implemented by modifying the OpenGL\ES library to generate trace triggers. Their measurements confirms the existence of an imbalance of workload between different graphics stages and applications. They test 6 DVFS schemes each with a diffident workload prediction algorithm. Their results show that applying DVFS to graphics pipeline could to saves up to 50% of energy. However, their energy model is derived from a general processor rather than a graphics processor, which differ in their architecture and complexity, thus their power consumption.

Similar work in [33] focuses on enhancing the workload prediction for tile-base architecture GPUs. Tile rendering architecture splits the task of rendering a frame into a grid of tiles which the frame is composed of. They propose two DVFS schemes, the first based on *tile-history* prediction, and the second based on *tile-rank* prediction. To evaluate the efficiency of the two approaches, they modify the ATTILA GPU simulation framework emulating a tile-based mobile graphics architecture. Their results show that both schemes save around 58% of energy consumption, with higher quality achieved by the *tile-rank* prediction. But unlike the previously mentioned studies, the method for driving their power model measurements was not clearly stated.

In conclusion, all the previous work in the area of energy efficiency of GPUs of handhelds and mobile devices have been limited to simulation-base and not tested on real mobile GPUs. Furthermore, the power measurements of these studies are only estimations derived from theoretical energy models. Therefor, and to the best of our knowledge, the work we are presenting is the first to propose the DVFS GPU governors implemented and tested on a real modern smartphone, with the energy consumption accurately measured using an external power monitor device.

Chapter 3

Methodology

3.1 Design and Implementation

Dynamic voltage and frequency scaling (DVFS) is one of the commonly-used techniques for power saving in electronic devices. The power consumed by any CMOS (Complementary Metal-Oxide-Semiconductor) component is proportional to the voltage and frequency [34], as shown by

$$Power \propto Voltage^2 \times Frequency \tag{3.1}$$

Therefore, when the frequency is lowered, the consumed power is proportionally lowered. While lowering the voltage, will make the power drop quadratically [35]. DVFS takes the advantage of a very common feature of computer applications, which is that their average computational throughput is often much less than the computational peek capacity of the processor [36]. Therefore, a significant amount of power can be saved by scaling the frequency and voltage of the processor according to the real-time load of the running applications. This power reduction will in its turn reduce the heat emission in large scale supercomputers and save battery in low-scale embedded processors. As mentioned previously, in addition to CPUs many studies have shown the applicability of this method in GPUs as well. These findings encourages us to study the effectiveness of different DVFS schemes on real smartphone GPUs.

The behavior of any hardware component is defined by its device driver. Therefor, in order to implement the GPU DVFS governors, we had to rewrite parts of the GPU driver at the kernel level of the Android operating system. The following subsection will give more details about the architecture of the Android OS. Then we will talk more specifically about the Mali-400 device driver, the GPU chosen for this study. And finally, before presenting the implementation of our approach, we will introduce the Linux CPUFreq governors that form the base of our design.

3.1.1 Android Architecture

Android is a Linux-based operating system created by Google and Open Handset Alliance for touchscreen mobile devices, such as PDAs and smartphones. Android is an open source project, the code of the android platform and kernel is resealed publicly under the Apache license. This gives chance for manufactures, developers, and researchers to freely modify any part of Android and redistribute it. Figure 3.1 displays the Android architecture [37].

At the topmost of the Android software stack is the applications layer. Andorid applications are written in JAVA, and could have limitless access to the phone features and services as long as the user allows them. Following that is the Application Framework layer, which offers APIs to enable the reuse of components. Then we have a set of C/C++ libraries used by developers through the application framework APIs, and the Android Runtime environment consisting of core libraries and the Dalvik virtual machine.

The final and lowest layer of the Android architecture is the kernel. The Android kernel relies on the Linux version 2.6. The main core system service of Android such as process management, network stack, and security are all Linux-based. In addition, the Linux kernel provides Android with its driver model. Linux acts as an abstraction

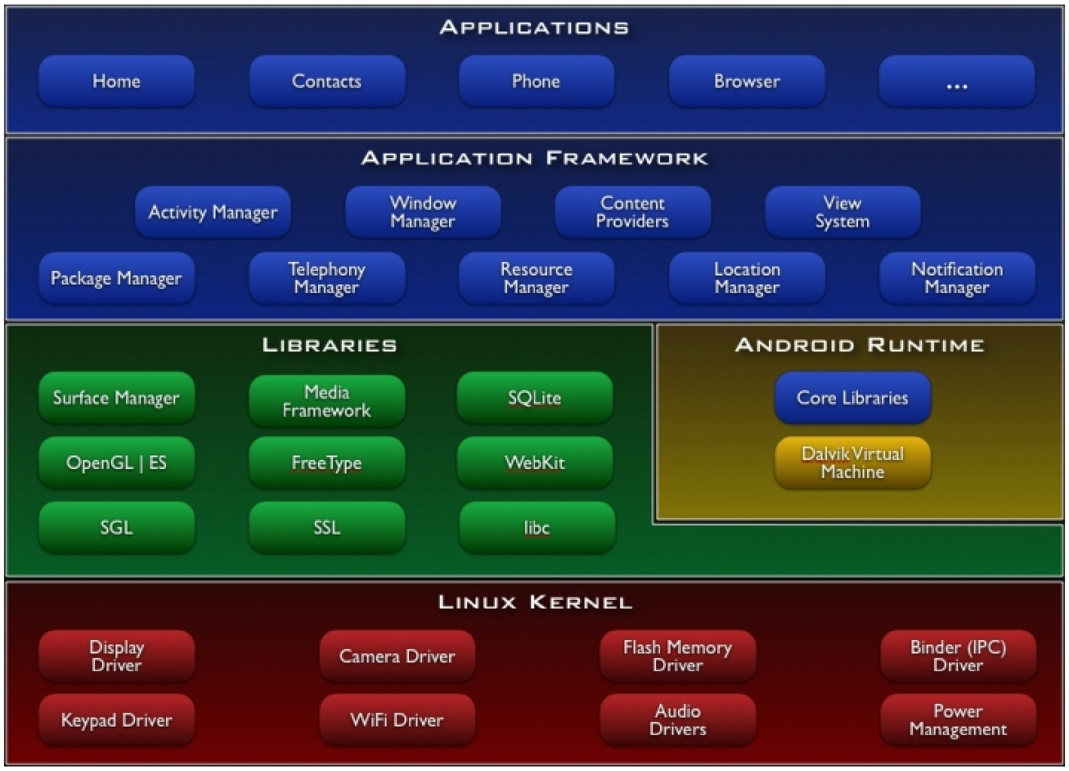


Figure 3.1: Android architecture [37]

layer between the software and the hardware components. The Android Open Source Project [38] publishes the source code of multiple kernel versions, each corresponds to a specific chipset. After downloading the correct source code, the ”gcc” prebuilt tool-chain is used to build the kernel, and the output ”zImage” can be then ”flashed” into the target device.

3.1.2 Mali-400 GPU Driver

The GPU selected for this study is the ARM Mali-400 MP [39]. Mali-400 is one of the most powerful GPUs installed in smartphones. It support both 2D and 3D graphics, and its throughput can achieve 30M triangles/s and up to 1.1G pixels/s at 275MHz. Figure 3.2 shows the Mali GPU device driver stack.

