UNIVERSITY OF CALIFORNIA, SAN DIEGO

Power, Thermal, Reliability and Variability Management of Mobile Devices

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Computer Science

by

Pietro Mercati

Committee in charge:

Professor Tajana Šimunić Rosing, Chair
Professor Chung-Kuan Cheng
Professor Sujit Dey
Professor Rajesh Gupta
Professor Yuanyuan Zhou

2017

The dissertation of Pietro Mercati is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

_____

_____

_____

_____

_____

Chair

University of California, San Diego

2017

DEDICATION

To my father Massimo, my mother Angela and my brother Alberto.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to thank all the people that one way or another contributed to my PhD. First, I want to sincerely thank my advisor, Prof. Tajana Šimunić Rosing. From the very beginning, she believed in me more than I did myself, and motivated me to take chances and face challenges. She provided inestimable guidance during my research and she constantly encouraged and supported me. I feel extremely lucky that I met her and that she let me be a part of her research team. I also want to thank my doctoral committee, Prof. Chung-Kuan Cheng, Prof. Sujit Dey, Prof. Rajesh Gupta and Prof. Yuanyuan Zhou for their valuable feedback and contributions to my PhD.

I want to thank the people that more closely helped me with my research. I want to thank Francesco Paterna for his help and support, from Bologna, to San Diego, to Portland. Thanks to Andrea Bartolini for providing feedback and collaboration during the PhD years. A special thanks to Prof. Luca Benini for his trust, help and guidance. I would not have ended up in the US if it wasn't for him. Thanks to the researchers that I worked with in Samsung, Vinay Hanumaiah, Jitendra Kulkarni, Simon Bloch, Charles Briere, Robert Hasslen. Thanks to my manager at Intel, Michael Kishinevski, and my supervisor Raid Ayoub.

I really want to thank my colleagues and friends at SEELab, both current and past, for their comments, help and support. I feel extremely lucky that I had a chance to work side by side with such smart people. I owe special thanks to Yeseong Kim, Christine Chan, Mohsen Imani and Henrique Rodrigues.

I want to thank my friends all around the world, who contributed to make my PhD years an awesome journey. In particular, I owe special thanks to Lorenzo Ferrari, Diletta Giuntini, Francesco Fraternali, Negin Nazarian, Teresa Dormi, Andrea Alberti.

Last, but most importantly, I want to sincerely thank my father Massimo and my mother Angela for always being there and providing unconditioned love and support. Thanks to my brother Alberto, who is by far the smartest person I know. Thanks to my grandparents Fosco and Bianca, Giuseppe and Renata.

Chapter 2 contains material from "BLAST: Battery Lifetime-constrained Adaptation with Selected Target", by Pietro Mercati, Vinay Hanumaiah, Jitandra Kulkarni, Simon Bloch and Tajana Šimunić Rosing, which appears in Proceedings of the 12th EAI

International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services on 12th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MOBIQUITOUS'15) [111]. The dissertation author was the primary investigator and author of this paper.

Chapter 3 contains material from "User-centric joint power and thermal management for smartphones", by Pietro Mercati, Vinay Hanumaiah, Jitandra Kulkarni, Simon Bloch and Tajana Šimunić Rosing, which appears in Proceedings of the 6th International Conference on Mobile Computing, Applications and Services (MobiCASE), 2014, Austin, TX, 2014. [108]. The dissertation author was the primary investigator and author of this paper.

Chapter 4 contains material from "WARM: Workload-Aware Reliability Management in Linux/Android", by Pietro Mercati, Francesco Paterna, Andrea Bartolini, Luca Benini and Tajana Šimunić Rosing, which appears in IEEE Transactions on Computer-Aided Design of Integrated Circuits and System [106]. The dissertation author was the primary investigator and author of this paper.

Chapter 4 contains material from "Workload and user experience-aware dynamic reliability management in multicore processors", by Pietro Mercati, Andrea Bartolini, Francesco Paterna, Luca Benini and Tajana Šimunić Rosing, which appears in Proceedings of the 50th Annual Design Automation Conference (DAC '13). ACM, New York, NY, USA [110]. The dissertation author was the primary investigator and author of this paper.

Chapter 4 contains material from "A Linux-governor based Dynamic Reliability Manager for android mobile devices", by Pietro Mercati, Andrea Bartolini, Francesco Paterna, Luca Benini and Tajana Šimunić Rosing, which appears in Proceedings of Design, Automation & Test in Europe Conference & Exhibition (DATE), Dresden, 2014 [105]. The dissertation author was the primary investigator and author of this paper.

Chapter 4 contains material from "An On-line Reliability Emulation Framework", by Pietro Mercati, Andrea Bartolini, Francesco Paterna, Luca Benini and Tajana Šimunić Rosing, which appears in Proceedings of the 2014 12th IEEE International Conference on Embedded and Ubiquitous Computing (EUC '14). IEEE Computer So-

ciety, Washington, DC, USA [109]. The dissertation author was the primary investigator and author of this paper.

Chapter 5 contains material from "Dynamic Variability Management in Mobile Multicore Processors under Lifetime Constraints", by Pietro Mercati, Francesco Paterna, Andrea Bartolini, Luca Benini and Tajana Šimunić Rosing, which appears in Proceedings of the 32nd IEEE International Conference on Computer Design (ICCD), Seoul, 2014 [107]. The dissertation author was the primary investigator and author of this paper.

Chapter 5 contains material from "VarDroid: Online Variability Emulation in Android/Linux Platforms", by Pietro Mercati, Mohsen Imani, Francesco Paterna, Andrea Bartolini, Luca Benini and Tajana Šimunić Rosing, which appears in Proceedings of the 26th edition on Great Lakes Symposium on VLSI (GLSVLSI '16). ACM, New York, NY, USA [112]. The dissertation author was the primary investigator and author of this paper.

VITA

2010	B. S. in Electrical and Computer Engineering *cum laude*, University of Bologna, Italy
2012	Student Researcher, Micrel Lab, University of Bologna, Italy
2013	M. S. in Electrical and Computer Engineering *cum laude*, University of Bologna, Italy
2013-2017	Graduate Student Researcher,University of California, San Diego
2014	Intern, Advanced Systems Engineering Lab (ASEL), Samsung Research America, San Jose, CA
2015	Intern, Advanced Processor Lab (APL), Samsung Research America, Mountain View, CA
2016	Intern, Strategic CAD Lab (SCL), Intel Corporation, Hillsboro, OR
2016	Instructor, University of California, San Diego
2017	Ph. D. in Computer Science, University of California, San Diego

PUBLICATIONS

Pietro Mercati, Andrea Bartolini, Francesco Paterna, Luca Benini and Tajana Šimunić Rosing "Workload and User Experience-Aware Dynamic Reliability Management in Multicore Processors", ACM Proceedings of the International Conference Design Automation Conference (DAC) 2013.

Pietro Mercati, Andrea Bartolini, Francesco Paterna, Luca Benini and Tajana Šimunić Rosing "A Linux-Governor Based Dynamic Reliability Manager for Android Mobile Devices", IEEE Proceedings of the International Conference on Design, Automation and Test in Europe Conference and Exhibition (DATE) 2014.

Pietro Mercati, Andrea Bartolini, Francesco Paterna, Luca Benini and Tajana Šimunić Rosing "An On-line Reliability Emulation Framework", IEEE Proceedings of the International Conference on Embedded and Ubiquitous Computing (EUC) 2014.

Pietro Mercati, Francesco Paterna, Andrea Bartolini, Luca Benini and Tajana Šimunić Rosing "Dynamic Variability Management in Mobile Multicore Processors under Lifetime Constraints", IEEE Proceedings of the International Conference on Computer Design (ICCD) 2014.

Pietro Mercati, Vinay Hanumaiah, Jitendra Kulkarni, Simon Bloch and Tajana Šimunić Rosing "User-centric joint power and thermal management for smartphones", 6th International Conference on Mobile Computing, Applications and Services (MobiCASE) 2014.

Lucas Wanner, Liangzhen Lai, Abbas Rahimi, Mark Gottscho, Pietro Mercati, Chu-Hsiang Huang, Frederic Sala, Yuvraj Agarwal, Lara Dolecek, Nikil Dutt, Puneet Gupta, Rajesh Gupta, Ranjit Jhala, Rakesh Kumar, Sorin Lerner, Subhasish Mitra, Alexandru Nicolau, Tajana Šimunić Rosing, Mani B Srivastava, Steve Swanson, Dennis Sylvester, Yuanyuan Zhou. "NSF Expedition of Variability-Aware Software:Recent Results and Contributions", Information Technology, vol. 57, no. 3, pp. 181-198, 2015.

Pietro Mercati, Vinay Hanumaiah, Jitendra Kulkarni, Simon Bloch, and Tajana Šimunić Rosing. "BLAST: Battery Lifetime-constrained Adaptation with Selected Target in Mobile Devices", Proceedings of the 12th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (Mobiquitous) 2015.

Mohsen Imani, Pietro Mercati and Tajana Šimunić Rosing, "ReMAM: Low energy Resistive Multi-stage Associative Memory for energy efficient computing", Proceedings of the International Symposium on Quality Electronic Design (ISQED) 2016.

Pietro Mercati, Francesco Paterna, Andrea Bartolini, Mohsen Imani, Luca Benini and Tajana Šimunić Rosing, "VarDroid: Online variability emulation in Android/Linux platforms", Proceedings of the International Great Lakes Symposium on VLSI (GLSVLSI) 2016.

Pietro Mercati, Francesco Paterna, Andrea Bartolini, Luca Benini, Tajana Šimunić Rosing, "WARM: Workload-Aware Reliability Management in Linux/Android", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD) 2016.

ABSTRACT OF THE DISSERTATION

Power, Thermal, Reliability and Variability Management of Mobile Devices

by

Pietro Mercati

Doctor of Philosophy in Computer Science

University of California, San Diego, 2017

Professor Tajana Šimunić Rosing, Chair

Today, there are more mobile devices than human beings on the planet. Mobiles execute a wide variety of applications and are expected to perform in many different environments, from the deserts to the mountain tops. Their goal is to meet user's expectations and deliver high quality of experience. Unfortunately, to achieve this they face a set of interrelated problems. The heterogeneous components that execute varied tasks can drain a battery in a matter of hours. Power dissipation raises the device temperature, which can be a source of discomfort for the user. Temperature stress also dramatically increases the impact of reliability degradation mechanisms on transistors and interconnects, which can lead to early failure. These problems only worsen with CMOS scaling, which reduces the accuracy of the fabrication process and increases the variability in power, performance and degradation rate. Dynamic management mitigates such issues

by adapting the operating conditions at runtime. Strategies have been very well studied for traditional systems like desktops and servers, but unfortunately they cannot be applied to mobiles, because they do not consider user experience. Existing dynamic management for mobiles only begins to target the problems of power and thermal management, and offers limited solutions for reliability and variability management. This is fundamental to guarantee power savings and prevent early failure in a scenario of increasing variability.

In this dissertation, we propose the design and real implementation of a novel unified framework for the comprehensive dynamic management of power, temperature, reliability and variability in mobile systems, subject to user experience requirements. We develop novel lightweight solutions for power and thermal management that optimize for both application behavior and battery lifetime. We complement this with our framework for the online emulation of reliability degradation and variability, which enables the development and rapid prototyping of hardware management solutions. We leverage our emulation framework to design and integrate dynamic reliability and variability management for mobile systems. The presented solutions have been implemented and tested on real devices. Our strategy can meet user experience requirements with a selected target battery lifetime extension of at least 25%. Also, it achieves up to 35% savings in power consumption at the device level, with up to 1 year reliability lifetime improvement for a multicore platform and up to 100% of performance improvement on cluster architectures. Finally, on devices affected by variability it achieves up to 160% performance improvement over the state-of-the-art while meeting the lifetime constraints.

Chapter 1

Introduction

Mobile devices experienced an impressive development during recent years, to the point at which there are more smartphones and tablets than human beings on the planet. From the first cell phones in the 70s and 80s that only had capability of sending and receiving calls, today we can hold an entire computer on the palm of our hand. Modern smartphones integrate heterogeneous subsystems in their multiprocessor systems on chip (MPSoC), such as CPU, GPU, DSPs, display, networking (Wifi, 4G, Bluetooth) in a compact form factor. They can execute a variety of applications, from traditional calls, to multimedia, gaming, browsing and more. Such different applications have varied performance requirements and power consumption [74].

The ultimate goal of mobiles is to deliver quality user experience [49]. Optimizing performance increases the activity of computation units and their peak power consumption, which can drain the battery in a few hours. Power dissipation raises the device temperature quickly, due to the compact form factor and the absence of active cooling solutions. This can be a source of discomfort for the user, since human skin tolerates up to 45^{o}C contact temperature [129]. Mobile devices have to provide the same level of user experience in environments and seasons characterized by very different external temperature, as opposed to desktops and server machines [130]. High temperature stress also dramatically increases the impact of reliability degradation mechanisms of transistors and interconnects. Mechanisms such as Time Dependent Dielectric Breakdown (TDDB) and Negative Bias Temperature Instability (NBTI) depend exponentially on temperature. They degrade the performance of circuits over time and lead them to

early failure [153]. These interrelated problems only worsen with the scaling of CMOS technology, which decreases the accuracy of the fabrication process. As a result, processors with the same nominal characteristics, in reality have variability in performance, power consumption and reliability degradation rate [116].

Past research investigated the dynamic management of these problems for traditional systems, such as single core and multicore processors for desktop and server machines [157], [21], [23], [62], [176]. Unfortunately, these approaches cannot be applied to mobiles, because they do not consider user experience. More recent efforts look at energy and temperature management for mobile devices [130], [129]. Very little work looks at the joint control of these problems subject to user experience constraints. Limited solutions are presented for the dynamic management of reliability and variability. A few publications explore the feasibility of implementation of these techniques on real mobile devices [131], [86], [121].

In this dissertation, we propose the design and real implementation of a unified framework for the comprehensive dynamic management of power, temperature, reliability and variability in mobile systems, subject to user experience requirements. We develop novel lightweight solutions for power and thermal management that account for both application behavior and battery lifetime. We propose a framework for the online emulation of reliability degradation and variability, which enables the development and rapid prototyping of hardware management solutions. Finally, using this emulation framework, we propose a dynamic reliability and variability management for mobile systems. All the solutions presented in this dissertation have been implemented and tested on real devices. We demonstrate that the proposed strategy meets user experience requirements with a selected target battery lifetime extension of at least 25%. It can achieve up to 35% average savings in power consumption. We also show that it achieves up to 1 year reliability lifetime improvement for a multicore platform, and up to 100% of performance improvement on cluster architectures. Finally, we demonstrate that it achieves up to 160% performance improvement over the state-of-the-art while meeting the lifetime constraints on devices affected by variability.

Next, we discuss the key issues addressed by this dissertation: user experience, power, temperature, reliability, variability. We highlight the main challenges in the con-

text of mobile devices and describe how these problems are related to each other. We conclude the chapter by presenting our solution and the main results achieved.

1.1 Mobile Devices and User Experience

Providing quality user experience is the main goal of mobile devices [17]. This depends on how quality is perceived by users. Quality user experience is achieved when the device behavior meets user's expectations. User's perception is highly variable across people of different age, gender, culture and ethnicity. It also depends on contextual and environmental factors. To achieve higher energy efficiency, the device behavior should dynamically adapt to the level of quality perception of the user. A model for user experience is then needed to adjust operating conditions at runtime without generating discomfort.