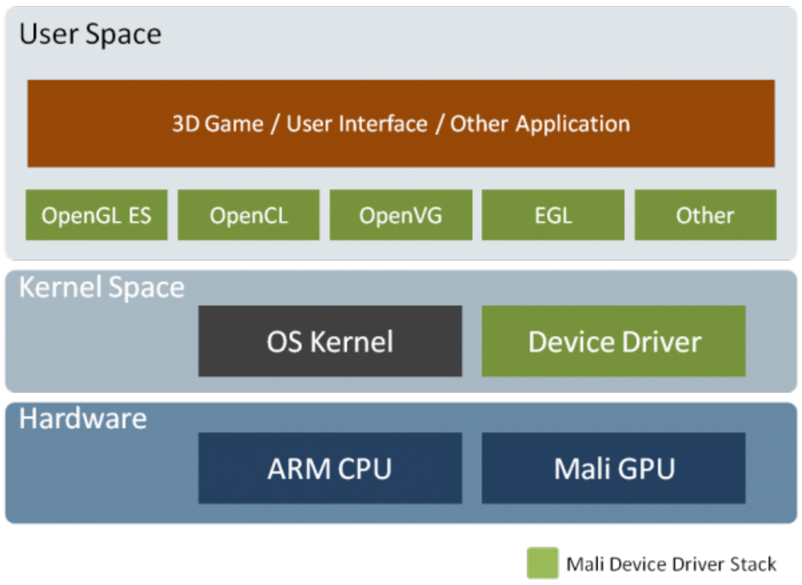


Figure 3.2: Mali device driver stack [40]

The Mali-400 is installed in the Samsung Galaxy S2 GT-I9100 which was used in our experiments. The full specification of Samsung Galaxy S2 is described in table 3.1. Samsung releases its specific Android kernel source code that gave us access to the implementation of the Mali-400 GPU device driver of the kernel space. We based our implementation on the eighth update of the Samsung Galaxy S2 GT-I9100 ICS (Ice-Cream Sandwich) and kernel, which is the latest release for that particular device published in the Samsung Open Source Release Center [41].

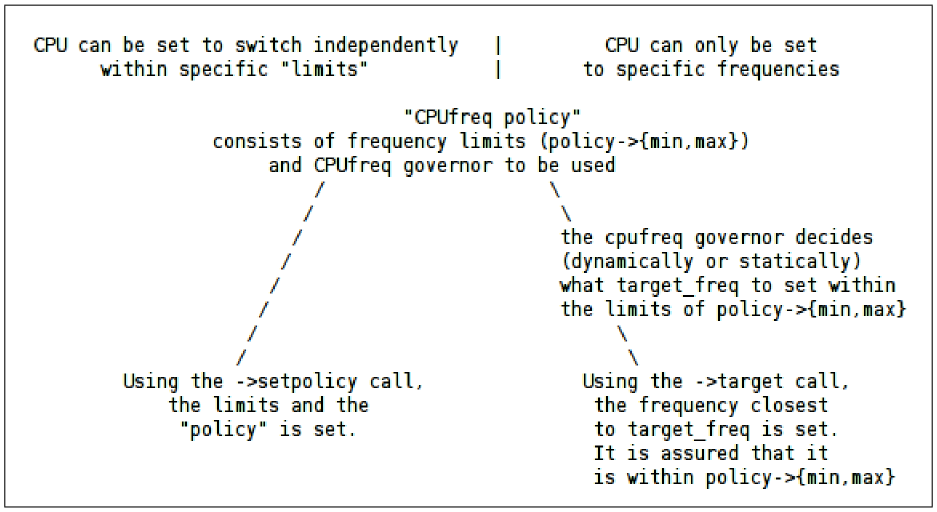
The Mali-400 GPU accepts scaling of its frequency and voltage in realtime. The original "stock" kernel driver offers a basic 2-step DVFS algorithm, which will be described in detail later on. Our goal of implementing the GPU scaling governors is to find more optimal DVFS algorithms in terms of energy efficiency, with minimal effect on the performance. Moreover, having multiple governors will give wider options for users to choose from based on their usage behavior and personal preference. Users can vary from a simple user who is mainly concerned about the phone battery life time, to a user who demands playing 3D-games and care about the high responsiveness of the system.

Table 3.1: Smasung Galaxy S2 GT-I9100 Specifications

Chipset	Exynos 4210
CPU	Dual-core 1.2 GHz Cortex-A9
GPU	Mali-400MP
Network	HSPA+ 21Mbps/ HSUPA 5.76Mbps EDGE/ GPRS Class 12 Quad band GSM 850/900/1800/1900 Quad band UMTS 850/900/1900/2100
Display	4.3" WVGA SUPER AMOLED Plus
Memory	16/32 GB storage, 1 GB RAM
Battery	1650mAh

3.1.3 Linux CPUFreq Governors

Most CPU frequency drivers offer the CPU to be set to only one frequency. Therefore, Linux added the additional layer of CPUFreq Governors to tell these drivers what frequency to apply at runtime [12]. Therefor, the CPUFreq governors allow the implementation of dynamic frequency scaling of the CPU. Since Android is based on Linux, it inherited these governors as well. Figure 3.3 explains the exact flow of the CPUFreq governors control.



```
CPU can be set to switch independently   |         CPU can only be set
      within specific "limits"           |       to specific frequencies

                          "CPUfreq policy"
              consists of frequency limits (policy->{min,max})
                   and CPUfreq governor to be used
                         /                    \
                        /                      \
                       /                   the cpufreq governor decides
                      /                    (dynamically or statically)
                     /                     what target_freq to set within
                    /                      the limits of policy->{min,max}
                   /                            \
                  /                              \
        Using the ->setpolicy call,          Using the ->target call,
            the limits and the                the frequency closest
             "policy" is set.                 to target_freq is set.
                                              It is assured that it
                                              is within policy->{min,max}
```

Figure 3.3: CPUFreq governors control flow [12]

Currently, there are five main governors in the Linux kernel:

- Performance: The Performance governor sets the frequency statically to the highest within the min and max limits of the frequency policy. The goal of this governor is to provide the maximum performance, regardless of the battery consumption. The Performance governor is often used for benchmarking purposes.

- *Powersave*: The *Powersave* governor is the opposite of the Performance governor, since it sets the frequency statically to the lowest within the min and max limits of the frequency policy. The aim of this governor is to achieve the lowest energy consumption possible.

- *Usersapce*: The *Userspace* governor allows a root user or any user-space application with root privileges to set the value of the CPU frequency. This can be achievable by making the *sysfs* file ”scaling_setspeed” available and accessible in the user-space.

- Ondemand: The Ondemand governor sets the CPU frequency dynamically based on the current utilization of the CPU. The utilization value is read periodically according to a ”sampling-rate”. An ”up-threshold” value is set to decide the average CPU utilization between two sample readings, at which the kernel needs to change the CPU frequency. What is unique about the Ondemand governor, is that it switches to the highest frequency once any load on the CPU is detected. This ensures the high responsiveness of the system. But after that, it gradually reduces the frequency if it detects a decrease in the CPU load. This governor is the default in most Android stock kernels, since it provides a good balance between performance and power. But the fact that the Ondemand jumps to the highest frequency each time an increase is needed, might consume extra amount of power that can be avoided .

- *Conservative*: The *Conservative* governor like the *Ondemand*, sets the frequency dynamically based on the current utilization. But it differs form the *Ondemand*, that it does not jump to the max frequency on every increase. It rather gracefully increases the frequency according to a "freq_step" value. The "freq_step" defines the percentage step at which the CPU frequency will be smoothly increased or decreased by. For example, if it is set to 5%, the frequency will be increased at 5% chunks of the maximum frequency each time. Setting the "freq_step" to 100 will theoretically make it behave like the *Ondemand*. The *Conservative* governor is more battery-friendly since it avoids jumping to the maximum frequency unless needed.

3.1.4 GPU DVFS Governors

In this work we have adapted the same model of the CPUFreq governors to dynamically scale the smartphone GPU. We implemented the two main scaling governors: *Ondemand* and *Conservative*. In addition to the two constant governors, *Performance* and *Powersave*. First we will describe the original stock algorithm of the Mali-400 driver, and then our implementation of the four GPU governors.

Stock Algorithm

The original stock algorithm of the Mali-400 orion-m400 platform driver scales between two frequency values based on the realtime utilization of the GPU. The two values are 160 MHz and the maximum frequency for that platform, which is 267 MHz, as shown in table 3.2. The scaling is done step-wise according to an up and down threshold limits for each step. It can be therefore concluded that the lowest frequency the stock algorithm uses is 160 MHz, even if the running application does not require this high level of frequency. In addition, the scaling is limited to only two performance levels, where one of them being the highest.

Table 3.2: Stock Algorithm Frequencies and Voltages

Frequency (MHz)	**Voltage (μV)**
160	950000
267	1000000

Table 3.3: GPU Governors Frequencies and Voltages

Frequency (MHz)	**Voltage (μV)**
50	825000
62	825000
73	825000
80	825000
89	875000
100	875000
133	900000
160	950000
200	950000
267	970000

We have implemented an Android logging application, which loges some of the realtime values of the system, including the frequency and voltage of the GPU. Figures 3.4 and 3.5 show the frequency scaling of the stock algorithm when running the 2D and 3D tests of the *AnTuTu* benchmark application. The figures show how the stock algorithm only switches between 160 MHz and 267 MHz.

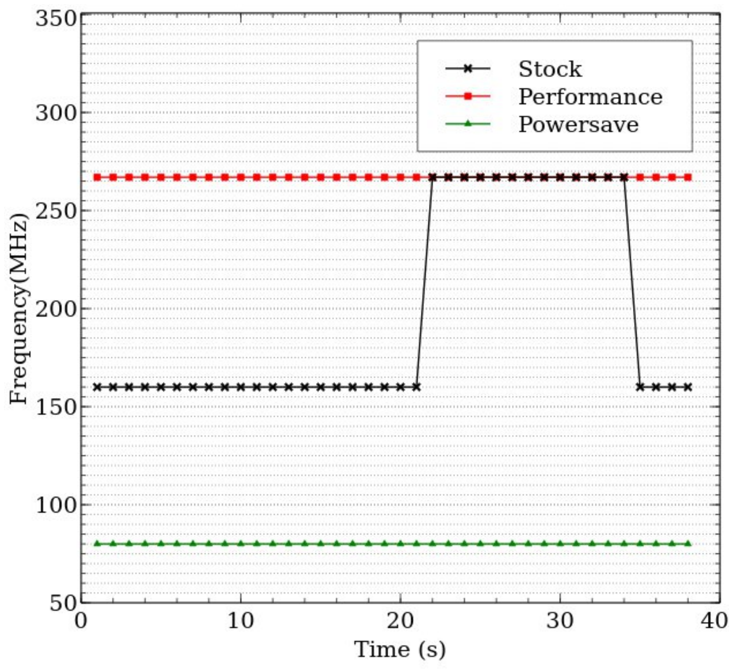


Figure 3.4: Stock algorithm scaling in *AnTuTu* 2D test

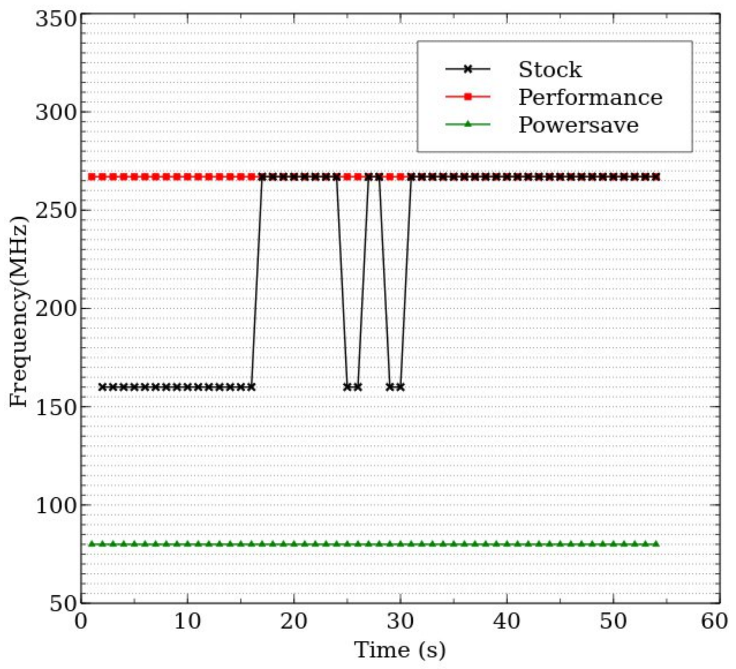


Figure 3.5: Stock algorithm scaling in *AnTuTu* 3D test

***Ondemand* Governor**

The *Ondemand* governor provides high responsiveness since it scales up to the highest frequency whenever the load is increased. However, it also scales down gradually until reaching the minimum frequency that is considered sufficient by the current load.

We rewrote the DVFS model of the kernel Mali-400 driver to make it resemble the structure of the CPUFreq driver. Four main ”tweakable” parameters were added: *sampling_rate, up_threshold*, *sampling_down_factor,* and *freq_step*. Instead of the two frequencies used in the stock algorithm, we extended them to ten frequency levels to have more granular scaling, as shown in table 3.3. Figures 3.6 and 3.7 show the scaling of the Ondeamnd governor when running the 2D and 3D tests of the *AnTuTu* benchmark application. It can be clearly seen how the *Ondemand* jumps to the highest frequency when needed and then gradually decreases the frequency to a suitable level.

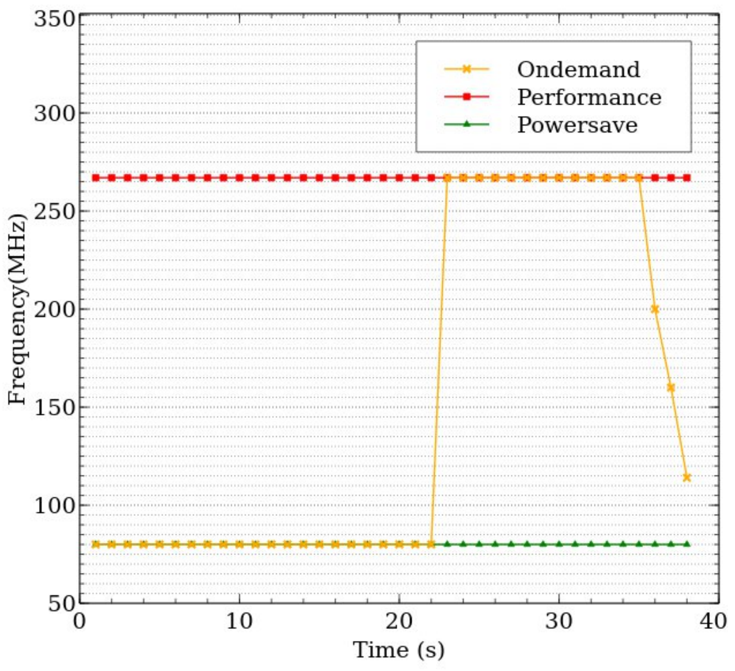


Figure 3.6: *Ondemand* governor scaling in *AnTuTu* 2D test

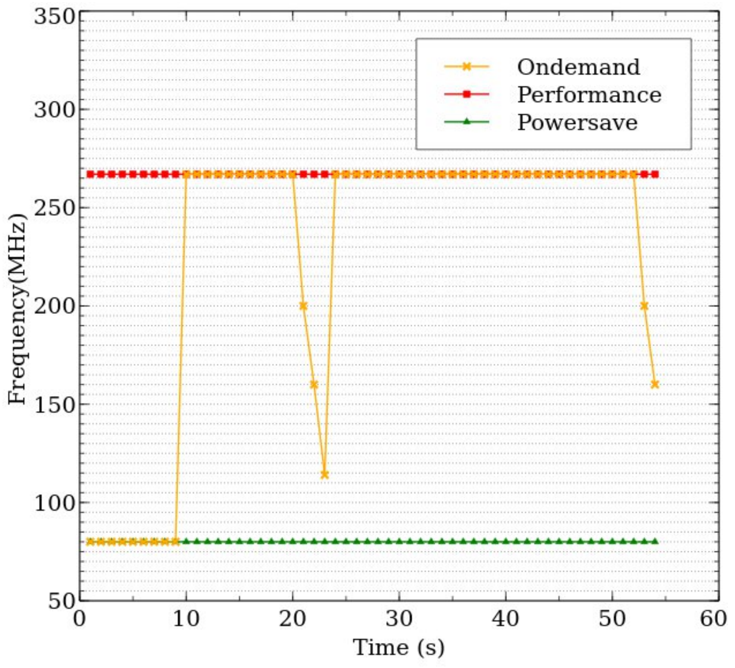


Figure 3.7: *Ondemand* governor scaling in *AnTuTu* 3D test

Conservative **Governor**

The *Conservative* governor is recommended for battery-based devices, since it scales gradually and smoothly between frequency levels. As mentioned before, the *Conservative* increases and decreases the frequency based on the *freq_step* value. We have kept the default value of the *freq_step* which is 5%. This implies that when the GPU utilization value increases over the *up_threshold* percentage, the frequency increases an additional 5% chunk of the maximum frequency. The GPU driver is responsible to find the valid frequency closest to the requested one. This ensures a gradual, but much more energy efficient scaling. The *Conservative* governor uses the same list of valid frequencies and corresponding voltages as the *Ondemand* mentioned in table 3.3. Figures 3.8 and 3.9 show the scaling of the *Conservative* governor when running the 2D and 3D tests of the *AnTuTu* benchmark application. The figures show how the *Conservative* governor scales up and down between the frequency levels very smoothly.