New dynamic management techniques should be implemented on real devices. This is the only way in which it is possible to close the loop with the user and get a real feedback in terms of user experience. Also, since the user experience for mobiles highly depends on the content of the foreground application, for development and prototyping it is reasonable to assume that each application has its own required level of performance, and that such level reflects the single user's expectations. For example, for dynamic management, the Linux kernel has drivers that manage operating conditions during system operations. Such drivers provide an interface between applications and hardware regulators, and transparently enable the dynamic switching of variables such as operating frequency of different components, number of active computing units, idleness, power state residency and more [125]. The action of these drivers should be regulated at runtime depending on application-specific user experience requirements. To implement dynamic solutions at the level of the operating system and achieve fast prototyping and user's response, it is necessary to have access to the operating system source code and to be able to identify and modify the components taking care of the runtime management of hardware resources.

For video and gaming, resolution and frame rate play a fundamental role in quality perception [45]. For example, 60 frames per second (FPS) is a target for mobile gam-

ing, as it corresponds to the refresh rate of the screen. The quality of experience then drops drastically below 30 FPS [19]. FPS is not the only indicator for video quality. Recent studies based on EEG monitoring revealed that also abrupt changes in quality may cause discomfort for the user [142]. It has been shown that also changes in the environment surrounding the user have an impact on his perception of video quality [171]. Moreover, recent studies have pointed out differences in perception dictated by age and gender [118].

Quality assessment of images is more challenging when considering 3D videos. This should account for effects such as incorrect stereography, binocular rivalry and depth misperception [139]. The study in [40] also investigated how configuration and visualization parameters, such as as different disparities, amount of parallax, monitor sizes and visualization angles, may influence the quality of experience of a 3D interactive environment. Finally, quality perception shows interesting variations when visual and audio content are delivered together. Studies on cross-modal interaction reveal that audio and video content are highly interrelated [137]. For example, a low resolution and low FPS video generate tolerable discomfort for users as long as the audio associated with it has high quality. Unfortunately, in general mobile applications do not have simple and reliable metrics such as FPS to quantify user experience.

References [95] and [96] propose a simple and general, yet effective model. Each application is characterized by a target set of maximum operating conditions, which is profiled by interviewing a large sample of people using the most common applications. For example, the authors find out that applications such as browser require lower performance level to still be rated as “good” by users. In publication [87], this approach is improved by automatically distinguishing different phases. Therefore, online management can provide further savings by lowering performance during application phases that are less critical to user experience. Application performance is not the only factor impacting user experience. High performance can drain the battery in a short time, which can also be a cause of discomfort for the user [49].

1.2 Mobile Power Management

Improving energy efficiency in mobiles is very challenging due to variable workload, environmental conditions and user experience requirements. A solution is represented by Dynamic Power Management (DPM), which adapts operating conditions at runtime to improve energy efficiency while meeting performance requirements. In AC-powered systems, such as desktops and servers, it results in a lower electricity bill and reduced pollution. For mobiles, it is fundamental in order to achieve improved battery lifetime, while meeting performance and usability requirements [115].

The main contributors to power consumption in mobiles are networking, display, CPU and GPU [33]. Most of the subsystems in a mobile device have control variables that can be tuned at runtime to change the impact on power consumption. For example, most smartphones can dim the screen brightness and turn antennas on or off. For computing units like CPU and GPU, modern MPSoC have the capability of switching frequency dynamically and power gating individual cores.

Power management for systems like desktops and servers has been extensively investigated in research, leading to the development of well-established techniques, summarized in reference [23]. Research on power management for mobiles is more recent and presents some major differences with respect to servers and desktops [158]. Other than being powered by a battery, mobiles are characterized by a tight interaction with the user. The goal is not to always provide high performance, but rather to provide the level of performance to meet user experience requirements. Mobiles have limited resources, which requires management algorithms to have a low memory overhead and execute in very short times. Most techniques implemented in real devices today are reactive, meaning that they take control decisions based on the current and past history of the system, leading to suboptimal decisions. Existing predictive techniques are difficult to implement and require the development of runtime models that are accurate and have a low overhead.

Power management is important not only for making the battery lifetime longer, but also for reducing peaks that can quickly increase the device temperature. High temperature can be also a source of discomfort, if the skin contact temperature exceeds 45^{o}C. It also increases the impact of reliability degradation mechanisms which might

lead to early hardware failure.

1.3 Temperature Modeling and Management

Power dissipated by electronic circuits is converted into heat causing an increase in the device temperature. High temperatures negatively impact performance, raise leakage power consumption and can damage circuits [48]. When temperature reaches a critical threshold, the system needs to be cooled down. In servers and enterprise systems, this is achieved with the use of fans and air conditioning. Most desktops and laptops also use fans to quickly remove the heat spread by computing circuits. Since the temperature of circuits has a linear dependence on power, it can also be reduced by managing operating conditions such as operating frequency and number of active cores.

In the case of mobiles, there are additional problems that make thermal control particularly challenging. First, since mobiles are battery powered and have a reduced form factor, they cannot implement active cooling solutions such as fans, but can only rely on dynamic thermal management. Due to limited memory and computational resources, thermal models for predictive approaches should have a low overhead, which motivates the use of model identification techniques. Finally, high temperatures are a concern for skin contact, since mobiles are handheld devices. For example, a temperature on the back of a smartphone higher than 45^{o}C is a source of discomfort for users [130].

The strategy of adapting operating conditions at runtime to maintain temperature in a safe range is Dynamic Thermal Management (DTM), and it is usually implemented by the operating system. Thermal management has been well investigated for traditional systems [21], [62], but these do not account for user experience. Thermal management techniques can be reactive or proactive. Reactive techniques take decisions based on the current and past thermal state of the system. For this reason, reactive thermal management must set a conservative threshold value, thus leading to performance penalties. Proactive techniques, instead, leverage a thermal model to make predictions on the future impact of control decisions, to fully exploit the available thermal margin. It keeps the temperature as close as possible to the thresholds during intervals with heavy work-

loads, and improve overall performance. The main challenge associated with proactive thermal management is the identification of a thermal model for the target device, and its use within a control loop [41].

A compact thermal model can be identified on the target device by exploiting readings from integrated thermal sensors [22]. This is usually done by measuring temperature and power on the target platform and identifying an input-output system as a state-space model. It is possible to demonstrate that temperature can be represented with a linear time-invariant model, which is easy to store and to execute at runtime on a mobile device. Model identification of a compact thermal model is a practical and effective solution to implement with dynamic thermal management, and it is well-suited for mobile devices.

Similar to power management, it is important to deploy dynamic thermal management techniques on real devices to obtain feedback and determine user experience. Thermal management is also useful to mitigate the impact of transistors and interconnects degradation mechanisms. These have an exponential dependency on temperature and can lead circuits to early failure. Unfortunately, thermal management alone does not account explicitly for the impact of reliability mechanisms. This is fundamental to meet product's lifetime requirements.

1.4 Reliability Modeling and Management

Optimizing performance in mobiles increases the peak power consumption, causing the device temperature to raise quickly. This dramatically worsens the impact of transistors and interconnects reliability degradation mechanisms, which depend on temperature and voltage stress. Semiconductor devices are subject to degradation mechanisms which harm their performance over time and eventually cause them to fail. CMOS transistors are affected by Time Dependent Dielectric Breakdown (TDDB), Negative Bias Temperature Instability (NBTI) and Hot Carrier Injection (HCI). Metal interconnects are affected by Electromigration (EM) and Thermal Cycling (TC) [168]. With the scaling of CMOS, the impact of degradation worsens as the dimensions of transistors and interconnects shrink, leading them to early failure [65].

The International Technology Roadmap for Semiconductors (ITRS) identifies reliability issues due to aging as a primary concern for integrated circuit [75]. Uncertainties in reliability can lead to performance, cost and time-to-market penalties and can originate field failures that are costly to fix and damaging to reputation. In the past it was possible to mitigate reliability issues with higher design margins, trading reliability with power and timing without affecting performance much. With latest technologies, however, increasing the design margin further would severely jeopardize performance. This is the reason why in the last decade much effort was spent in developing innovative solutions to guarantee high reliability at the process, design, OS, software and application level [64].

Solutions for reliability at the process and design level are difficult to realize in practice. Modifying the manufacturing process may not be affordable, while making chip design more robust can significantly increase the design cost and decrease profit margins [58]. Techniques at the compiler level, instead, introduce memory and performance overhead. They rely on the application/compiler programmers ability to adapt the software and enable reliability-aware execution [136], [123]. At the application level, proposed techniques guarantee the correctness of execution with no hardware overhead [52]. However, they are often application-specific, and thus they lack portability.

All the degradation mechanisms depend on voltage and temperature stress and can be described by a reliability function which at any point in time represents the probability that the device does not fail [153]. Given this, the degradation of devices can be changed at runtime by managing operating conditions that influence voltage and temperature. This requires having model-based estimation of reliability degradation over time [176]. Such strategy is referred to as Dynamic Reliability Management (DRM).

The most promising abstraction layer on which to implement DRM strategies is the operating system, in close cooperation with hardware. Modern operating systems, in fact, have power management capabilities which handle factors that impact reliability, power and temperature, with negligible overhead at runtime. A dynamic reliability management policy developed for the operating system is relatively easy to implement, portable to other devices, requires no hardware overhead and is application-independent.

However, an effective solution for reliability management requires detailed information about the status of the platform under control (through sensors), and also information about the quality requirements of the running application. Thus, a cross-layer approach is a promising solution for reliability.

Reliability can be monitored at runtime thorugh the use of degradation sensors [80], [82], [147]. Unfortunately, such sensors today are not available on commercial devices. Therefore, implementating DRM on real devices requires the online emulation of reliability degradation. This leverages readings from built-it voltage and temperature sensors as inputs for the reliability model and computes the current value. Such measure can be then used in a control loop to adapt the lifetime degradation of the target device.

The scaling of CMOS and the consequent increase in degradation rate variability make dynamic reliability management even more important. Because of this, the distribution of circuit lifetimes becomes larger. DRM then is required to balance the degradation of devices over time and avoid early failures.

1.5 Variability Modeling and Management

The problems of power consumption, temperature stress and reliability degradation will only worsen with CMOS scaling and the consequent increase in variability [58]. Variability in integrated circuits refers to the deviation of the actual value of a design parameter from its nominal value, such as transistor channel length and threshold voltage. Sources of variation are static and dynamic. Static variations are due to the intrinsic inaccuracy of the fabrication process. Dynamic sources, such as ambient temperature and system workload, change their impact over time. In multicore processors, transistor-level static variability determines power and performance distributions across cores of the same model [28]. Consequently, computing units have a power consumption, operating voltage/frequency and degradation rate which are different from nominal right after fabrication. The impact of variations becomes more dramatic as the scaling of CMOS technology progresses and dimensions of transistors shrink. Counteracting with higher design margins is extremely costly [75].

Simulators have been developed to help designers make chips more robust to

variations [141], [113]. These tools can be integrated into the design flow to estimate the resulting impact of variability. However, they cannot account for dynamic variations and cannot capture the real workload behavior. Moreover, they require a highly detailed architectural description, which may not be always available. Given that dynamic variations play an important role, a good strategy is to leverage sensors to expose variability to higher levels of the software stack (e.g. the operating system) to manage it at runtime [58]. This strategy is referred to as Dynamic Variability Management (DVM). DVM can leverage common hardware sensors such as performance counters and temperature sensors, or more sophisticated ones, such as degradation sensors and frequency sensors (which are devices capable of monitoring path delays) [90]. The latter, however, are only available on prototypes and research platforms.

Mobiles are characterized by interactive workloads, which are hard to simulate and predict, as they strongly depend on user needs [49]. Moreover, the goal of mobiles is to provide quality user experience, rather than high performance. It is crucial to be able to estimate the impact of variability on user experience, which is not possible with simulators. For the prototyping and implementation of dynamic variability management techniques it is necessary to emulate the effect of variability online, to enable what-if analysis on real devices.

1.6 Thesis Contributions

Mobiles need to meet the varied performance requirements of different applications to achieve quality user experience. This increases power consumption, which drains the battery quickly and raises the overall system temperature. High temperatures worsen the impact of reliability degradation mechanisms and can cause circuits to fail prematurely. In addition, power consumption, operating conditions and degradation rates are different from their nominal values due to variability. Existing solutions for mobiles consider these aspects separately and offer limited solutions for reliability and variability management. Developing and testing prototypes of reliability and variability management require online emulation of these effect on real devices and with real applications at long enough time scales. Our work solves this problem.

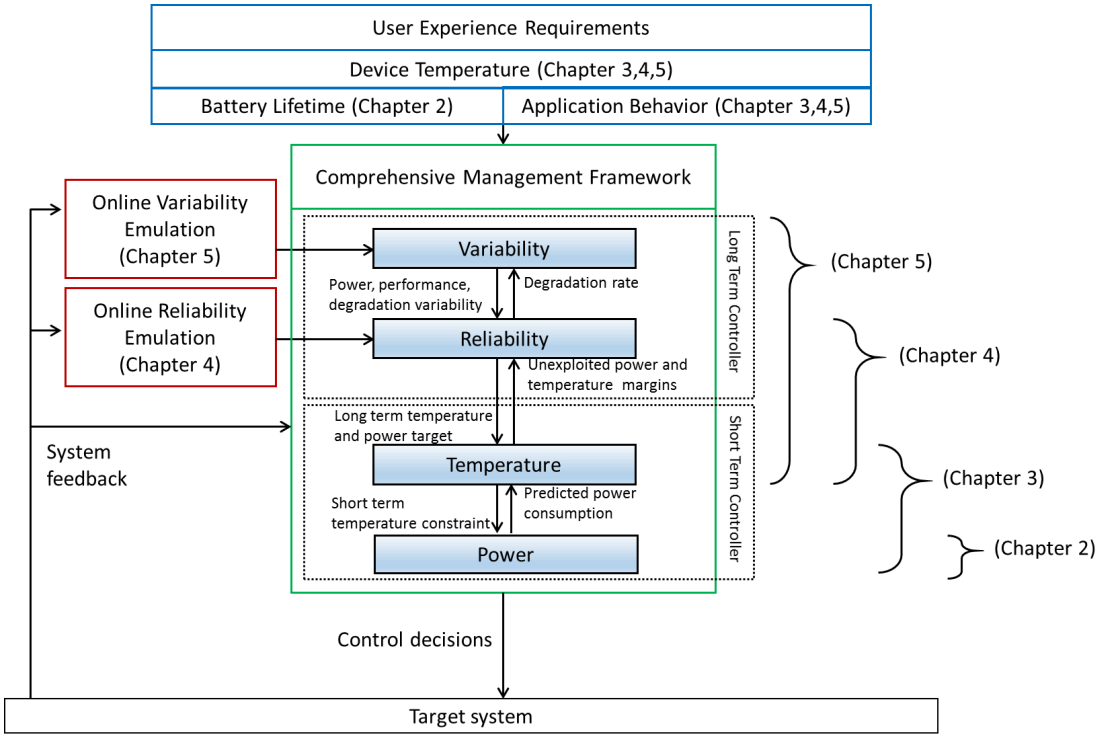

Figure 1.1: Diagram of the proposed framework.

The goal of this dissertation is to present the design and real implementation of a unified framework for the comprehensive dynamic management of power, temperature, reliability and variability in mobile systems, subject to user experience requirements. The reference framework for this dissertation is presented in Figure 1.1. The framework receives the **user experience requirements** as input, which are in terms of desired application behavior, battery lifetime and device temperature. Then, it makes control decisions based on system feedback, which can be part of power, thermal, reliability and variability management. Controls include dynamic voltage and frequency scaling of processors, task scheduling, power gating, adaptation of screen brightness. The system feedback is obtained by sensors and counters, such as utilization metrics, hardware counters, temperature sensors readings.

We propose a multi-rate **comprehensive management framework** that has two subcontrollers, the **long term** and the **short term controller**. This design choice is motivated by the fact that the time scale of interest for reliability changes and variability effects are in the order of weeks and months, which is very different from milliseconds

and seconds at which power and temperature change.