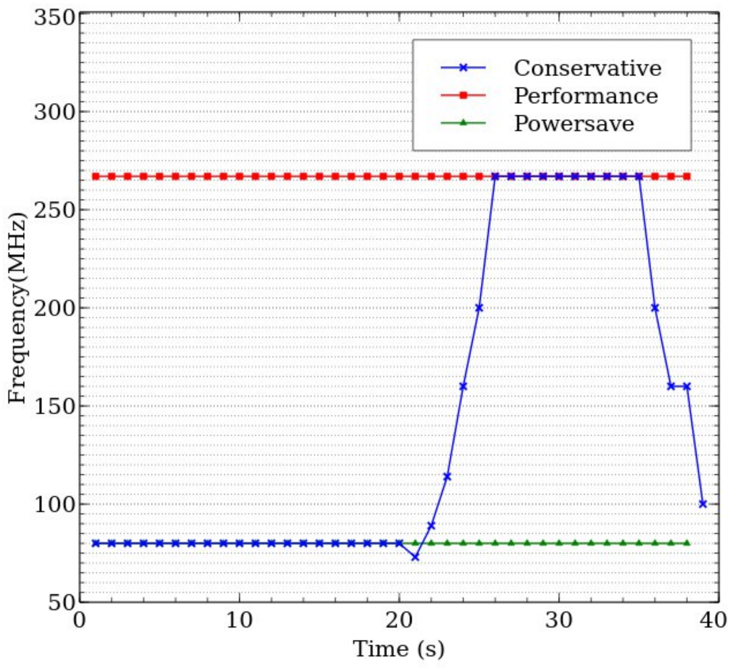


Figure 3.8: *Conservative* governor scaling in *AnTuTu* 2D test

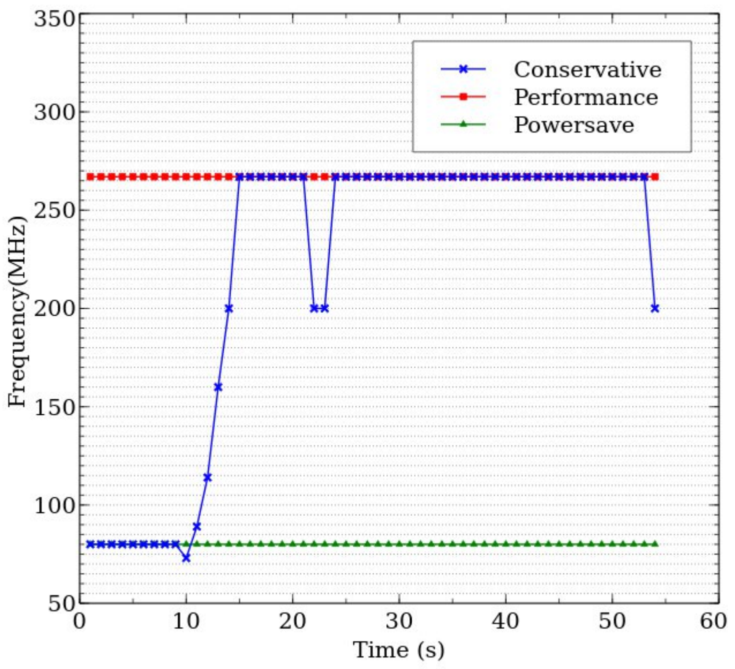


Figure 3.9: *Conservative* governor scaling in *AnTuTu* 3D test

Performance **Governor**

To implement the *Performance* governor, we disabled the DVFS of the GPU and set the frequency statically to 267 MHz which is the highest. This governor is intended for users who require the maximum graphical performance out of their devices, in order to enjoy 3D graphics and games without any lagging. In addition, this governor can be used for ranking and benchmarking purposes to test the performance level of a certain GPU.

Powersave **Governor**

The *Powersave* governor is the most power efficient, since it sets the frequency statically to the lowest value. However, we found that setting the GPU frequency to 50 MHz is not very practical since it slows down the rendering of some interface components, and it also failed to run some of the high resolution 3D games. Therefore, through experimental testing, we have decided to choose a more reasonable frequency for the *Power*save governor, and that it 80 MHz. This governor is suitable for users who do not use their smartphones for displaying 3D graphics or playing 3D games. However, it is worth mentioning that setting the frequency at 80 MHz does not effect regular 2D graphic rending, such as in web browsing or using regular mobile applications.

Figures 3.10 3.11 display a comparison of the scaling behavior of the *Ondemand* and *Conservative* governors against the stock algorithm in both 2D and 3D modes.

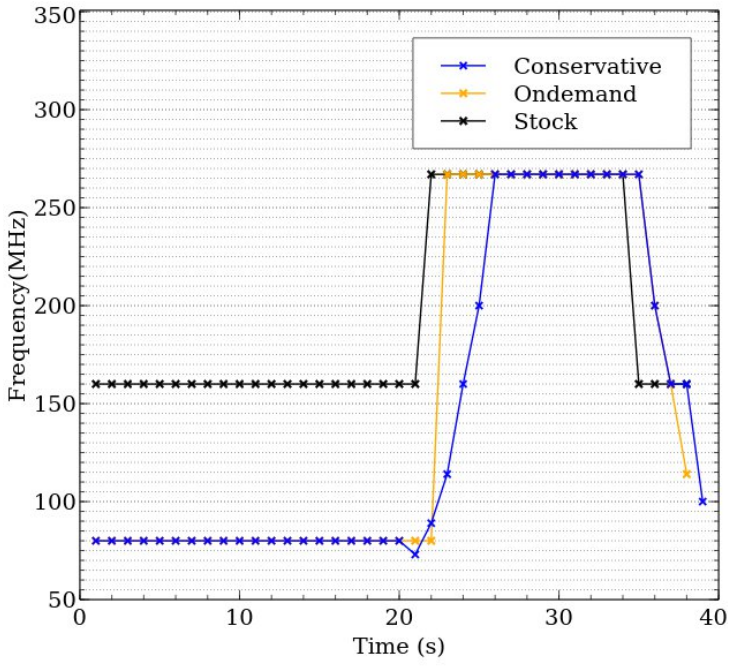


Figure 3.10: The three scaling algorithms in *AnTuTu* 2D test

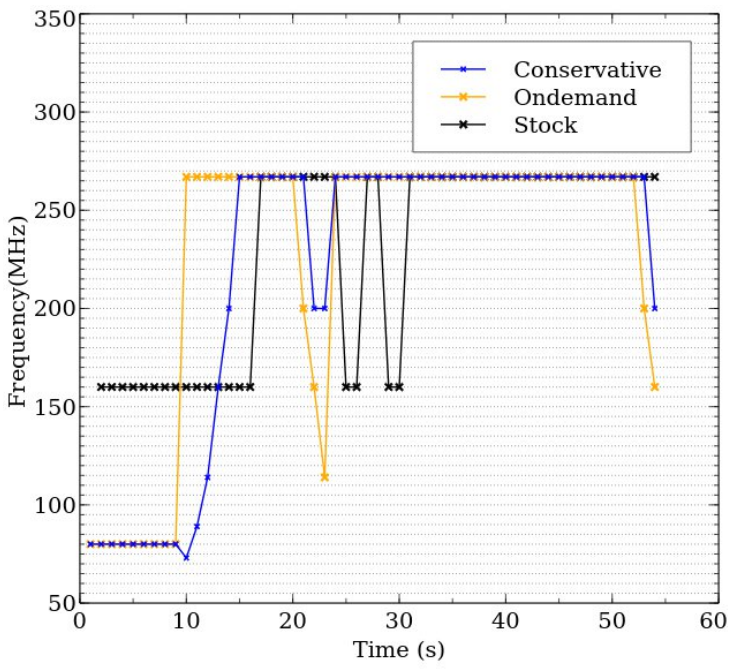


Figure 3.11: The three scaling algorithms in *AnTuTu* 3D test

3.2 Experimental Setup

3.2.1 Benchmarks

In order to test the power efficiency of each of the governors, we used three of the most well known graphics benchmark applications: A*nTuTu, Passmark,* and *TechBench*. A total of 7 different benchmark tests were conducted, which cover both 2D and 3D modes. The tests were: *AnTuTu_2D, AnTuTu_3D, Passmark_Image_Rendering, Passmark_Image_Filtering, Passmark_3D_Simple, Passmark_3D_Complex*, and finally *Techbench_3D_Cube*. Each test was repeated three times and the average of the resulted measurements was taken. In addition, the energy consumption of the idle state was measured for 30 seconds for each of the governor with the screen on.

3.2.2 Power Measurement

We have measured the total consumed energy using the Monsoon Power Monitor [42], with a sampling rate of 5kHz . The experimental setup of both software and hardware is shown in figure 3.12.

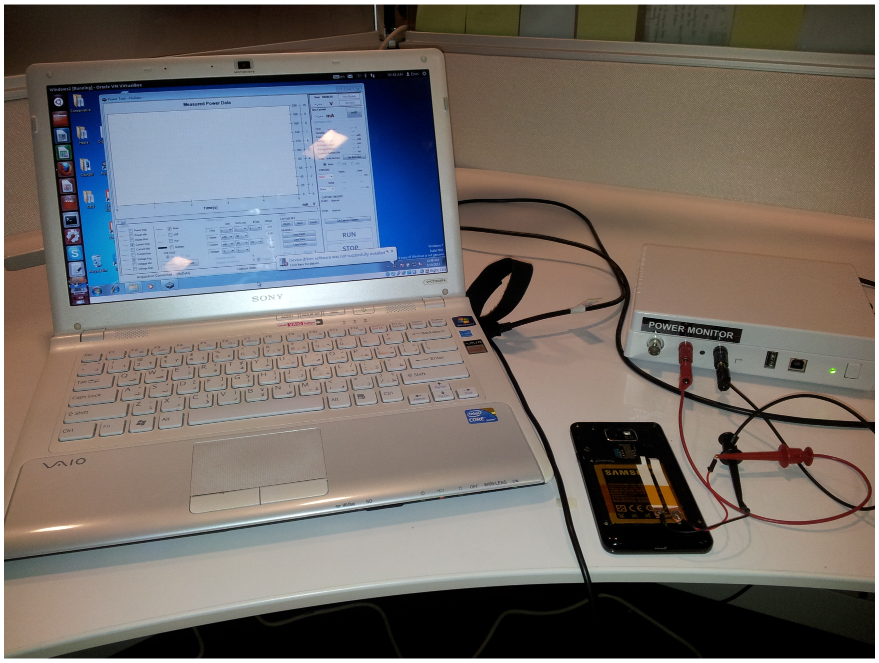


Figure 3.12: Experimental setup: Monsoon Power Monitor

In order to eliminate any variations in power consumption of the testing environment, the following measures were taken:

- SIM card removed (GSM is off)
- Flight mode activated
- Bluetooth, Wifi, and 3G off.
- No running applications in the background
- Lowest brightness of the display, with screen timeout disabled

Chapter 4

Results and Discussion

This section will present the evaluation of the GPU DVFS governors in terms of energy savings and performance. But first, in order to fully understand the mechanism of the DVFS, figure 4.1 shows the effect of varying the GPU frequency on the instantaneous power. The data was sampled while running the *AnTuTu_2D* using the *Conservative* governor. It can be seen how the gradual increase of the frequency using the *Conservative* governor reflects on the power, which increases gradually as well. The CPU jumps from 200 MHz to 1200 MHz at around the 11th second and goes back to 200 MHz, which we suspect caused the sharp spike in the power as seen in the figure around that period.

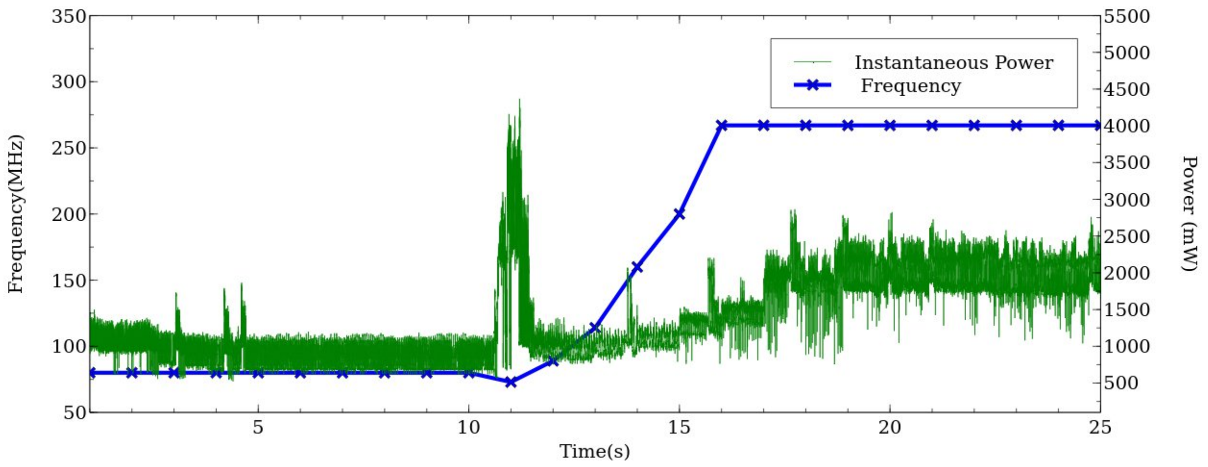


Figure 4.1: Relation between the GPU frequency and the instantaneous power

4.1 Energy Saving Evaluation

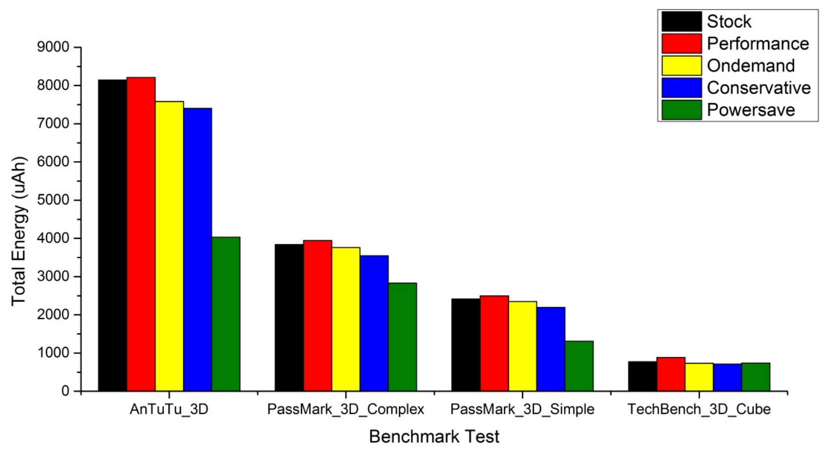


Figure 4.2: Comparison of energy consumption (3D benchmark tests)

To evaluate the energy efficiency of the GPU governors, we measured the total consumed energy for each of them running the different benchmark tests. We repeated each test three times, and then took the average value of the three readings. Figure 4.2 displays the total energy consumption results of the 3D benchmark tests. It can be clearly seen that the *Powersave* governor is the most power efficient consuming the least energy of the remaining governors. It saves up to 50.5% of energy compared with the stock algorithm running the *AnTuTu_3D*, and 45.8% running the *Passmark_3D_Complex*. The *Performance* governor however is the least power efficient, consuming 14.5% more energy than the stock algorithm running *TechBench_3D_Cube*, and a minimum of 0.83% more energy when running the *AnTuTu_3D*.