The **short term controller** activates at a fine-grain rate and aims to meet the fast-changing application-level performance requirements dictated by user experience, while optimizing for temperature and power. In doing this, it determines the solution to meet the average targets provided by the long term controller, which account for variability and reliability. These are used to update the thermal constraints at a fine-grain rate, and the solution is adjusted by predicting power and temperature for the next intervals. For this, we first propose a novel power characterization strategy for mobile devices called application-dependent power states (AP-states). Based on that, we formulate a management problem to improve performance under battery lifetime constraints, and we implement the management framework on a real Android device. We call our framework BLAST: Battery Lifetime-constrained Adaptation with Selected Target. The goal of the framework is to maximize performance while ensuring the device battery lasts at least for a user required lifetime. The implementation does modify the OS and can be ported and installed on any Android device. We experimentally verify that our strategy can still meets user experience requirements with a selected target battery lifetime extension of at least 25%. This is presented in Chapter 2.

We also propose a joint power and thermal management solution, which takes a proactive approach in reducing energy consumption while providing expected user-experience. The proposed technique modulates the operating conditions based on users application preferences and exploits the "change blindness" effect to reduce display power consumption. Another important aspect of our implementation is that it does not require any restructuring of the underlying operating system. A novel thermal model of the entire smartphone is derived using model identification techniques, based on the device's operating conditions. This has the the purpose of monitoring and controlling the operating conditions to keep the device temperatures within safe operating ranges. Our ready-to-use management technique has been implemented on Google Nexus 5 and has been demonstrated to achieve a 46% application-specific savings on power consumption and up to 35% savings in power consumption at the device level. The mean temperature estimation error is 1.17^{o}C. This is presented in Chapter 3.

The **long term controller** activates at a coarse-grain rate and focuses on meeting

the target **reliability** in the long term, based on information on **variability** in power consumption, performance and degradation rate. Variability information is updated based on the reliability degradation. The long term controller also computes target average temperature and power values that are used as constraints by the short term controller. The target on reliability is met if the average temperature and power are lower than the target at the coarse-grain level. In this case, the short term controller returns the unexploited margins which are used to increase the average targets for the next coarse-grain control interval. We formulate dynamic reliability management as an optimization problem that accounts for reliability, temperature and performance. We optimize for multicores using convex optimization, and show that it is not feasible to implement on real systems. For this reason, we propose Workload-Aware Reliability Management (WARM), a fast DRM technique adapting to diverse workload requirements to trade reliability and user experience. WARM is implemented and tested on a real Android device. It leverages RelDroid, an infrastructure for the online emulation of reliability degradation. RelDroid enables the design of workload-aware dynamic reliability management on real mobile devices with accurate reliability models. Our framework captures the effect of variable workload and environmental conditions and allows to emulate longer degradation in a short time scale. We implement the framework on a real Android device and exploit it to enable workload-aware dynamic reliability management. WARM approximates the solution of the convex solver within 18% in the worst case, while executing more than 40x faster. It integrates a Thermal Controller that allocates tasks to meet thermal constraints. This is required since degradation strongly depends on temperature. WARM task allocation achieves up to 1 year lifetime improvement for a multicore platform. It can achieve up to 100% of performance improvement on cluster architectures, such as big.LITTLE, while still guaranteeing the reliability target are met. This is presented in Chapter 4.

Due to the scaling of CMOS, processors with the same normal characteristics actually have variability in power, performance and reliability degradation, which should also be taken into account in the runtime management of hardware resources. For this, we present VarDroid, a low-overhead tool to emulate power and performance variability on real platforms, running on top of the Android operating system. VarDroid enables

us to analyze the effect of variability in power and performance while capturing the complex interactions characteristic of mobile workloads, thus relating to users quality of experience. We present use cases to show the utility of VarDroid to test applications, device and OS robustness under the effects of variability. Our results show that a variability-agnostic OS can incur in a performance penalty of up to 60% and a power penalty of up to 20%. Then we use VarDroid to develop a novel dynamic variability management technique, which leverages a variability-aware OS algorithm to assign the workload to the cores and set the power/performance tradeoffs to meet the mobile processors lifetime constraints while adjusting to variability and improving the overall user experience. The proposed DVM solution uses sensors to monitor the variable operating conditions and the degradation rate. We implement our algorithm in Android OS on a mobile phone and show that it achieves up to 160% performance improvement over the state-of-the-art while meeting the lifetime constraints. This is presented in Chapter 5. Finally, Chapter 6 concludes this dissertation by summarizing the main results and suggesting important areas of future work.

Chapter 2

Power Management with a Battery Lifetime Constraint

Mobile devices today contain many power hungry subsystems and execute different applications. Power management implemented in today's system is not aware of the desired battery lifetime and has no visibility into which applications are executing. In this chapter, we propose a novel power characterization strategy for mobile devices called *application-dependent power states* (AP-states). Based on that, we formulate a management problem to improve performance under battery lifetime constraints, and we implement the management framework on a real Android device. We call our framework **BLAST**: Battery Lifetime-constrained Adaptation with Selected Target. The goal of such framework is to maximize performance subject to a required battery lifetime. The implementation does not require OS modifications and can be ported and installed on any Android device. We experimentally verify that our strategy can still meets user experience requirements with a selected target battery lifetime extension of at least 25%.

2.1 Introduction

Mobile devices such as smartphones and tablets contain a variety of power hungry subsystems (CPU, GPU, camera, display, antennas, etc.) and execute applications with different requirements: from browsing, to multimedia, to gaming and many more. Power consumption heavily depends on the application running in foreground (e.g. the

one showing on the display), as it mostly determines the usage of different parts of the system. Also, it is the application which attracts users attention, thus influencing user experience the most. Such intense activity contributes at making the battery lifetime as short as few hours for most devices [145], [15]. Therefore, power management to trade battery lifetime and user experience is a primary requirement for mobiles.

In the last decade, the target of mobile designers and developers has shifted from high performance to high user experience. The concept of user experience depends on a number of variables, from device specifications and performance to user personal profile and level of attention. However, we can define it as the scenario in which device behavior meets user expectations. Therefore, in the case of mobiles we can identify two main factors determining user-experience: (i) application behavior and (ii) battery lifetime. The first refers to the case in which the user is satisfied with application execution (for example, a Youtube video that reproduces smoothly or a 3D game with high frames per second). The second indicates the case in which the achieved battery lifetime is as long as the one expected by the user. The two targets are contrasting, as there is a tradeoff between them: if power is optimized for providing a minimum required level of user experience, this could mean trading on battery lifetime, if the level of expectation is high (for example for a user playing 3D games). On the other hand, if the goal is to reach a predefined battery lifetime, this could penalize the behavior of some applications. This means that even if both are factors affecting user experience, either one of the two can be the constraint of power management, but not both at the same time. A comprehensive management solution requires the possibility of dynamically switching from one strategy to the other, depending on the users main concern at the time: application behavior or battery lifetime.

Recent publications mainly address the first problem [145], [99], [86], [108], [95]. These approaches all require some description of user-experience to adapt, which is provided either by the users configuration, or with user experience modeling. However, no unique model for user-experience depending on application behavior is widely accepted so far, they all suffer from inaccuracies due to the heterogeneity and high complexity of user profiles [49]. To the best of our knowledge, no work addresses the problem of maximizing performance while ensuring that a minimum battery lifetime for mobiles is

met. Such scenario is better explained by a motivational example in the next subsection.

The power management of todays mobile devices is implemented at the OS level and regards mainly CPU and GPU. These two are the most power consuming subsystems [33]. Display is also very power hungry, but it should be managed independently as it is critical to user-experience [108]. Therefore, we do not consider it in this chapter. Other subsystems either have no power management control (e.g. antennas) or have proprietary kernel code (e.g. modem, DSPs), which makes it difficult to modify and evaluate. For example, the Android operating system, which is based on the Linux kernel, has modules called *Frequency Governors* to implement the power management policy for the CPU and GPU [125]. The performance governor always sets maximum frequency, while the powersave governor always sets minimum frequency. Similarly, the conservative governor allows for low power consumption, at the cost of potential performance loss. Todays standard governor, the ondemand, scales frequency over time depending on CPU (or GPU) utilization. Such approach has two main limitations: (i) it is agnostic of which application is currently running on the device and (ii) it does not account for battery lifetime.

In this chapter, we present BLAST: Battery Lifetime-constrained Adaptation with Selected Target. BLAST is a novel power management framework for mobile devices, which dynamically adapts to different applications while ensuring a predefined (e.g. selected) battery lifetime. In this chapter we formulate an application and battery lifetime-aware power management problem for mobile. We propose the concepts of *Application-dependent Power state* (AP-state), *battery discharging profile* and *energy tank* to determine power management decisions. Based on this, we develop BLAST: a lightweight, ready-to-use, high-level and portable implementation on a real Android smartphone, which does not require OS modifications and thus can be easily extended to any mobile device. The proposed implementation is in the user space and it is compatible with any frequency governor in the kernel.

With a set of experiments conducted on real devices executing common Android applications we demonstrate the effectiveness of our strategy in guaranteeing the predefined battery lifetime and compare against device native power management. Also, we show that our strategy can still meets user experience requirements with a selected

target battery lifetime extension of at least 25%. This claim is demonstrated by testing the framework with real users. The average rating of users is within 5% for a battery lifetime improvement of 25%.

2.1.1 Motivating Example

Assume that two users X and Y are leaving work to get back home by train. They both usually use their smartphone on the way home, but while user X enjoys playing 3D videogames, user Y prefers to read emails or browse through news websites. The train takes 1 hour to bring them home, and during that period of time they absolutely want their smartphone to not run out of battery, no matter what the quality of application behavior is. Once home, they are both going to put the device into charge.

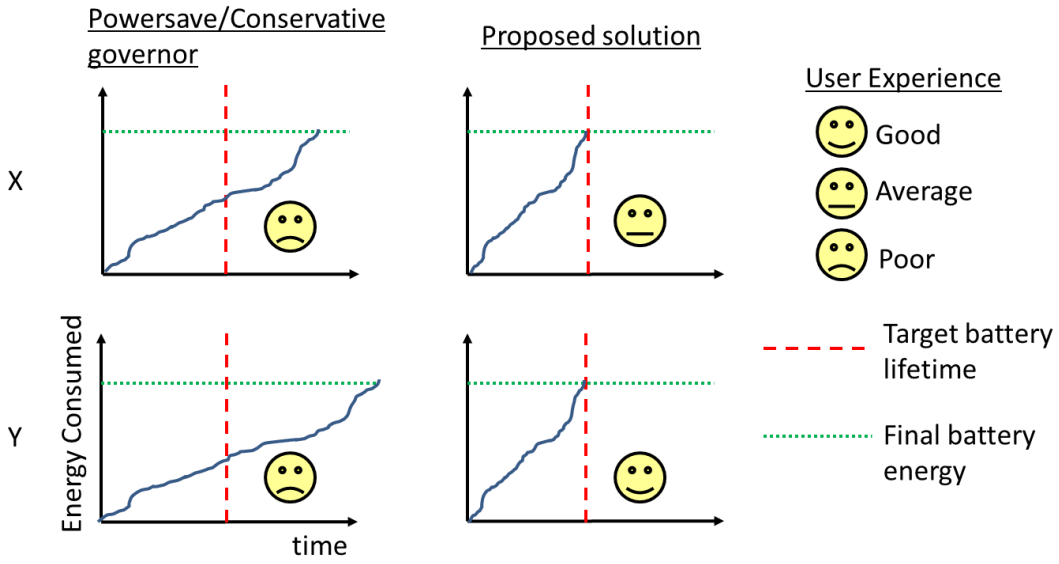

Figure 2.1: Illustration of motivating example.

One obvious solution would be to lower the operating conditions, for example by setting the powersave or the conservative governor. However, such approach presents three downsides. First, it requires the user to be aware of what a governor is and how it works, and to be able to install and use an interface application to change it. Second, the use of the powersave governor is likely to extend the lifetime of the battery way beyond 1 hour, depending on the initial state of charge. This is because it is not aware of battery

energy. As a result, the powersave governor may hurt application behavior more than what is required to meet the constraint on battery lifetime. Third, the powersave governor is not aware of which application is executing. However, in the scenario described, user Y (mail and news) is likely to consume less energy than user X (3D games). Therefore, the performance level required to meet the same target lifetime is different for the two users.

Our solution, on the contrary, only requires the user to set the desired minimum battery lifetime (1 hour in this example), that is, the selected target. Then, the framework automatically detects the battery state of charge and the executing applications, and regulates energy consumption thanks to the AP-states by adapting the maximum CPU and GPU frequency. The result is that both user X and Y will have a working smartphone for the next hour after leaving work. The presented example is better shown in the qualitative plots in Figure 2.1, which show the energy consumed over time for user X and Y respectively when using a powersave or conservative governor and when using our proposed solution. We also highlight the user experience achieved in the four cases (either good, average or poor). Finally, note that the target lifetime for the proposed solution should be selected in a defined range. This is better clarified by Figure 2.2. The lower bound is represented by the battery lifetime obtained with all cores active executing at maximum frequency (e.g. with performance governor), while the upper bound is given by a single core active executing at minimum frequency (e.g. with powersave governor).

2.2 Related Work

Power management is an extensively investigated area of research, from server systems, to desktops and laptops, to mobiles [23]. The characterizing aspect of mobile devices with respect to other systems is the reduced form factor, which limits battery size [33]. For this reason, researchers spent many efforts in the last decade in power analysis, modeling and management for mobiles. Publications [145] and [33] analyze phone power consumption and investigate the impact of different user activities and different applications. Work in [145] also demonstrates that CPU, GPU and screen are the most power consuming subsystems in modern smartphones. Yoon et al. propose

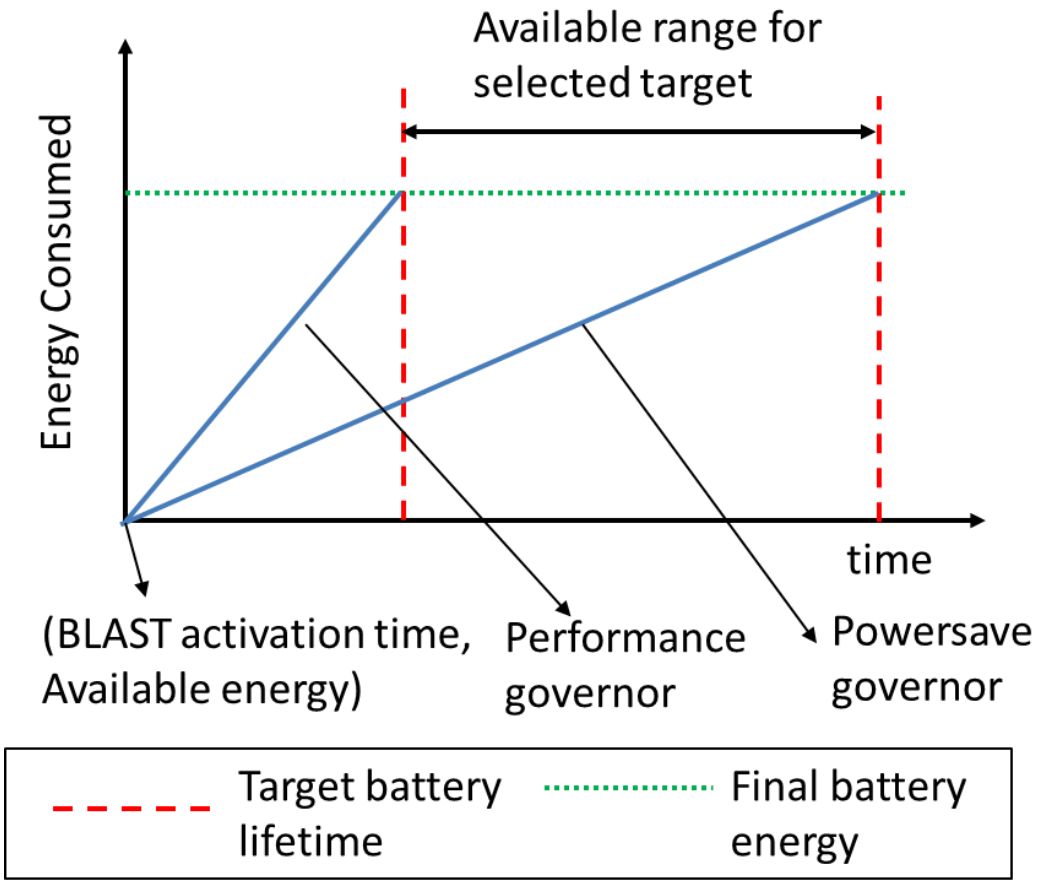

Figure 2.2: Range for selected battery lifetime target.