The *Conservative* governor saves between 9.2% of energy when running the *Passmark_3D_Simple* and a minimum of 7.5% running the *Passmark_3D_Complex*. But interestingly, the *Conservative* governor saves around 7.7% energy when running the *TechBench_3D_Cube*, which is more than the *Powersave* governor, that save only 4.1%. This is due to the fact that the frequency levels of the *Conservative* can

Table 4.1: Total Energy Savings in 3D Benchmark Test

	Powersave	*Conservative*	Ondemand	*Perfromance*
AnTuTu_3D	50.5%	9.1%	6.9%	-0.83%
Passmark_3D_Complex	26.3%	7.5%	0.02%	-2.8%
Passmark_3D_Simple	45.8%	9.2%	0.03%	-3.2%
TechBench_3D_Cube	4.1%	7.7%	5.1%	-14.5%

be as low as 50 MHz, while the *Powersave* is statically set to 80 MHz. Therefore using the frequencies that are lower than 80 MHz can in some cases make the *Conservative* governor more power efficient than the *Powersave*. As for the *Ondemand* governor, it behaves almost the same as the stock algorithm when running the *Passmark_3D_Complex* and *Passmark_3D_Simple*. However, it saves 6.9% and 5.1% in *AnTuTu_3D* and *TechBench_3D_Cube* in that order. Table 4.1 displays the detailed energy savings in 3D mode.

As for the 2D benchmark tests, figure 4.3 displays the total energy consumption results in 2D mode. The *Conservative* governor saves a significant amount of 15% in the *AnTuTu_2D*, 4.8% savings for the *Ondemand*, and the highest of them all 31.7% for the *Powersave*. The *Performance* governor consumes 4.0% more energy for the same benchmark test. As for the image rendering, the *Conservative* and *Ondamned* outperform the *Powersave* in terms of energy saving. This can be explained by the same reason of the lower frequency levels mentioned previously. And finally, not much difference was recorded in the energy consumption for the image filters test between the *Performance*, *Ondemand*, *Powersave*, and the stock algorithm. While around 5.2% is saved using the *Conservative* governor. Table 4.2 displays the detailed energy savings in 2D mode.

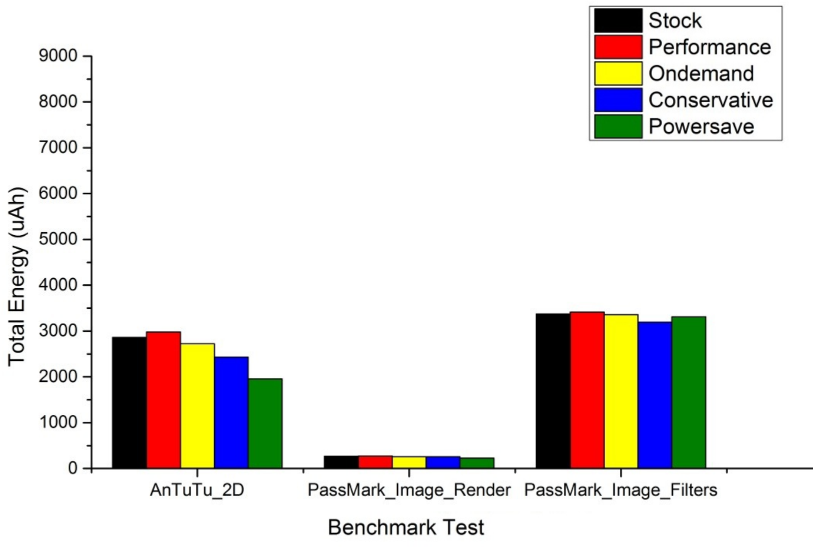


Figure 4.3: Comparison of energy consumption (2D benchmark tests)

Table 4.2: Total Energy Savings in 2D Benchmark Test

	Powersave	*Conservative*	*Ondemand*	Perfromance
AnTuTu_2D	31.7%	15.0%	4.8%	-4.0%
Passmark_Image_Render	1.3%	4.5%	4.3%	-1.9%
Passmark_Image_Filter	1.7%	5.2%	0.37%	-1.31%

The energy consumption was recorded for 30 seconds for the five different algorithms in the idle sate and the screen on, as shown in figure 4.4 . Since smartphones spend a lot of their time being idle, any percentage of idle energy saving can be considered significant. The *Ondemand* governor does not show any significant energy saving in the idle state, while the *Powersave* saves around 1.4% and the *Conservative* saves a significant amount of 3.7%. The *Performance* governor consumes 28.3% more energy, which can easily drain the battery. Table 4.3 displays the total energy saving of the idle state.

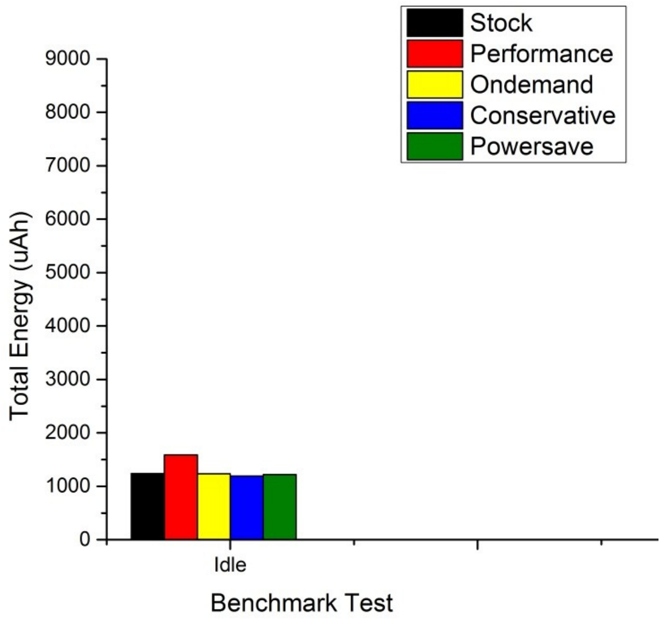


Figure 4.4: Comparison of energy consumption (idle state)

Table 4.3: Total Energy Savings in Idle State

	Powersave	*Conservative*	*Ondemand*	Perfromance
Idle	1.4%	3.75%	0.37%	-28.3%

Figure 4.5 summarizes the percentages of energy saving for all of the benchmark tests.

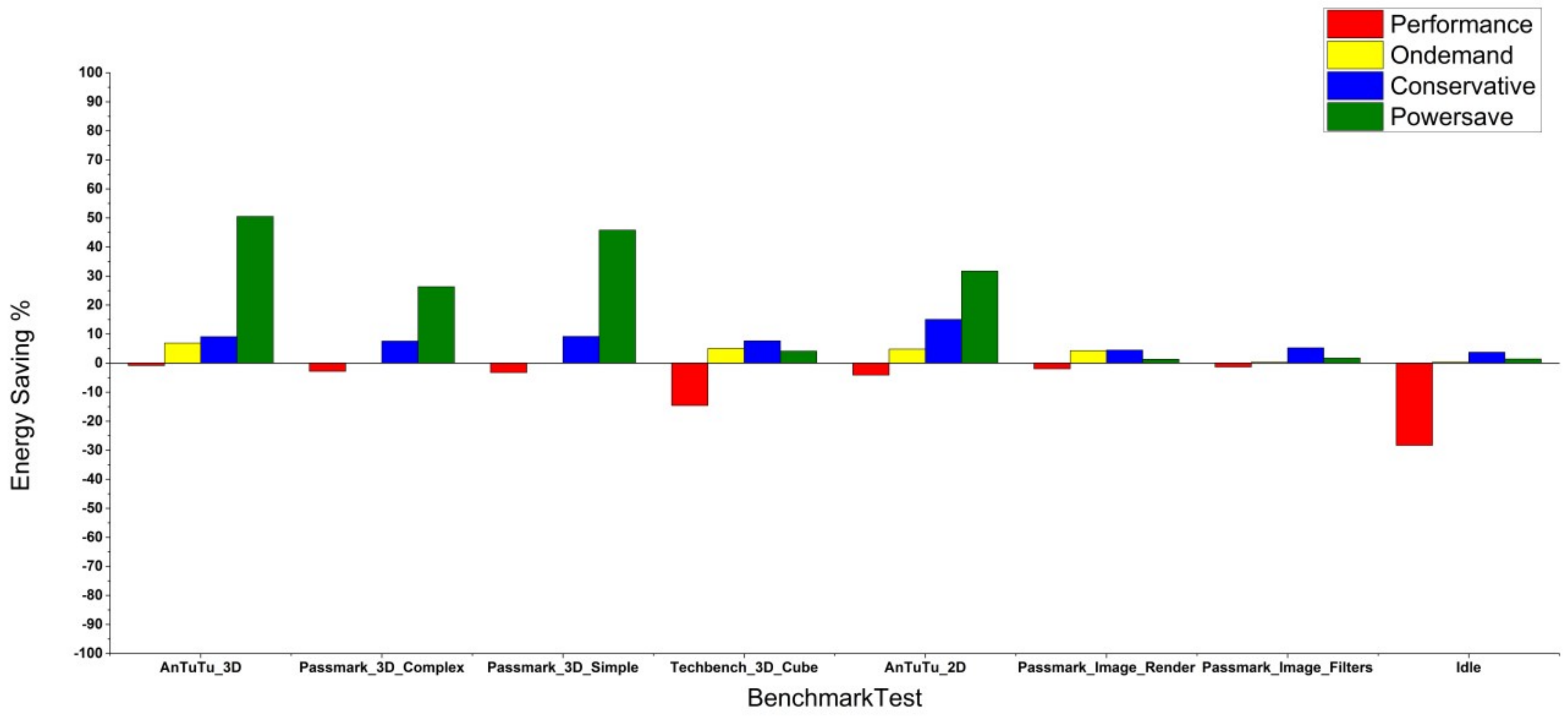


Figure 4.5: Total energy saving percentages

4.2 Performance Evaluation

Since we are using off-the-shelf benchmark applications, the only performance metric we can provide at this stage are the scores and measurements that are given by those tests. Table 4.4 displays the performance results.

Table 4.4: Benchmark Performance Results

	Stock	*Powersave*	*Conservative*	*Ondemand*	*Performance*
AnTuTu_2D	1023	474	895	1024	1094
AnTuTu_3D	4008	1231	3828	3939	4014
Passmark_Image_Render(Images/s)	1075	994	1002	1004	1230
Passmark_Image_Filters (Filters/s)	71.7	72.9	72.9	74	74.1
Passmark_3D_Simple (Frames/sec)	59.8	25.3	56.6	59.8	59.8
Passmark_3D_Complex (Frames/s)	17.3	5.34	16.4	17	17.6
TechBench_3D_Cube (Frames/s)	59.8	59.6	59.6	59.6	60.04

The *AnTuTu* results are not standardized, but are rather ranking scores given by that benchmark application, therefore we mention them only for referencing purpose. Table 4.5 shows the performance loss percentage for the remaining benchmarks. The *Ondemand* governor achieves almost the same level of performance as the stock algorithm. While the performance difference for the *Conservative* governor for all the benchmark tests do not exceed 7%. The *Powersave* governor fails in the 3D Passmark tests having more then 50% performance loss. This confirms that the *Powersave* governor is not suitable for running 3D graphics or playing 3D games. On the other hand, its performance in 2D benchmark is considered very acceptable. The *Performance* governor shows around 22.5% performance enhancement in image rendering, while not much noticeable difference was witnessed in the other tests. This is due to the fact that the *Ondemand* and *Conservative* governors can scale to the highest frequency level used by the *Performance* governor when needed, which is why they are able to compete in terms of performance.

Table 4.5: Performance Loss Percentages

	Powersave	*Conservative*	*Ondemand*	*Performance*
Passmark_Image_Render	-0.99%	-0.19%	0%	22.5%
Passmark_Image_Filters	-1.49%	-1.49%	0%	0.14%
Passmark_3D_Simple	-57.69%	-5.35%	0%	0.14%
Passmark_3D_Complex	-69.13%	-5.20%	-1.73%	1.73%
TechBench_3D_Cube	-0.33%	-0.33%	-0.33%	0.40%

In conclusion, the above results prove that by dynamically scaling the frequency and voltage of the GPU using the governors has a major effect on the battery life time. In addition, the governors offer wide range of options for the users to choose from according to their personal preferences. Using the *Ondemand* governor will maintain the same level of performance and high responsiveness, while saving up to 7% of energy in 3D mode, and up to 5% of energy in 2D mode. Moreover, using the *Conservative* governor saves energy up to 15% in 2D mode, and 9% in 3D mode with less than 7% effect on the performance. And finally, if the user is only interested in 2D rendering then using the *Powersave* governor would save up to 31% of energy, with maintaining an acceptable level of performance.

Chapter 5

Concluding Remarks

5.1 Conclusion

Smartphone GPUs are advancing in their computational and processing power. However, like almost all the other components of modern smartphones, users are not using them to their full capacity the entire time. But unfortunately they are paying the price of such improvements in terms of battery power. Therefore, users should be given the ability to scale the capabilities of the different components of their phones according to their usage behavior and personal preference. In this study we have proposed the novel implementation of the GPU DVFS governors, similar to the linux-based CPUFreq governors, to dynamically scale the smartphone GPU. To the best of our knowledge our work is the first to present a quantitative analysis of the effectiveness of different DVFS algorithms tested on a real smartphone GPU, with the energy consumption accurately measured using an external power monitor device. Our results show that the energy efficiency of the GPU can be enhanced up to 15% in 2D mode, and 9% in 3D mode with less than 7% effect on the performance using the *Conservative* governor. Moreover, using the *Powersave* governor would save up to 31% of energy in 2D mode, while maintaining an acceptable level of image rendering.

5.2 Future Research Work

In this work we have only implemented the four main governors of the original Linux CPUFreq layer. Never the less, there could be numerous enhancements on these basic algorithms in order to find the most optimal energy efficient GPU governor. Moreover, instead of using off-the-shelf benchmark test applications, we plan to use our own Android OpenGL API test code, which could give more in depth analysis and accurate profiling of the GPU. In addition, to get a more general view of the total energy efficiency of the smartphone, the association between different GPU and CPU governors can be evaluated, and then the most optimal combination can be proposed.

5.3 Contributing to the Android Community

Since Android is an open source project, it welcomes contributions to their source code from institutions or individuals. Theretofore, we are studying the possibility of proposing our implementation of the GPU governors to be added to the Android platform. However, the main challenge is that the GPU drivers are very device specific. We have used the Mali-400 GPU to prove the effectiveness of our approach, but we are looking in to ways of making the GPU governors scheme more general and independent of any specific device driver. We believe that by doing so, the GPU DVFS governors would make a very significant contribution to the Android platform in terms of energy efficiency.

REFERENCES

[1] P. Lasewicz. (2012) Worlds first smartphone = apple or android? think again. The Greater IBM Connection. [Online]. Available: http://greateribm.wordpress.com/tag/ibm-simon

[2] L. Goasduff and C. Pettey. (2010) Gartner says worldwide mobile phone sales grew 35 percent in third quarter 2010; smartphone sales increased 96 percent. Gartner, Inc. [Online]. Available: http://www.gartner.com/newsroom/id/1466313

[3] A. Rahmati, A. Qian, and L. Zhong, "Understanding human-battery interaction on mobile phones," in *Proceedings of the 9th international conference on Human computer interaction with mobile devices and services*, ser. MobileHCI '07. New York, NY, USA: ACM, 2007, pp. 265–272. [Online]. Available: http://doi.acm.org/10.1145/1377999.1378017

[4] A. Carroll and G. Heiser, "An analysis of power consumption in a smartphone," in *Proceedings of the 2010 USENIX conference on USENIX annual technical conference*, ser. USENIXATC'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 21–21. [Online]. Available: http://dl.acm.org/citation.cfm?id=1855840.1855861

[5] G. P. Perrucci, F. H. P. Fitzek, and J. Widmer, "Survey on energy consumption entities on the smartphone platform," in *Vehicular Technology Conference (VTC Spring), 2011 IEEE 73rd*, 2011, pp. 1–6.

[6] C. Yoon, D. Kim, W. Jung, C. Kang, and H. Cha, "Appscope: Application energy metering framework for android smartphone using kernel activity monitoring," in *Proc. USENIX Annual Technical Conference*, 2012.

[7] R. Mittal, A. Kansal, and R. Chandra, "Empowering developers to estimate app energy consumption," in *Proceedings of the 18th annual international conference on Mobile computing and networking.* ACM, 2012, pp. 317–328.