Appscope, a tool for Android energy metering, and characterize power consumption for different applications in reference [172]. Paper [162] proposed a framework to estimate power consumption of different applications from battery power traces. These publications highlight that energy consumption for mobiles is highly influenced by different applications and that CPU and GPU are crucial in determining battery lifetime.

For this reason, power models for mobile devices have been proposed recently. Reference [12] develops a power model based on user activity for an Android-based smartphone, using regression techniques. Work in [26], [120] estimates power consumption through adaptive modeling based on monitored performance activity, and integrate it in the *MPower* app, which provides the user suggestions to improve power efficiency. MPower collects measures on the target device and transmit them to a server for power estimation. Performing the estimation online would result in performance overhead.

In general power models may not be practical for runtime power management, due to computation overhead.

Recent work on power management for mobiles focuses on user-experience determined by application behavior. For doing this, some techniques allow the user to configure personal preferences and application priority levels [122], [103]. Other techniques, instead, are based on user-experience models. The strategy in [86] increases CPU frequency in response to user interaction, to minimize perceived delay. Publication [145] presents a model for user typical activity session duration, and use it to compare various power management strategies. Work in [166] proposes a scheduling algorithm for energy-based fair queuing, aiming at optimizing activity and idle periods for user comfort. Such techniques achieve better energy efficiency only if users requirements are not too strong, as they target application behavior. However, they do not give guarantees on battery lifetime. Moreover, models for user experience may be inaccurate, due to diversity of user profiles [49]. Our technique does not require user experience models, as it targets battery lifetime. Moreover, it only requires the user to configure the desired battery lifetime. Also, the implementation of such techniques requires modifying the operating system, which may affect portability across devices. Li et al. propose an intelligent and self-adaptive scheme for mobile power management, called SmartCap [96]. The objective of SmartCap is to automatically configure the CPU frequency subject to user experience requirements. The proposed approach is shown to significantly outperform the standard ondemand governor. Our work is fundamentally different as we try to maximize performance subject to a battery lifetime constraint. SmartCap, on the other hand, aims at minimizing power consumption while meeting a user experience (e.g. performance) constraint. As discussed in the introduction, these two problems are complementary to each other. Also, Smartcap focuses on CPU solely, while we include also GPU frequency control in our implementation.

Other techniques for mobile power management are developed for specific applications. Reference [131] presents a joint Dynamic Voltage and Frequency Scaling (DVFS) for CPU and GPU targeting 3D games. Work in [94] makes Youtube more energy efficient by intelligently scheduling download activities. Being specific to certain applications, such techniques cannot be extended to full phone power management.

Instead, our technique is developed to be compatible with any application.

A particular case is made for display power management. As shown in reference [145], display brightness plays a fundamental role in user experience; therefore it should be managed independently from CPU, GPU and other subsystems. For example, the authors of paper [37] develop a technique to adapt voltage scaling of OLED displays to video streaming while accounting for user satisfaction. In this work we do not include display management, but we show how it can be integrated. Our work formulates a management problem for mobiles considering battery lifetime as a constraint rather than as an objective function, and implements a portable and lightweight framework for managing power consumption on Android devices executing real applications.

2.3 Manager Formulation and Implementation

In this section, we first show the assumptions of our work and key observations. Based on that, we describe the concepts of *AP-states*, *battery discharging profile* and *energy tank*, and the management problem formulation. Finally, we describe the solution strategy and the framework implementation.

The target platform of this work is a battery-powered mobile handheld device equipped with DVFS-enabled CPU and GPU, controllable from the userspace. This is common in modern devices, for which the operating system exposes control capabilities at the *sysfs* interface, like setting the maximum frequency. Both CPU and GPU have predefined voltage/frequency operating points. The battery power consumption and charge level are sampled from the sysfs interface as well, without the need of external equipment. In this work we use the ondemand governor, except when clearly stated. However, note that the proposed solution is implemented in the userspace thus it is compatible with any frequency governor. This is better shown in the results section.

The key for formulating and solving a management problem constrained by battery lifetime is having a reference model for battery power consumption. Battery power at a generic time t can be expressed as in Equation (2.1).

$$P_{batt}(t) = \sum_i P_i(t) \tag{2.1}$$

Here P_i is the power consumption of the i^{th} subsystem. Unfortunately, commodity mobile devices usually do not have per-subsystem power information available at runtime. Moreover, as mentioned in the introduction, not all subsystems can be controlled at runtime to adjust power consumption. Therefore we rewrite battery power as in Equation (2.2).

$$P_{batt}(t) = P_c(t) + P_{nc}(t) \tag{2.2}$$

Where P_c represents the power contribution that can be controlled at runtime and P_nc is the contribution which cannot be controlled. In this work, we assume P_c to be a function of CPU and GPU frequency, therefore $P_c = P_c(t, f_{CPU}, f_{GPU})$. As for the non-controllable contribution P_{nc}, this is heavily determined by the behavior of the application running in foreground, which determines CPU and GPU load, number of active CPUs, antennas usage, task scheduling, and other factors in which we either have no control or that are already managed by other entities in the device. For example, in Android the number of active CPUs is controlled by the userspace daemon called *mpdecision*.

Deriving a model for $P_c(t)$ at each time t leads to inaccurate estimations for two reasons: (i) only battery power consumption can be monitored from userspace in commodity devices, therefore it is hard to isolate the two components P_c and P_{nc} from observations, and (ii) the sampling rate at the userspace should be at least 1 second to avoid excessive overhead. What matters is not instantaneous battery power consumption, but rather its average over time, or consumed energy, shown in Equation (2.3), and approximated in Equation (2.4). In this Equation, P_{avg} is the average battery power consumed over the time period T.

$$E_{batt} = \int_{t=t_0}^{t=t_0+T} P_{batt}(T)dt \tag{2.3}$$

$$E_{batt} = P_{avg}T \tag{2.4}$$

The fundamental observation we make is that once the operating conditions are fixed (in our case f_{CPU} and f_{GPU}), then the average power consumption tends to stable

values over time for different applications. To motivate this fact, we show a simple measurement of battery energy in Figure 2.3 and 2.4. In this experiment, we measure the battery power consumption on a Nexus 5 smartphone while executing Chrome (browsing, scrolling and zooming) and Angrybirds for 150 seconds with CPU frequency fixed at 1.5Ghz and GPU at 390Mhz. Then we calculate the average power consumption on time windows of different durations. In Figure 2.3 we report the average power consumption over a time window of different duration. We notice that the average is almost the same for a given application, but changes for different apps. Referring to Equation (2.2), in this case the difference between the two applications is determined by P_{nc}, as frequency is constant in both cases. In Figure 2.4 we measure the energy consumed over time. For doing this we sample battery power consumption and battery capacity and relate that to the total battery energy (derived from datasheets). We observed that the energy consumed is almost linear over time, with different slopes for different applications. This is an example of the simple fact that playing 3D games for one hour drains battery more than browsing.

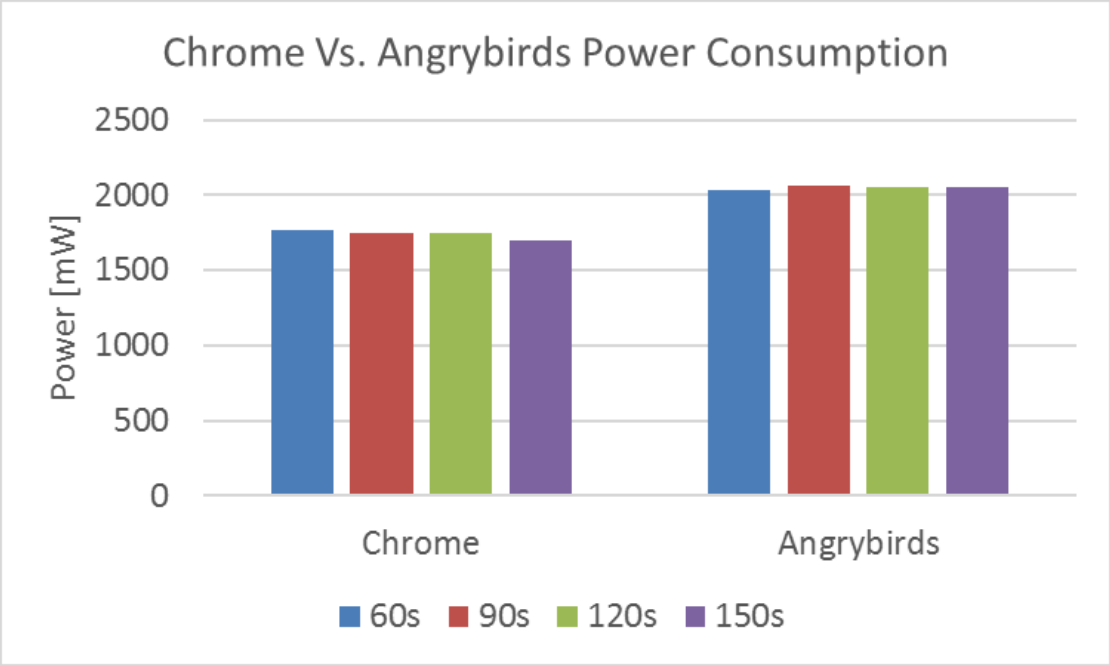

Figure 2.3: Power consumption of Chrome and Angrybirds on Nexus 5.

Based on these two observations, we associate a value of average power con-

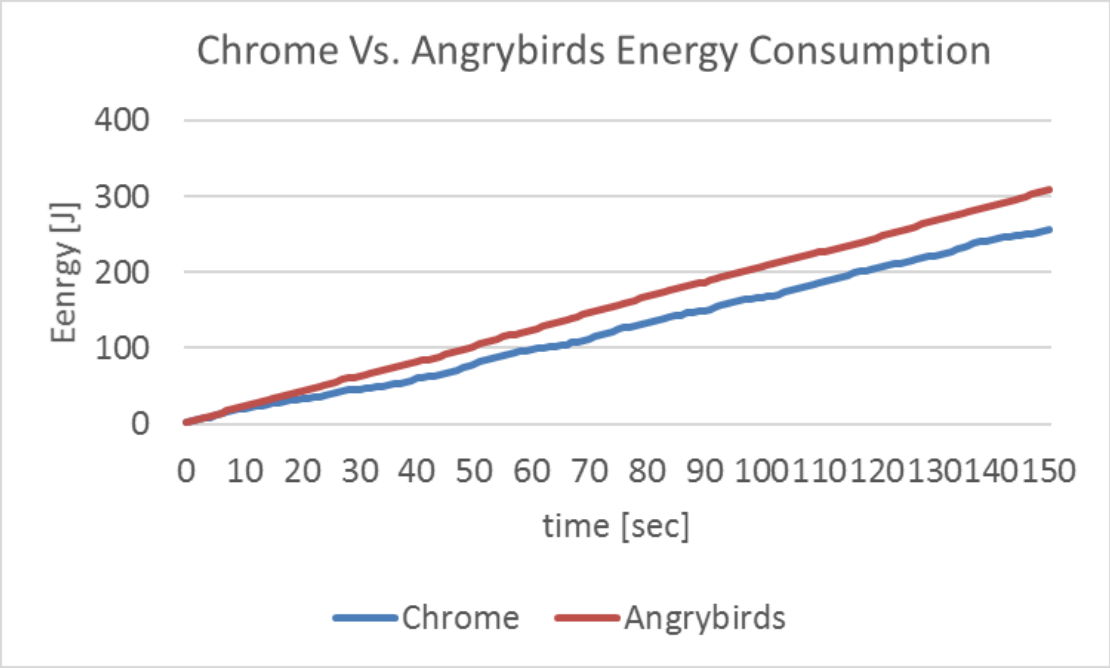

Figure 2.4: Energy consumption of Chrome and Angrybirds on Nexus 5.

sumption to each pair $f = (f_{CPU}, f_{GPU})$, independently for each application. Then, we create a table for each app such as the one shown in Table 2.1. In this table, $P_{APP_i}(f_j)$ is called AP-state, and it is the average battery power consumed by application i in state j and T_{ij} is the total time spent by the device executing application i in state j. The values of P_{APP} and T are updated at runtime based on the monitored operating conditions, battery power and application executing in foreground. AP-states are stored in memory and there is at most one list of AP-states for each application. Given this, the energy battery consumption can be rewritten as in Equation (2.5).

$$E_{batt} = \sum_i \sum_j P_{APP_i}(f_j) T_{ij} \tag{2.5}$$

In other words, an application running on the device contributes an average value to the total energy consumption. This depends on the execution frequency and is weighted with the time spent executing it. At this point, Equation (2.5) allows us to formulate the management problem.

The management problem requires the specification of a target battery lifetime t_{charge} and of an energy budget E_{budget}. Note that the energy budget could be simply set

equal to the remaining battery charge. The final goal of management is to not consume more than energy E_{budget} before time t_{charge}. The manager activates at a constant time rate and we indicate with t_k the current time step. The time of manager activation is t_0 and the initial battery energy is E_{start}. The problem can be formulated then as in Equation (2.6).

$$\max(f_{k+1}) \text{ s.t. } E_{batt_{k+1}} < E_{target_{k+1}} \tag{2.6}$$

Where f_{k+1} is the frequency selected for the next time interval and E_{target} is the constraint on battery energy consumption. The value of E_{target} is calculated at each time instant based on the concept of battery discharging profile. This is a function of energy over time, starting from point (t_0,E_{start}) and ending at point (t_{charge},E_{budget}). For practical reasons, in this work we assume the discharging profile to be linear in time, therefore $E_{prof_k} = mt_k + E_{start}$. This is motivated by the result shown in Figure 2.4, for which battery discharging can be well approximated by a linear function. Note that the concept is general and different discharging profile (e.g. piece-wise linear, quadratic, etc.) may lead to a different management behavior.

If the device consumes less power than what is allocated for using at a certain time instant t_k, the energy left over is added to the energy tank E_{tank}. On the other hand, if it consumes more power, the excess energy is subtracted from E_{tank}. Therefore, in the end the target energy is given by Equation (2.7).

$$E_{target_{k+1}} = E_{prof_{k+1}} + E_{tank_{k+1}} \tag{2.7}$$

To solve this management problem we leverage a heuristic. Depending on which application i is currently running, the heuristic selects the highest frequency f_N and checks whether the condition in Equation (2.6) is met for $P_{APP_i}(f_N)$. If not, it selects f_{N-1} and checks again. If no valid frequency is found for the current time instant, then the manager selects f_0. The heuristic is feasible given the limited number of CPU and GPU predefined operating frequencies.

Table 2.1: AP-states

Frequency list	Power	Time count
f_0	$P_{APP_i}(f_0)$	$T_{i,0}$
f_1	$P_{APP_i}(f_1)$	$T_{i,1}$
...	...	...
f_{N-1}	$P_{APP_i}(f_{N-1})$	$T_{i,N-1}$

2.3.1 Framework Implementation

We implemented the management framework on a Nexus 5 smartphone. This device has a Qualcomm Snapdragon 800 chipset, with a quad-core Krait 400 CPU. Its maximum battery capacity is 31,257 Joules. The four cores have a frequency range from 300Mhz to 2.26Ghz in 14 fixed operating points, with $f_{CPU_{13}}$ being the maximum frequency. It also has an Adreno 330 GPU with frequency range from 200Mhz to 450Mhz, in 4 fixed operating points, being f_{GPU_3} the maximum. For simplicity, the CPU and GPU frequency pairs adopted for our AP-states are associated with the following rule: $(f_{CPU_{13-10}}, f_{GPU_3})$, $(f_{CPU_{9-6}}, f_{GPU_2})$, $(f_{CPU_{5-3}}, f_{GPU_1})$, $(f_{CPU_{3-0}}, f_{GPU_0})$. Figure 2.5 presents the block diagram of our implementation.

The **App Monitor** is an independent program, which periodically checks the executing processes and writes in the file *Foreground App* the name of the application currently executing in foreground.

The **AP-states Monitor** periodically samples the operating conditions of the multicore platform (CPU and GPU frequency) and the battery power with a rate of 1 second. Based on this and on the current foreground application, it updates the values of AP-states and writes it back to the file *AP-states*. If APP_j is executing, the current sampled power is P_{curr} and the current operating conditions are equal to f_k, then the k^{th} AP-state of APP_j is updated as in Equation (2.8). Also, the value of T_{jk} is increased by 1.

$$P_{APP_j}(f_k) = P_{APP_j}(f_k)\frac{T_{jk}}{T_{jk}+1} + \frac{P_{curr}}{T_{jk}+1} \tag{2.8}$$

The **Power Controller** loads the *Target Battery Lifetime* and the *Initial Energy Budget* configured by the user when starting execution. Then, it periodically applies op-

erating conditions (CPU and GPU frequency), by solving the problem shown in Equation (2.6) based on the values of energy tank, available energy (left from E_{start}) and discharging profile. The activation rate of the power controller is also 1 second.

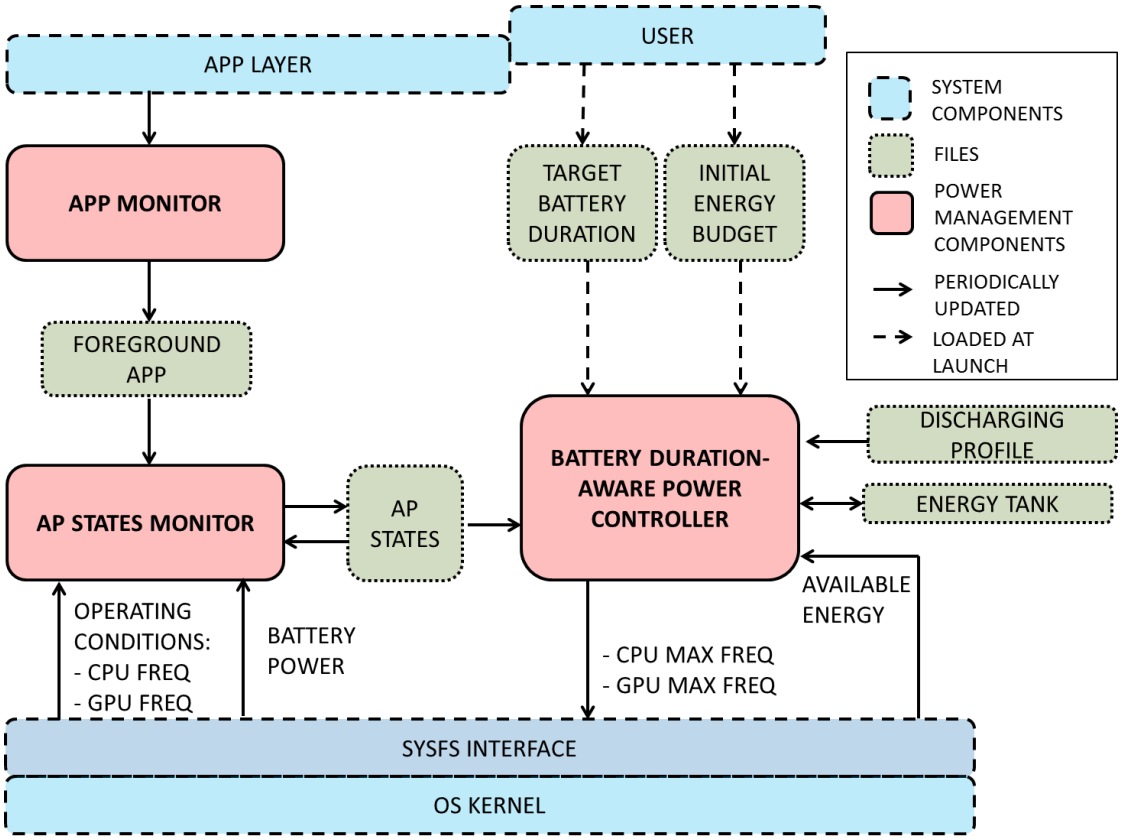

Figure 2.5: Block diagram of the implemented framework

Since the framework is implemented in the userspace, the minimum activation rate to avoid overhead is 1 second. However, the activation rate of frequency governors in the kernel space is much higher (20ms is the standard for the ondemand governor). Then, we must guarantee that our Power Controller is compatible with any governor. Therefore the output of the Power Controller is not a fixed frequency value, but it is a maximum operating frequency for all the four cores. This can be set by writing values to the appropriate sysfs file. Consequently, we decided to configure the AP-states Monitor so that it updates AP-states based on the current maximum value of frequency among the 4 cores. This choice guarantees that every time a certain maximum frequency configuration f_j is selected, the average contribution to the total power consumption will

not be higher than $P_{APP_i}(f_j)$, which is exactly what we require to meet a certain t_{charge}.

The two programs that compose this implementation, i.e. the App monitor and the AP-states Monitor/Power Controller, are written in C and cross compiled with Android NDK tools to run on ARM-based platforms. To run the framework, it is sufficient to load the binaries of the two programs in the device and execute them. No modification to the operating system is required.

2.4 Results

In this section, we describe the experimental results obtained by measurements on the target platforms. To perform comparisons on real applications, we wrote a program that allows to record and replay display touch event traces. In this way, comparisons are performed on the same trace of events.

For the first experiment, we recorded two traces on the Nexus 5 from two popular Android applications: Chrome and Angrybirds. We chose to show results for these two applications because they are representative corner cases of typical mobile usage: browsing and gaming. In the first, we are browsing through popular webpages. In the second we are playing the game. The first two plots of Figure 2.6 report the discharging curves for the two traces when no control is active. This allows us to fix an initial reference t_{charge} equal to 400 seconds and energy budgets E_{budget} of respectively 750 Joules and 800 Joules for Chrome and Angrybirds. Considering this we show how to configure the control for battery stretch, that is, for a longer t_{charge}. The second two plots of Figure 2.6 show three cases of battery stretch for the applications, respectively +12.5% (450s), +25% (500s) and +37.5% (550s). Reported values are the real energy consumed over time (bold green line) and the energy discharging profile E_{prof} (black dotted line). For practical reasons, the x-axis only reports the final section of the curves. We can notice that in all three cases, the total final consumed energy is less or equal than the predefined target, therefore the power management goal is met. The only exception is the case of +37.5% battery stretch for Chrome. In this case the final target exceeded. This means that t_{charge} for this case was set too long and the controller cannot meet it even with constant minimum frequency set. Moreover, a subjective evaluation suggested

that while in the case of +25% stretch the application behavior is still satisfactory, in the +37.5% stretch both applications become blurry.

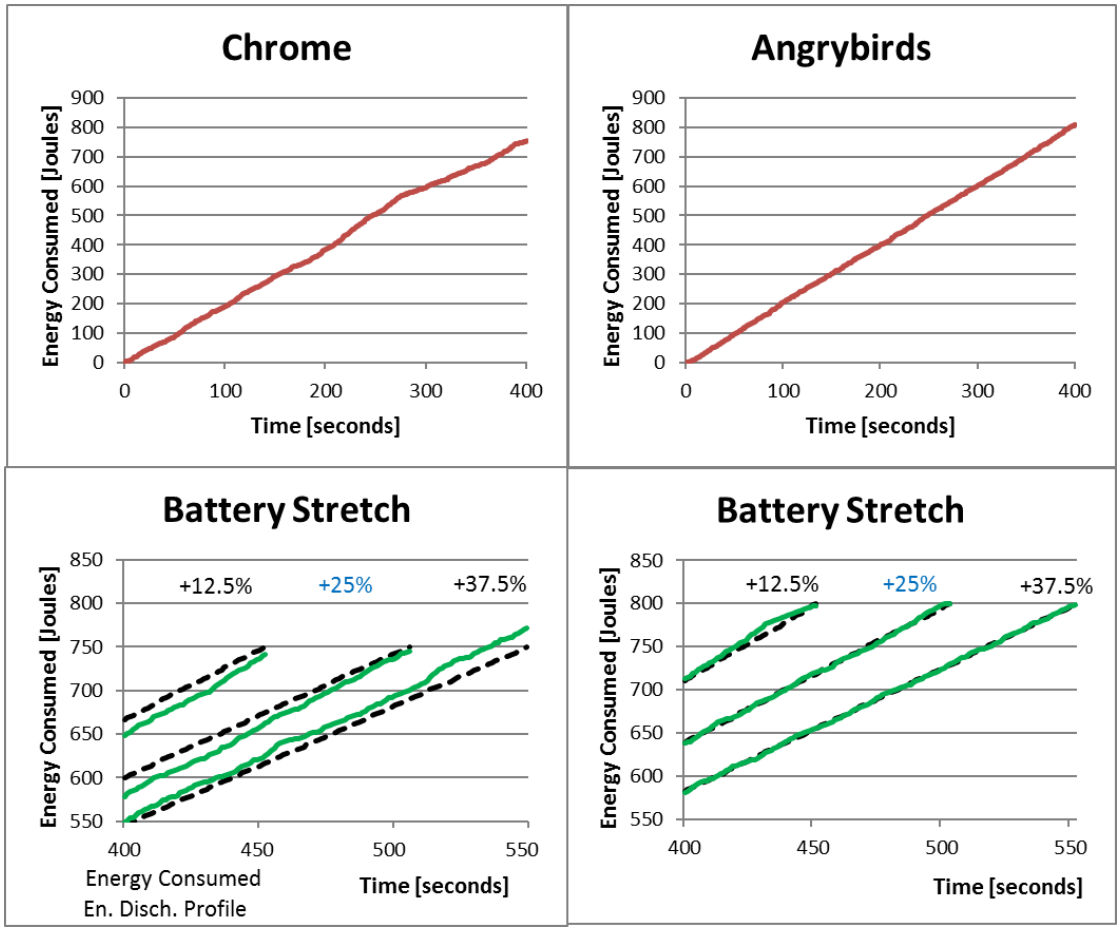

Figure 2.6: Standard energy consumption and battery stretch

We also conducted an experiment in which we asked 15 students from the university campus to check application behavior under battery stretch. To illustrate how portable our framework is, this study has been conducted instead on a Qualcomm APQ8064 development tablet. We asked people to play the popular game Temple Run and to browse on popular websites (CNN and BBC) for one minute under 0%, 25% and 50% battery stretch conditions respectively. In order to avoid influencing their judgment, users were not informed about the nature of the control and the goal of the experiment. In addition, the battery stretch conditions are applied in a random order. At the end, users were asked to rate their experience with a number from 1 (bad) to 10 (good). Fig-

ure 2.7 reports the average value of scores for the two applications. What we observe is that there is very low or no negative difference when the 25% stretch is applied, while there is a more negative effect with 50% stretch. Our conclusion is that the control can still meet a satisfactory experience for the user when the selected target battery lifetime is extended by 25%.

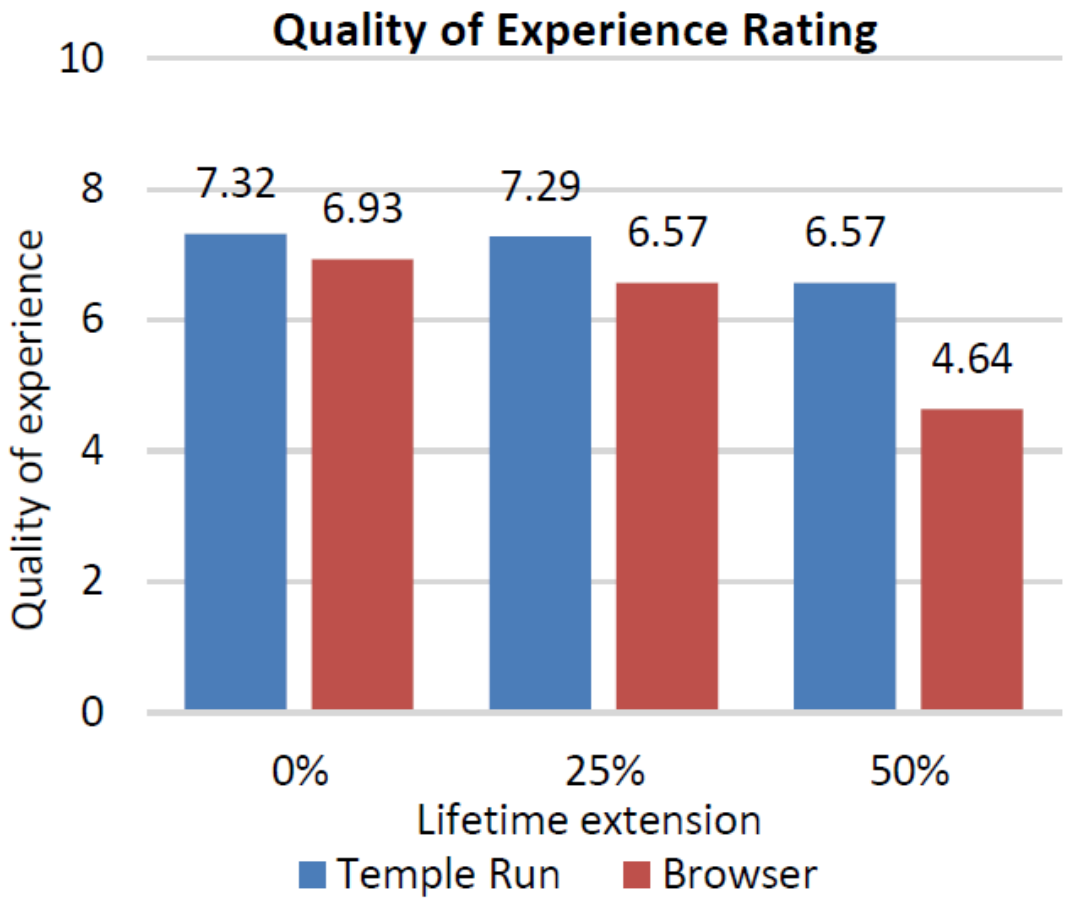

Figure 2.7: Quality of experience rating

To further prove this point we refer to the fact that there is a well known relation between Frames per Second (FPS) and user experience in the case of gaming applications [131]. For this reason, we observe the behavior of FPS during the execution of Temple Run in the three cases of 0%, 25% and 50% target extension in our Qualcomm tablet. FPS can be monitored in Qualcomm devices thanks to a built-in feature that enables the system to write values to logcat. Figure 2.8 reports the FPS traces for 30

seconds of execution. In the case of 0% extension, the FPS is close to 60, which is the maximum value allowed by Android, and represent the highest quality of user experience. In case of 25% extension, the FPS is around 50, which represent a lower, but still acceptable user experience (as confirmed by results in Figure 2.7). Finally, for a 50% extension, the FPS drops even below 40, which starts to represent a source of discomfort for the user.