[8] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang, "Accurate online power estimation and automatic battery behavior based power model generation for smartphones," in *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, ser. CODES/ISSS '10. New York, NY, USA: ACM, 2010, pp. 105–114. [Online]. Available: http://doi.acm.org/10.1145/1878961.1878982

[9] A. Pathak, Y. C. Hu, and M. Zhang, "Where is the energy spent inside my app?: fine grained energy accounting on smartphones with eprof," in *Proceedings of the 7th ACM european conference on Computer Systems*, ser. EuroSys '12. New York, NY, USA: ACM, 2012, pp. 29–42. [Online]. Available: http://doi.acm.org/10.1145/2168836.2168841

[10] R. Zahir and P. Ewert, "The medfield smartphone: Intel; architecture in a handheld form factor," pp. 1–1, 2013.

[11] Hardware acceleration. Android Developers. [Online]. Available: http://developer.android.com/guide/topics/graphics/hardware-accel.html

[12] D. Brodowski and N. Golde. Linux cpufreq governors. The Linux Kernel Archives. [Online]. Available: https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt

[13] M. Wakihara and O. Yamamoto, *Lithium ion batteries.* Wiley-Vch, 2008.

[14] V. Kononen and P. Paakkonen, "Optimizing power consumption of always-on applications based on timer alignment," in *Communication Systems and Networks (COMSNETS), 2011 Third International Conference on*, 2011, pp. 1–8.

[15] J. Paek, J. Kim, and R. Govindan, "Energy-efficient rate-adaptive gps-based positioning for smartphones," in *Proceedings of the 8th international conference on Mobile systems, applications, and services*, ser. MobiSys

'10. New York, NY, USA: ACM, 2010, pp. 299–314. [Online]. Available: http://doi.acm.org/10.1145/1814433.1814463

[16] U. Bareth and A. Kupper, "Energy-efficient position tracking in proactive location-based services for smartphone environments," in *Computer Software and Applications Conference (COMPSAC), 2011 IEEE 35th Annual*, 2011, pp. 516–521.

[17] A. Pathak, Y. C. Hu, and M. Zhang, "Bootstrapping energy debugging on smartphones: a first look at energy bugs in mobile devices," in *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*, ser. HotNets-X. New York, NY, USA: ACM, 2011, pp. 5:1–5:6. [Online]. Available: http://doi.acm.org/10.1145/2070562.2070567

[18] A. Pathak, A. Jindal, Y. C. Hu, and S. P. Midkiff, "What is keeping my phone awake?: characterizing and detecting no-sleep energy bugs in smartphone apps," in *Proceedings of the 10th international conference on Mobile systems, applications, and services*, ser. MobiSys '12. New York, NY, USA: ACM, 2012, pp. 267–280. [Online]. Available: http://doi.acm.org/10.1145/2307636.2307661

[19] H. Kim, J. Smith, and K. G. Shin, "Detecting energy-greedy anomalies and mobile malware variants," in *Proceedings of the 6th international conference on Mobile systems, applications, and services*, ser. MobiSys '08. New York, NY, USA: ACM, 2008, pp. 239–252. [Online]. Available: http://doi.acm.org/10.1145/1378600.1378627

[20] X. Zhang and K. G. Shin, "E-mili: Energy-minimizing idle listening in wireless networks," *Mobile Computing, IEEE Transactions on*, vol. 11, no. 9, pp. 1441–1454, 2012.

[21] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani, "Energy consumption in mobile phones: a measurement study and implications for network applications," in *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, ser. IMC '09. New York, NY, USA: ACM, 2009, pp. 280–293. [Online]. Available: http://doi.acm.org/10.1145/1644893.1644927

[22] Y. Agarwal, R. Chandra, A. Wolman, P. Bahl, K. Chin, and R. Gupta, "Wireless wakeups revisited: Energy management for voip over wi-fi smartphones," in *PROC. 5 TH INTERNATIONAL CONFERENCE ON MOBILE SYSTEMS, APPLICATIONS AND SERVICES (MOBISYS 07), PUERTO RICO*, 2007, pp. 179–191.

[23] R. Duan, M. Bi, and C. Gniady, "Exploring memory energy optimizations in smartphones," *2012 International Green Computing Conference (IGCC)*, vol. 0, pp. 1–8, 2011.

[24] S. Swanson and M. Taylor, "Greendroid: Exploring the next evolution in smartphone application processors," *Communications Magazine, IEEE*, vol. 49, no. 4, pp. 112–119, 2011.

[25] W.-Y. Liang and P.-T. Lai, "Design and implementation of a critical speed-based dvfs mechanism for the android operating system," in *Embedded and Multimedia Computing (EMC), 2010 5th International Conference on*, 2010, pp. 1–6.

[26] T. Mittal, L. Singhal, and D. Sethia, "Optimized cpu frequency scaling on android devices based on foreground running application," in *Computer Networks & Communications (NetCom).* Springer, 2013, pp. 827–834.

[27] T. Akenine-Moller and J. Strom, "Graphics processing units for handhelds," *Proceedings of the IEEE*, vol. 96, no. 5, pp. 779–789, 2008.

[28] J. Pool, A. Lastra, and M. Singh, "Energy-precision tradeoffs in mobile graphics processing units," in *Computer Design, 2008. ICCD 2008. IEEE International Conference on.* IEEE, 2008, pp. 60–67.

[29] V. M. del Barrio, C. González, J. Roca, A. Fernández, and E. Espasa, "Attila: a cycle-level execution-driven simulator for modern gpu architectures," in *Performance Analysis of Systems and Software, 2006 IEEE International Symposium on.* IEEE, 2006, pp. 231–241.

[30] Y. Gu, S. Chakraborty, and W. T. Ooi, "Games are up for dvfs," in *Design Automation Conference, 2006 43rd ACM/IEEE*, 2006, pp. 598–603.

[31] Quake ii. id Software. [Online]. Available: http://www.idsoftware.com/gate.php?referer=%2Fgames%2Fquake%2Fquake2

[32] B. Mochocki, K. Lahiri, and S. Cadambi, "Power analysis of mobile 3d graphics," in *Design, Automation and Test in Europe, 2006. DATE'06. Proceedings*, vol. 1. IEEE, 2006, pp. 1–6.

[33] B. Silpa, G. Krishnaiah, and P. R. Panda, "Rank based dynamic voltage and frequency scaling fortiled graphics processors," in *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*. ACM, 2010, pp. 3–12.

[34] D. Grunwald, C. B. Morrey, III, P. Levis, M. Neufeld, and K. I. Farkas, "Policies for dynamic clock scheduling," in *Proceedings of the 4th conference on Symposium on Operating System Design & Implementation - Volume 4*, ser. OSDI'00. Berkeley, CA, USA: USENIX Association, 2000, pp. 6–6. [Online]. Available: http://dl.acm.org/citation.cfm?id=1251229.1251235

[35] Y. Jiao, H. Lin, P. Balaji, and W. Feng, "Power and performance characterization of computational kernels on the gpu," in *Green Computing and Communications (GreenCom), 2010 IEEE/ACM Int'l Conference on Int'l Conference on Cyber, Physical and Social Computing (CPSCom)*, 2010, pp. 221–228.

[36] P. Pillai and K. G. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems," in *Proceedings of the eighteenth ACM symposium on Operating systems principles*, ser. SOSP '01. New York, NY, USA: ACM, 2001, pp. 89–102. [Online]. Available: http://doi.acm.org/10.1145/502034.502044

[37] Android architecture. Android Developers. [Online]. Available: http://developer.android.com/about/versions/index.html

[38] Android open source project. Android. [Online]. Available: http://source.android.com

[39] Mali-400 mp. ARM - The Architecture For The Digital World. [Online]. Available: http://www.arm.com/products/multimedia/mali-graphics-hardware/mali-400-mp.php

[40] Mali gpu device driver model. Mali Developer Center. [Online]. Available: http://malideveloper.arm.com/develop-for-mali/drivers/mali-device-driver-model/

[41] Samsung open source release center. Samsung. [Online]. Available: http://opensource.samsung.com/

[42] Power monitor. Monsoon Solutions Inc. [Online]. Available: http://www.msoon.com/LabEquipment/PowerMonitor/