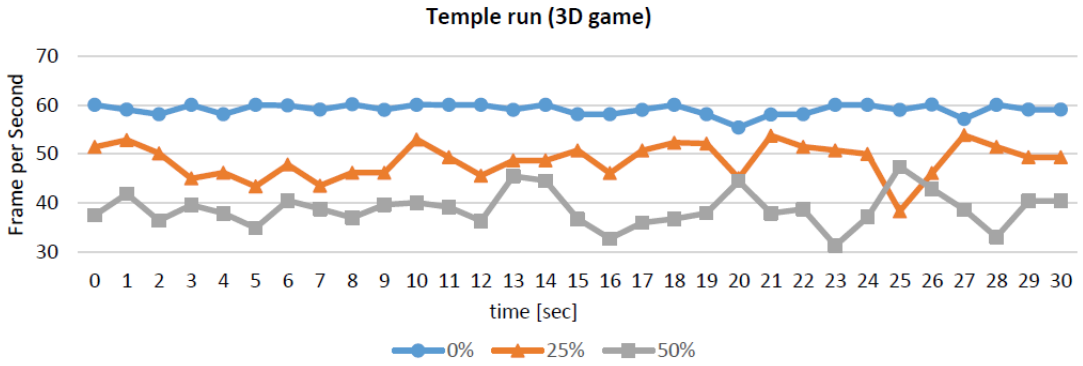

Figure 2.8: Frame per second (FPS) traces with different target lifetimes

To show more into details how the management policy works, we report in Figure 8 the Nexus 5 execution frequency of the 4 CPU cores and the maximum frequency set by the controller for the case of +25% battery stretch in 200 seconds of the Angrybirds trace. Our controller does not fix the frequency, but only limits the range of the underlying frequency governor (black dotted line).

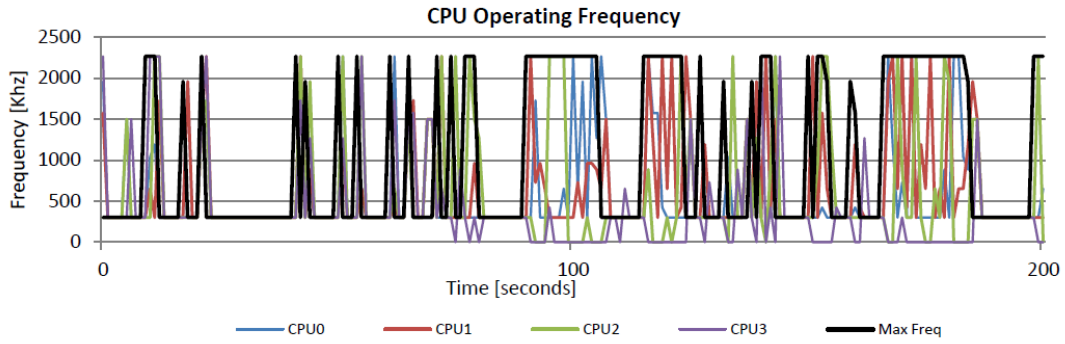

Figure 2.9: CPU operating frequency for Angrybirds +25% battery stretch

To show the tradeoff between t_{charge} and performance, we execute a set of ex-

periments on popular Android benchmarks on the Nexus 5: Antutu, Geekbench3, GFX and Vellamo Browser. Antutu is a suite of benchmarks comprising CPU, 2D and 3D evaluation. Geekbench3 is a CPU evaluation tool which tests both single and multicore performance (Gb3_single and Gb3_multi). GFX is a suite of graphics benchmarks from which we selected t-rex (GFX_trex). Finally, Vellamo Browser is a benchmark testing device performance while using Chrome. All these benchmarks provide a final score which is an indicator of performance quality. Figure 2.10 shows the scores obtained for the presented benchmarks with varying values of t_{charge} (respectively at 15%, 25% and 50% extension), normalized over the maximum score (which is obtained when the control is not active).

From the results, we can notice that as battery lifetime is extended, the score of the benchmark (i.e. the device performance) decreases. In addition, we can notice that different applications show different sensitivities, which motivates further the choice of differentiating AP-states based on applications.

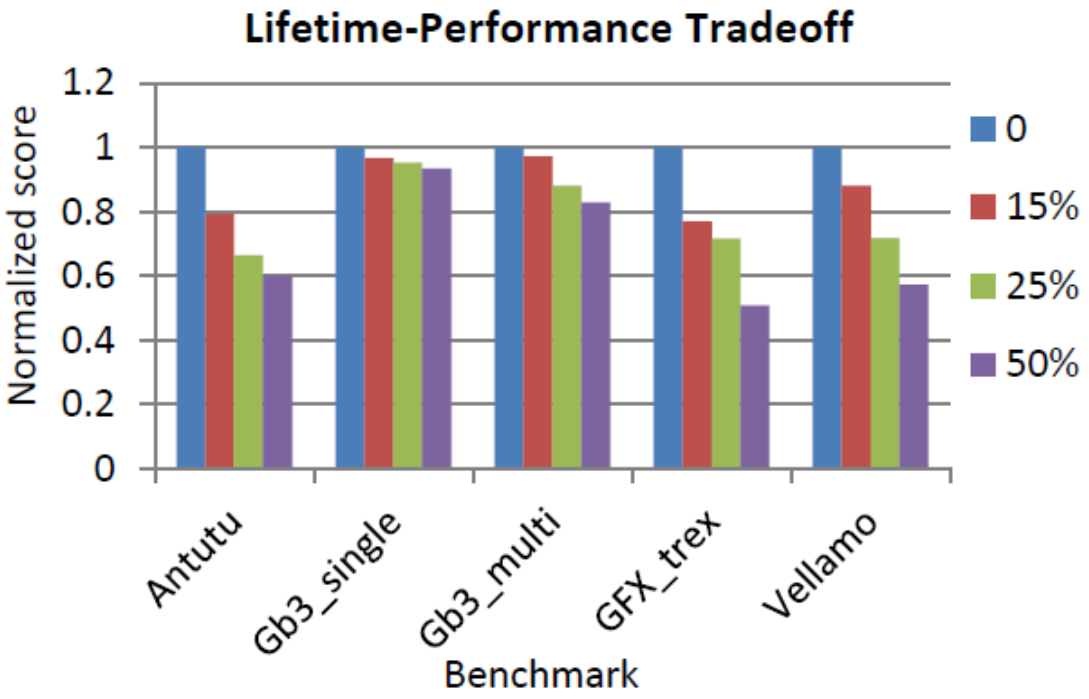

Figure 2.10: Benchmark evaluation of battery lifetime and performance tradeoff

Finally, in the next experiment we want to demonstrate the compatibility of our

framework with different kernel frequency governors. For doing this we execute the Vellamo Browser benchmark with a target time t_{charge} of 300 seconds (corresponding to a 25% extension over the non-controlled scenario) and a budget E_{budget} of 550 Joules. In Figure 2.11 we show the curve of consumed energy in three different cases, in which respectively are active the ondemand, interactive and conservative governor. This com-

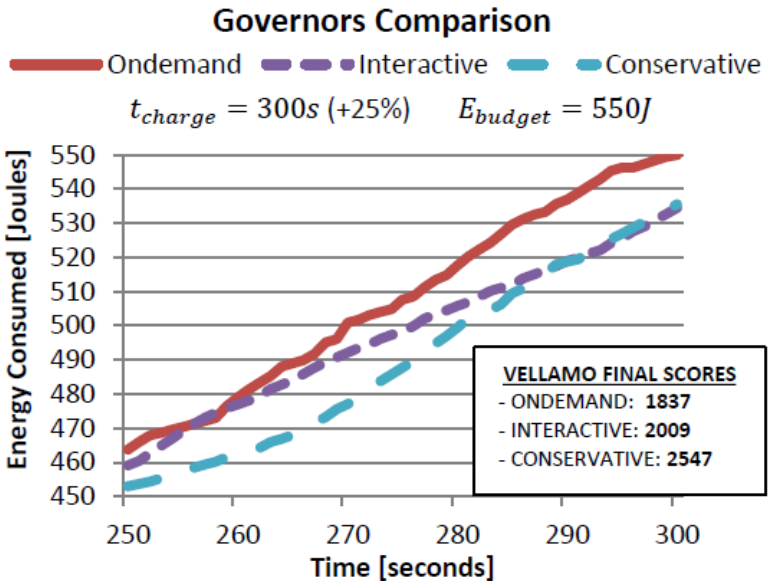

Figure 2.11: Comparison of different frequency governors

parison shows that our controller is compatible with different governors and in all cases the target on battery lifetime is met. It also shows that in the case of Vellamo benchmark the conservative governor shows a better energy efficiency with respect to the other two governors, as it achieves a higher score with a lower power consumption. This can be justified assuming that for the conservative governor, the Vellamo benchmark controllable power component prevails over the non-controllable one.

2.5 Conclusion

In this chapter we presented BLAST: Battery Lifetime-constrained Adaptation with Selected Target. BLAST is the first application-aware power management framework for mobile devices which controls operating conditions in order to meet a prede-

fined battery lifetime. We also presented a lightweight and portable implementation on a real Android device, compatible with different Frequency Governors. The experiments show that our solution is effective in guaranteeing the predefined target battery lifetime and that it still meets user experience requirements with a selected battery lifetime extension set to 25%. The average rating of real users is within 5% for a battery lifetime extension set to 25%.

Chapter 2 contains material from "BLAST: Battery Lifetime-constrained Adaptation with Selected Target", by Pietro Mercati, Vinay Hanumaiah, Jitandra Kulkarni, Simon Bloch and Tajana Šimunić Rosing, which appears in Proceedings of the 12th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services on 12th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MOBIQUITOUS'15) [111]. The dissertation author was the primary investigator and author of this paper.

Chapter 3

User-aware Joint Power and Thermal Management

In the previous chapter, we presented a solution to maximize performance subject to a target battery lifetime. In this chapter, we address the complementary problem of maximizing battery lifetime subject to application-specific performance requirements for user experience, and include thermal constraints. The proposed proactive joint power and thermal management solution leverages a novel thermal model of the entire smartphone to keep device temperatures within safe operating ranges. The mean temperature estimation error is 1.17^{o}C. Our ready-to-use management solution has been implemented on Google Nexus 5, and has been demonstrated to achieve a 46% application-specific and 35% device-level power savings. A custom frequency governor is also implemented for fine-grain power management, which enabled an additional 27% power savings.

3.1 Introduction

Modern smartphones and tablets are composed of heterogeneous subsystems (CPU, GPU, Display, Modem, 4G, WiFi, etc.), which execute a wide variety of applications such as multimedia (YouTube, camera), gaming (Angrybirds, Temple Run), social media (Hangouts, Whattsapp), browsing (Chrome). Many of these subsystems are power hungry and drain a lot of battery energy in executing the apps, causing incon-

venience to users in forcing them to charge their batteries more than once a day. Apart from limited battery charge time, tight form factors of the phone coupled with the absence of any active cooling mechanisms (such as fans) make the thermal management crucial to (i) maintain the circuit integrity, (ii) avoid inconveniences to the user [143].

CPU, GPU and display are the primary contributors to both power consumption and temperature [145], [34]. CPU and GPU are the device core computational elements, thus they have high switching activities. When operated at high frequencies and voltages, they exacerbate the dynamic power consumption. This also increases temperature, which depends on on-chip power density [21]. The display has a significant contribution to power consumption due to its relatively large form factor.

The focus of mobiles designers and developers is shifting from high performance to high user experience [38]. The quality of user experience is a broad concept which depends on a large number of variables, from personal tastes and preferences, to device operating conditions, to the applications currently running. User experience is hard to measure, but it can be easily defined as the situation in which the device behavior meets user expectations. Expectation and experience change depending on which application is executing in foreground on the mobile device. For example, the level of expectation of a user playing games is higher than that of a user browsing through websites. As a consequence, the level of performance that the device should provide to meet user expectation is different in these two scenarios (higher for gaming, lower for browsing). This means that in this example we can obtain a power reduction by simply operating the device at a lower performance when browsing and at a standard performance when gaming, without impacting user experience. Potential for power savings also comes from change blindness. This is defined as the inability of users to notice small changes in the brightness of the display over large time intervals. Therefore, even if starting from a standard brightness at the beginning of a session, brightness can be slowly reduced to save power [145], [59].

Power management techniques which have been recently proposed in literature do address user experience, but they have major limitations that make them impractical, e.g. (i) the techniques are very specific to certain applications [13], [131]; (ii) they are not feasible to implement or they are not compatible with the existing standard ap-

proaches, thus they would require expensive OS restructuring [134], [121]. For such reasons, the actual power management of mobile devices has not changed much over the years.

Thermal management refers to a set of techniques aiming at keeping the temperature of electronic systems under control. It is a very actively researched area, spanning from high performance multi-clusters to mobile devices [143], [130], [62]. Thermal management can be broadly classified into reactive, if decisions are based on the current or the immediate past temperature, or proactive, if decisions are based on the predicted future temperature. Proactive thermal management requires a thermal model for predicting future temperatures. Most of the thermal modeling techniques proposed in the literature use the actual power measurements to predict temperature. Unfortunately, in mobile devices such power measurements are not available, as they do not have power sensors. An alternative involves deriving power consumption through power models, which depend on the device performance metrics. However, power models suffer from inaccuracies and could be too computationally expensive for runtime management.

Considering this background, we identify limitations in the standard power and thermal management schemes present in mobile devices today. Assuming the Android operating system (OS) as a reference, power management at the Linux kernel level is actuated by *frequency governors*, which switch voltage and frequency at a high rate (in the order of milliseconds) to match workload requirements and meet the set goals, such as high performance or low power consumption. Today, a standard governor, called ondemand, scales frequency according to CPU utilization [124]. On the other hand, thermal management is handled at the user space level. In Qualcomm-based devices such as the Google Nexus 5 and APQ8064 based tablets, temperature is controlled by a user space daemon called Thermal Engine (TE), interfaced with temperature sensors. If a certain temperature threshold is exceeded, the TE reacts by applying a set of actions defined in its configuration file, such as limiting the frequency range or shutting down the device. This power and thermal management infrastructure is subject to limitations. Power and thermal management are not coordinated and thermal management is purely reactive. Also, power management is agnostic of which applications are currently running and treats all of them in the same way.

In this work, we propose a novel user-centric proactive joint power and thermal management, designed for mobile devices, with the following contributions. We design management to be proactive and implement temperature prediction on the device itself. We develop a general procedure to derive a simple yet accurate thermal model of the entire phone, based on observable quantities like CPU and GPU frequencies, and display brightness. Thus, it does not require actual power measurements. Our thermal model has a mean error of 1.17^{o}C. We account for change blindness in our comprehensive power and thermal management. We propose a lightweight and ready-to-use implementation for the user space manager, which is compatible with the standard Android environment and requires no OS modifications. Finally, we implemented a custom frequency governor, which provides cross-layer interface to our user space manager. The custom frequency governor shortens the activation interval from 1 second to tens of milliseconds and provides additional power savings.

We test our management technique in comparison with the standard available power and thermal management schemes on a real Android device, the Google Nexus 5, with a set of common applications. The results shows that our solution can achieve up to 46% of application-specific power savings and up to 35% of average power savings on average at the device level. Moreover, we show that our custom governor achieves 27% additional power savings with respect to the ondemand governor.

3.2 Related Work

Battery lifetime is a key limiting factor for mobiles. There has been a large focus in developing energy efficient solutions in the last few years. Some of the existing literature addresses the problem of energy efficiency at the application level. Reference [102] presents a tool to automatically identify energy hotspot in Android applications. Reference [11] presents a method for appropriate wakelock acquire and release placement to achieve higher energy efficiency. Work presented in [89] requires the maintenance program to explicitly specify the suspected energy consumption hotspots in a mobile application, so that parts of it can be automatically offloaded to the cloud. Such approaches require app developers to be aware of system energy consumption.

Other publications implement energy efficient solutions at the middleware and the OS level. An overview of such techniques is presented in [36]. Reference [84] presents a temperature-aware frequency scaling strategy that can be alternatively configured for performance optimization or power optimization. Reference [98] presents a prediction technique to estimate memory access rate and select operating frequency accordingly. Similarly, Reference [97] presents techniques to achieve energy efficiency by scaling frequency according to memory access rates. However, none of the aforementioned publications account for user experience in the area of mobile devices.

The attention of both industry and research on user experience in addition to performance has grown consistently in the last decade, and many publications aim at achieving both high user experience and energy efficiency for mobile devices. Various papers aim at measuring and modeling user experience [38], [145], [49] by collecting user feedback and correlating with users' personal profile. The message that emerges from this area of research is twofold. First, it is extremely challenging to measure and model user experience. Second, since the primary focus in mobile phones is to ensure quality user experience, power management should account for it. As claimed in Reference [71], to date there is a weak understanding of which factor influence the mobile user's quality of experience.

For this reason, some of the existing techniques adopt a practical strategy and allow users to configure power management based on personal preferences. The authors of [121] propose a novel task allocation infrastructure, which accounts for user experience as the user can assign different levels of priorities to different applications. Similarly, authors of [104] propose a way to identify and select different priority levels for applications and suggest how this can be used to optimize power management. Reference [95] delivers a power capping technique for mobile devices, which relies on a self-adaptation scheme driven by system activity. The presented technique is effective in achieving power savings, but it neglects thermal management.

Another class of approaches instead of providing configuration interface to the user, models user experience depending on perceived delays and alternation of activity and idle periods. Reference [86] proposes an event-driven power management framework which increases CPU frequency in response to interactive events, in order to mini-

mize user's perceived delay. Work in [99] presents a power management strategy which uses hardware timers to capture idle and activity time intervals. A model for a typical user session activity is proposed in [145]. The authors then use the proposed model to compare various power management strategies. Also, the above work exploits change blindness to reduce display power consumption. The technique described in [166] employs a novel scheduling algorithm called energy-based fair queuing (EFQ), which achieves energy proportionality in battery-constrained systems while improving user experience. Another power efficient scheduling approach is presented in [134]. The approach takes advantage of heterogeneous platforms to allocate tasks with different energy impact. While they achieve better energy efficiency, these approaches require an event-detection and timing infrastructure which involves modifications at the OS level. This makes the implementation challenging and can affect the portability of the framework between different devices. This is even exacerbated when the power management policy involves task allocation. In fact, the scheduler is one of the most critical sections in an OS. Modifying it is risky and time expensive [105].

Some recent publications address power management in mobile phones for specific applications. Reference [131] presents a unified dynamic voltage scaling (DVS) algorithm for CPU and GPU, which takes advantage of the execution profiles of 3D games to improve energy efficiency. The technique presented in [126] also focuses on GPU-intensive workloads. Power management techniques presented in [47], [46] reduce power consumption for gaming by adapting to the current and the predicted program state. Work in [94] optimizes power consumption for YouTube by intelligently scheduling download activities. The technique in [13] adopts tone mapping technique to adapt brightness and reduce LCD backlight level for mobile games, without compromising user experience. A similar technique is presented in [83]. Even though these techniques successfully target power reduction and user experience, they are developed specifically for a target application, thus they cannot be generalized.

Thermal management is a broad area of research, especially in the field of high performance multiprocessors [143], [62]. Reference [62] proposes an optimal closed-loop power control algorithm for multicore systems, that keeps the temperature in a safe range. Reference [21] presents a distributed energy-aware thermal management solu-

tion exploiting model-predictive control. Thermal model identification has also been investigated. Reference [22] proposes a gray-box identification technique to extract the thermal model of a many core platform. Few publications target thermal management for mobile devices. Reference [169] presents a thermal simulator for smartphones which can be used to generate accurate thermal maps. The methodology employed is static and cannot be used for dynamic management of temperature. In [130], the authors present a thermal model of a mobile system and use it to develop a thermal management strategy. The thermal model presented in [170] focuses on the thermal interaction with the battery subsystem. None of the above works present a coordinated power and temperature management solution. To the best of our knowledge, our proposed work is the first prototype of a ready-to-use implementation of a user-aware proactive joint power and thermal management technique, which can optionally be interfaced with a custom frequency governor enabling cross-layer management.

3.3 Thermal Modeling Methodology

Most mobile devices consist of heterogeneous computing elements equipped with temperature sensors. They have a set of operating conditions $\mathbf{F}$ that can be controlled at runtime, which influence the device power consumption and temperature at various locations of the phone.

In this work, we control only the CPU frequencies, GPU frequency, and display brightness, since CPU cores, GPU and display are the major contributors to the power consumption, devices temperature and user-experience [145]. The above set of operating conditions is denoted by $\mathbf{F} = [f_{c0}, f_{c1}, f_{c2}, f_{c3}, f_{GPU}, \beta]$, where f_{ci} is the frequency of CPU core i, f_{GPU} is the GPU frequency, and $\beta \in \{0, 1, \ldots, 255\}$ is the display brightness level. Indeed, most mobile devices today have embedded multi-core processors with per-core voltage and frequency control capability [32]. In the following, all vectors and matrices are denoted with **bold** text.

The reference device used in this work is Google Nexus 5. This device has a quad-core Qualcomm's Krait processor with Adreno 330 GPU and runs the Android operating system. The smartphone has 11 temperature sensors, referred to as *thermal*

zones, whose values are exposed through `sysfs` interface of Linux kernel. The set of temperatures are denoted by vector $\mathbf{T} = [T_0, \ldots, T_{10}]$.

The thermal model of a processor is described by the compact modeling techniques such as HotSpot, which uses electro-thermal analogy to describe heat spreading and storing phenomena [68]. The simplified state space representation is shown in Equation (3.1) [143], [62].

$$\frac{d\mathbf{T}(t)}{dt} = \mathbf{A}'\mathbf{T}(t) + \mathbf{B}'\mathbf{P}(t), \tag{3.1}$$

where $\mathbf{A}'$ and $\mathbf{B}'$ are constant matrices. They are related to the thermal conductance ($\mathbf{G}$) and capacitance ($\mathbf{C}$) matrices as follows: $\mathbf{B}' = \mathbf{C}^{-1}$; $\mathbf{A}' = -\mathbf{B}'\mathbf{G}$. Power consumption is represented by $\mathbf{P}$. It is not necessary to have same dimensions for both $\mathbf{T}$ and $\mathbf{P}$. We discretize the above equation for a constant sampling interval Δt and we obtain the results of Equations (3.2) and (3.3). Here k indicates the k^{th} time instant, and $\mathcal{I}$ denotes identity matrix of the same size as $\mathbf{A}'$.

$$\frac{\mathbf{T}_k - \mathbf{T}_{k-1}}{\Delta t} = \mathbf{A}'\mathbf{T}_{k-1} + \mathbf{B}'\mathbf{P}_k, \tag{3.2}$$

$$\mathbf{T}_k = ((\mathbf{A}' - \mathcal{I})\mathbf{T}_{k-1} + \mathbf{B}'\mathbf{P}_k)\Delta t. \tag{3.3}$$

We assume that the operating conditions have linear dependency with power with negligible loss in accuracy. This is because many commercial mobile devices do not have power sensors, i.e. $\mathbf{BF}_k = \mathbf{B}'\mathbf{P}_k\Delta t$. Replacing $\mathbf{A} = (\mathbf{A}' - \mathcal{I})\Delta t$ results in an alternative thermal model equation, reported in Equation (3.4).

$$\mathbf{T}_k = \mathbf{A}\mathbf{T}_{k-1} + \mathbf{B}\mathbf{F}_k. \tag{3.4}$$

The advantage with the above modeling approach is that the operating conditions are usually exposed to the user space and they can be switched without the need for any operating system (OS) modification.

To extract the above thermal model for a real device, such as Google Nexus 5, we developed a simple profiling tool which periodically samples temperatures and operating conditions. The sampling rate was set to 1 second. This is implemented as a bash script that runs on the phone and can be launched with a terminal emulator or

an application manager. The tool reads the files related to the 11 temperature sensors and those related to the 6 selected operating conditions mentioned before, and then it writes values to a file. The output file is then fed to a least-square solver implemented in MATLAB to derive matrices $\mathbf{A}$ and $\mathbf{B}$.

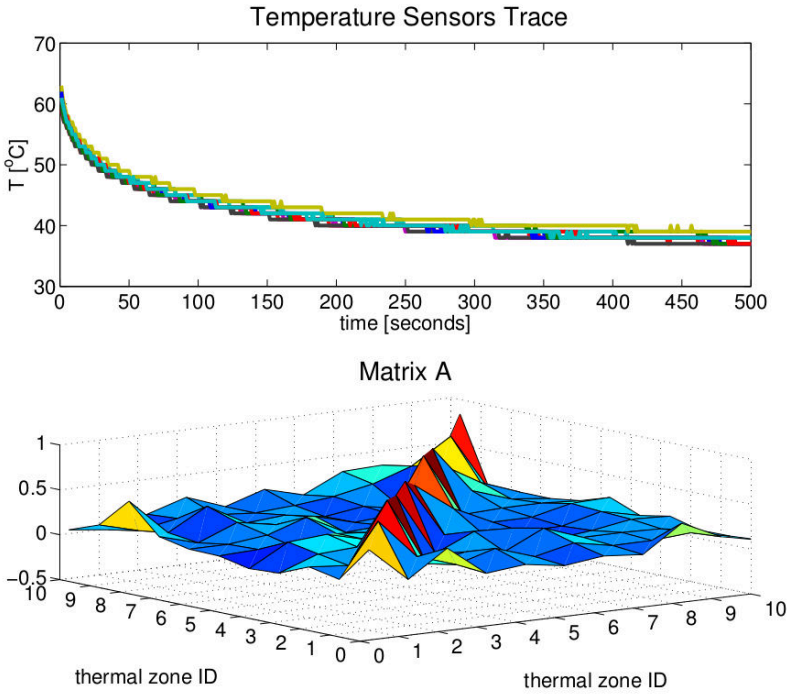

Figure 3.1: Top figure: Cooling transient temperature trace; Bottom figure: matrix $\mathbf{A}$

Extracting the model consists of computing matrices $\mathbf{A}$ and $\mathbf{B}$. The procedure we propose involves solving two least squares regression problems similar to reference [62]. The first problem is obtained by imposing $\mathbf{F} = \mathbf{0}$, which gives the expression of Equation (3.5).

$$\mathbf{T}_k = \mathbf{A}\mathbf{T}_{k-1} \tag{3.5}$$

The solution of Equation (3.5) gives matrix A. This can then be substituted in (3.4) and solved for $\mathbf{B}$. The computation of matrices $\mathbf{A}$ and $\mathbf{B}$ is done only once for a given phone, as the coefficients are dependent only on the geometry of the phone and the material composition. The computation is done offline as the resources available on a mobile device are limited.

Figure 3.1 shows a sample temperature trace used for determining matrix $\mathbf{A}$ (top figure), and the resultant matrix is shown in the bottom figure. In practice it is not possible to set $\mathbf{F} = \mathbf{0}$, as this would mean switching off the device. To obviate this, we set $\mathbf{F} = \mathbf{F}^{min}$, where $\mathbf{F}^{min}$ is the set of minimum operating conditions that guarantee functionality for the profiling tool, e.g., $\mathbf{F}^{min} = [f_{c0}^{min}, 0, 0, 0, 0, 0]$. As shown in Figure 3.1, the temperature trace used to derive $\mathbf{A}$ should capture the device cooling rate. On the other hand, the trace recorded for deriving $\mathbf{B}$ should capture the real user activity. Results for matrix $\mathbf{B}$ are analogous to those shown in Figure 3.1.

Figure 3.2 shows the case of a trace used to validate the model and its accuracy, which is expressed in terms of root mean square error (RMSE). The trace consists of temperatures collected for 500 seconds of execution of AnTuTu benchmark for 3 consecutive runs [2]. The top two plots show the real temperature trace, while the middle two plots show the model-predicted temperature and the predicted temperatures using the immediate previous temperature, respectively. The model-predicted temperature is derived thanks to the thermal model described in Equation (3.4). Most reactive thermal management schemes predict temperatures using the immediate previous temperatures.

Finally, the bottom two plots show the maximum and the mean prediction errors obtained in the above two cases. From the plots, we can see that our model-based prediction has a mean RMSE error of **1.17**^{o}C, which is an improvement of **55**% with respect to previous temperature-based prediction. Also, our model achieves **80**% reduction in peak error with respect to previous temperature-based prediction.

3.4 Management Formulation and Implementation

In this section, we present the formulation of the joint power-thermal management problem, present our solution, and describe the power-thermal manager and its implementation. The joint power and thermal management problem formulation is described in Equations (3.6) and (3.7).

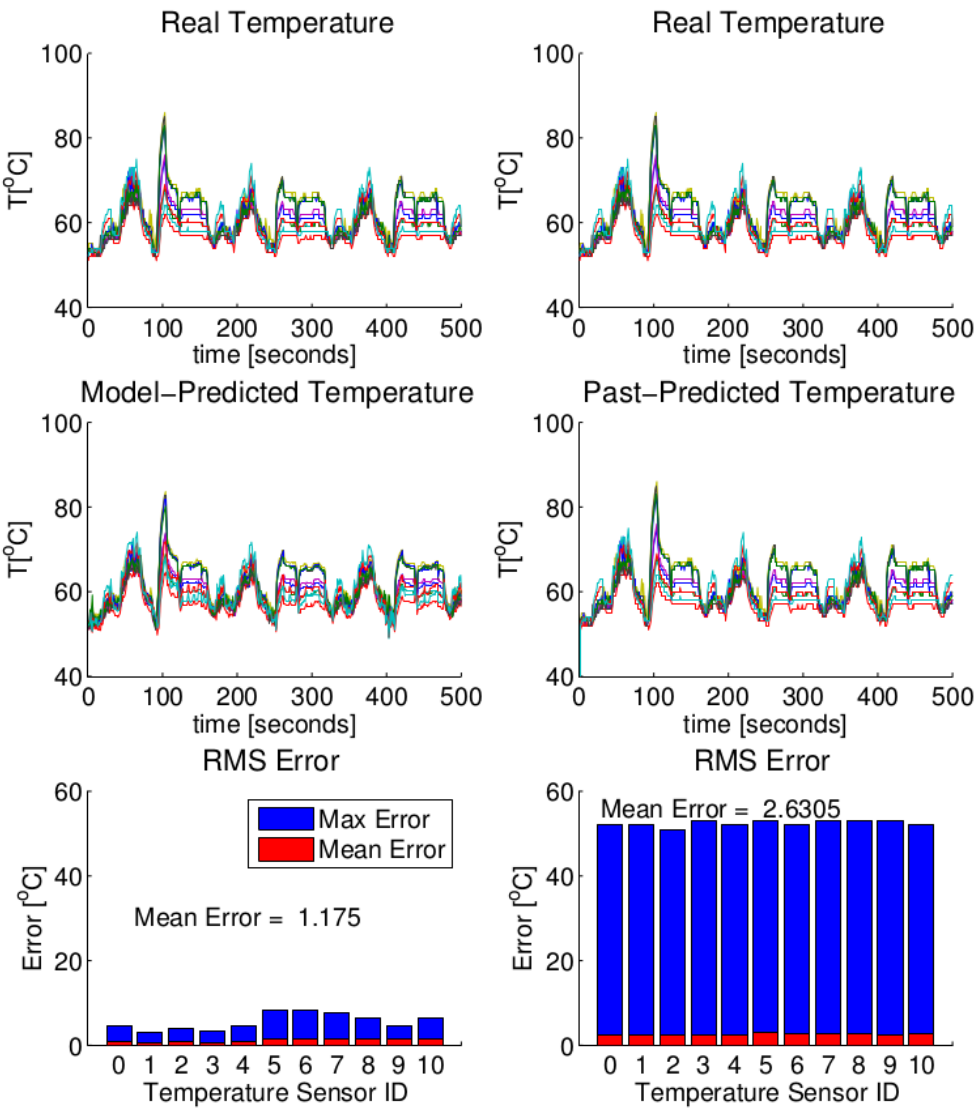

Figure 3.2: Results showing the accuracy of the proposed thermal model

$$\min_{\mathbf{F}_{k+1}} \quad \left|\mathbf{F}_{k+1}^{req} - \mathbf{F}_{k+1}\right|, \tag{3.6}$$

$$s.t. \quad \mathbf{T}_{k+1} \leq T_{thr}. \tag{3.7}$$

In the above formulation, $\mathbf{F}_{k+1}$ represents the set of operating conditions that will be applied at the next time step and it is the output of the management problem. $\mathbf{F}_{k+1}^{req}$ is the set of required operating conditions at the next time step, and it is a function of the application executing in foreground and of the level of user expectation. We assume that if $\mathbf{F}_{k+1} \geq \mathbf{F}_{k+1}^{req}$, then the user experience is met. The term $\mathbf{T}_{k+1}$ represents the vector of system temperatures at the next time step, which are predicted using the model presented in Section 3.3. The temperature threshold T_{thr} is also a function of the foreground application and of the user expectation. In this work the threshold is a single value, but we can extend the idea by defining one threshold per each sensor. A different choice of vectors $\mathbf{F}$ and $\mathbf{T}$ is consistent with the problem general formulation. The goal of this management problem is to never give more performance than what is required from a user's perspective. This is the key source of power reduction in our solution.

Figure 3.3 shows the block diagram of the proposed management technique. The main components are the **Manager** and the **Application Observer**. The proposed implementation runs in Linux user space and requires no modifications at the operating system level. In the following subsections, we describe the main components more into details.

3.4.1 Sysfs Interface

The **sysfs interface** is the main interface between the Linux kernel and the user space, in which the OS kernel exposes data such as current operating conditions and temperature sensors values. In the current versions of Android/Linux it is also possible to set operating conditions through the sysfs interface, such as CPU frequency, GPU frequency and display brightness. Therefore, the power and thermal management can be completely implemented in the user space, without modifying the OS.

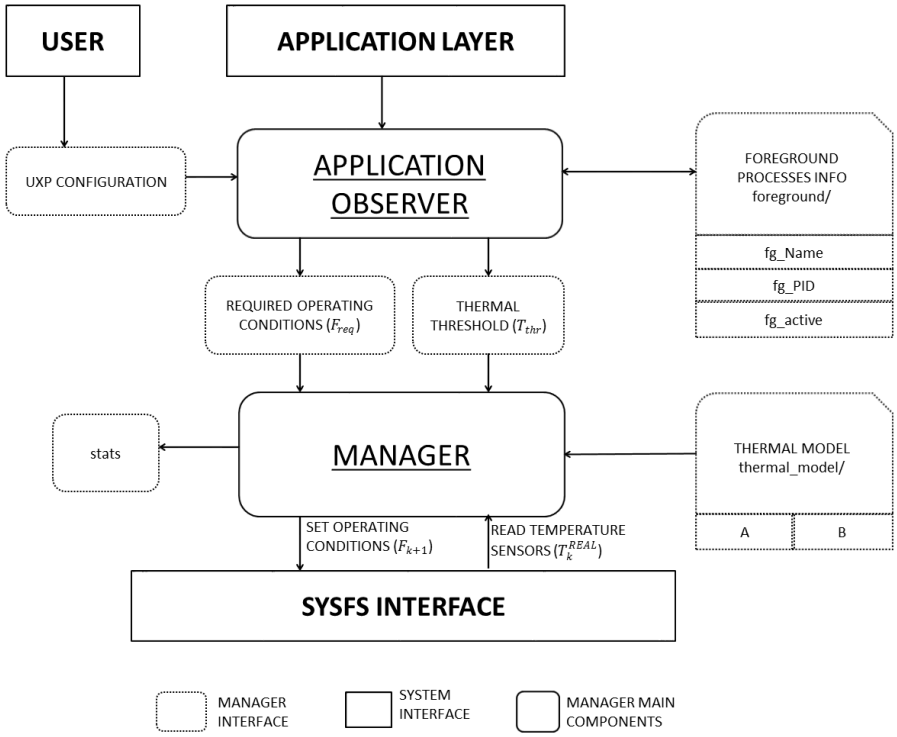

Figure 3.3: Block diagram of the proposed power-thermal manager

3.4.2 Manager

The Manager is the component that finds the optimal operating conditions for the mobile device. When starting execution, it loads matrices $\mathbf{A}$ and $\mathbf{B}$. The two matrices are computed offline according to the procedure described in Section 3.3.

The manager wakes up at a fixed time rate and reads the current temperatures $\mathbf{T}_k$ from the sysfs interface. It also reads the required operating conditions $\mathbf{F}_k^{req}$ and the thermal threshold T_{thr} from the user experience configuration files. Using these data it first computes $\mathbf{T}_{k+1}$ using Equation (3.4), then it computes the operating conditions $\mathbf{F}_{k+1}$ by solving the problem described in Equations (3.6) and (3.7) and applies them by writing values to the sysfs interface. The manager also writes data to the `stats` file, for debugging purposes. The Manager source code is written in C code and it is cross-compiled with Android Native Development Kit (NDK) tools to run on our target device, the Google Nexus 5 [1]. The default activation rate is 1 second, as this was found to be a good tradeoff between performance and overhead. To make the manager as less in-

trusive as possible, we decided to set a maximum bound on CPU and GPU frequencies, rather than fixing a constant value. This design choice guarantees the compatibility with the kernel-level power management schemes actuated by frequency governors. Indeed, these modules activate at a much higher rate (the Ondemand governor activates every 20 ms) [124]. The manager the determines the value $\mathbf{F}_{k+1}$ through a simple heuristic algorithm. The manager first predicts temperature by computing Equation (3.4) with $\mathbf{F}_{k+1} = \mathbf{F}^{req}_{k+1}$. If the required operating conditions $\mathbf{F}^{req}_{k+1}$ are such that they do not cause the temperatures to exceed the threshold, then the manager applies $\mathbf{F}_{k+1} = \mathbf{F}^{req}_{k+1}$. Otherwise $\mathbf{F}_{k+1}$ is reduced and predicted temperature is checked again until the threshold is met. The strategy is to reduce $\mathbf{F}_{k+1}$ and take advantage of the fact that CPUs in mobile devices have predefined voltage and frequency operating points. GPU frequency is not reduced, because the manager assumes that if the required GPU frequency is high, then it means that the foreground application involves graphics and requires the requested performance. In this case, penalizing the GPU may sacrifice the user experience.

Display brightness is handled separately, according to the concept of change blindness. When a new application is launched, display brightness β is set to the required value. After that, the manager progressively reduces the brightness until reaching a predefined minimum value β^{min}. The rate at which β is reduced and the minimum value β^{min} are parameters that can be tuned in the manager configuration. The chosen values should guarantee that the user does not perceive brightness changes and degrade user experience [145].

3.4.3 Custom Governor

The proposed user space manager is compatible with any frequency governor, as it only limits the maximum frequency. However, the standard governors are not able to distinguish between different applications. The main reason is that they cannot access information about the task currently scheduled (e.g. there is no cross layer interaction). In particular, the governors do not know the process ID (PID) of the scheduled task. We extend our user space implementation with a custom frequency governor and a cross-layer infrastructure, which allows passing such information from the user space to the kernel space.

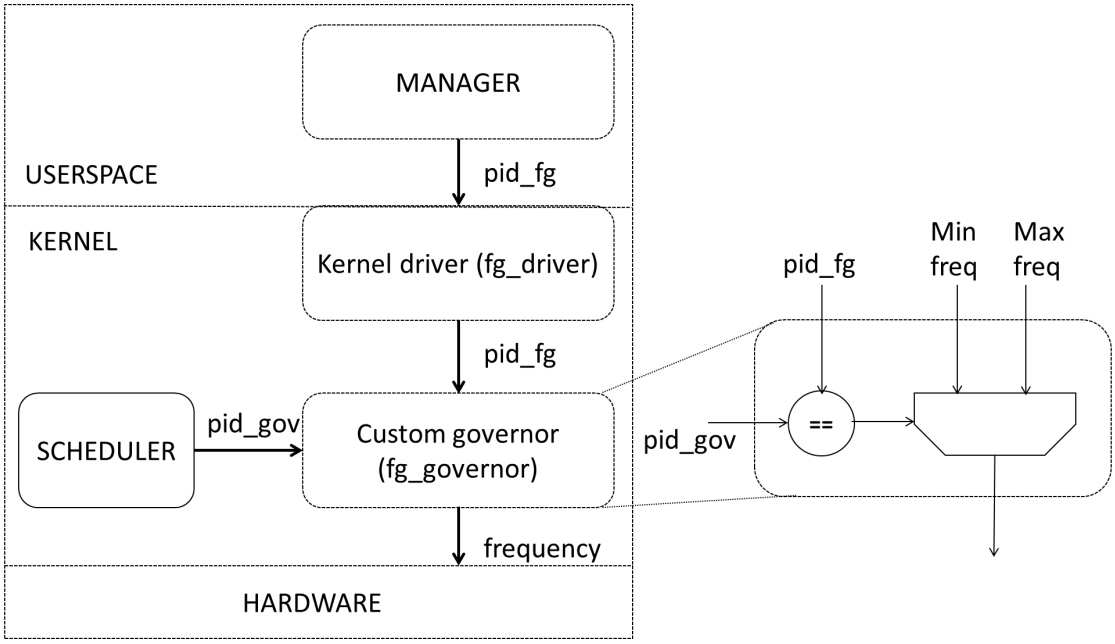

Figure 3.4: Interaction between the manager and the custom governor

Figure 3.4 shows the main components of such implementation. The Manager is the user space component, which was described in the previous subsection. In the source code for this component, we added a function devoted to get the PID of the current foreground application, open the kernel driver (`fg_driver`) and write the value of the pid to the kernel space. The kernel driver is a Linux driver implementing a write method to store the value of the current foreground application in the shared kernel variable pid_fg. The driver is implemented in C code and cross-compiled for ARM-based platforms with Android tools.

The custom governor is a frequency governor, which periodically checks the PID of the task currently scheduled (pid_gov). If pid_fg and pid_gov are equal, then the governor selects the maximum frequency (which is constrained by the user space manager), otherwise it selects the minimum. In this way, the controller is able to switch operating conditions based on different applications at a finer time granularity. In fact, the governor periodically activates with a rate equal to the kernel scheduling tick (100 Hz). The custom governor has been implemented starting from the structure of the Ondemand governor. We made two major modifications with respect to it. First, we introduced the variable pid_gov (standard governors do not have information about process IDs). Sec-

ond, we replaced the Ondemand algorithm with the selection criteria described above.

The scheduler is the standard Linux scheduler (in the version of the operating system that we used is the Completely Fair Scheduler). The only modification that we applied here is to define the shared variable pid_gov. Then, every time the core function scheduler_tick() is invoked, the pid of the task currently scheduled is saved to pid_gov. Since the variable is shared in the kernel, the value will be immediately available for the governor as well. We do not apply any modification to the scheduling algorithm.

Such implementation is easy to port between different devices. In fact, the kernel driver and the custom governor can be compiled and mounted for any Android device which supports the cpufreq driver. The kernel driver can be mounted using the insmod command. As for the governor and the modifications to the scheduler, these require recompiling the operating system and flashing the new kernel image in the device.

3.4.4 Application Observer

The Application Observer monitors which application is currently running in the foreground on the device. Based on this and on the User Experience (UXP) configuration provided by the user, it periodically updates the required operating conditions $\mathbf{F}_{k+1}^{req}$ and the thermal threshold T_{thr} for the current execution, by writing them to the appropriate file locations. The source code for the application observer is also written in C code and cross-compiled with NDK tools. In the current implementation, the Application Observer wakes up at a fixed rate (default is 1 second) and executes the `top` command. The output of this command is the list of running processes with their properties. In particular the PCY field in the top denotes whether a process is in foreground (flag "fg") or in the background (flag "bg"). Usually there is not only a single process with an "fg" flag, but rather there is a list of them, and the application executing on the display is one of them. For this reason the observer parses the output of top and writes the names of all foreground processes to the file fg_Name, and the list of process IDs to the file fg_PID. Then it compares the list of current foreground processes with the UXP configuration. The UXP configuration contains the list of required operating conditions configured by the user for a set of applications, together with the desired thermal threshold. If any application in the UXP configuration appears in the fg_Name file, then the

corresponding required operating conditions are written to the file containing $\mathbf{F}^{req}$, and the corresponding thermal threshold is written in the file containing T_{thr}.

3.4.5 The User Experience (UXP) Configuration

The UXP Configuration is obtained by the configuration manager which requires a user to execute an application of his/her choice under the maximum operating conditions. While the user is executing the application in foreground, the configuration manager progressively lowers the operating conditions. When the user notices a degradation in the user experience of the application, s/he notifies the configuration manager, which registers the least acceptable operating conditions for that application. Alternatively, the procedure can start from the minimum operating conditions and increase them until the user experience of the application behavior reaches acceptable levels. The user is also asked to choose the default operating conditions for the device. These are used in the case the Application Observer does not find any correspondence between the list of foreground processes and the applications in the UXP configuration. The approach though it requires the user's effort (relatively low) in setting up the UXP configuration, it has the advantage of being personalized for that user. Such a provision is not available in other UXP model-based approaches such as [86], [99]. Finally, thermal thresholds are selected based on the accepted values typically used for thermal management, e.g. Qualcomm's Thermal Engine daemon.

3.4.6 Manager Installation and Execution

To install the management framework on an Android device it is just required to copy the executables for the Manager and for the Application Observer to a location in the phone file system. After that it is sufficient to launch both and let them run in the background. To insure the correct behavior, other power or thermal management program running in the User Space should be stopped first. However, we have verified that the manager is compatible co-running with Qualcomm's mpdecision. This is a service that automatically handles CPU hotplug based on computational load.

3.5 Experimental Results

The target platform for the evaluation of the proposed user space solution is a Google Nexus 5 smartphone. This device is equipped with the Qualcomm Snapdragon 800 chipset, featuring a quad-core Krait 400 CPU. The four cores have a range of frequency from 300 MHz up to 2.26 GHz, with predefined voltage and frequency operating points. It also has an Adreno 330 GPU, with frequency from 200 MHz up to 450 MHz. The display is a Full HD with In-Plane Switching (IPS) LCD [4] with 255 brightness levels. The phone was rooted to get root privileges to control the frequencies of CPU and GPU, and display brightness.

The evaluation has been conducted selecting applications among the most popular ones: Chrome, Gmail, Angrybirds, Hangouts, Dialer (standard phone call), Camera, YouTube [49], [5]. To ensure that comparisons are consistent and reproducible, we wrote an application that allows to record and replay sequences of events on the device (e.g. touches on the display, pressing the power button, pressing the volume button, etc.).

In this device they are already executing by default two other user space managers: the Thermal Engine and mpdecision. Moreover, the standard Ondemand governor is operating at the Kernel level. The Thermal Engine operates by limiting the maximum operating conditions, based on current temperature. To avoid interference with our manager, it should be stopped. Unfortunately, this program cannot be stopped and it is necessary to reduce its functionalities by replacing its configuration file with an empty file. As for mpdecision, this program switches on and off CPU cores based on utilization metrics. Since it does not perform frequency scaling, it should be kept active and compatibility should be guaranteed.

Figure 3.5 reports an extract of 20 seconds of an execution trace monitored while applying the proposed manager, to describe the details of the behavior of the proposed manager. For this experiment we are executing Google Chrome application with the required operating conditions $\mathbf{F}^{req}$ = [1 GHz, 1 GHz, 1 GHz, 1 GHz, 200 MHz, 150]. The first plot reports the temperature traces of the 11 sensors present in our device. For simplicity, no legend is reported here. In general the exact location of sensors is not disclosed. In this case, the thermal threshold was set to 80^{o}C, so temperature is kept

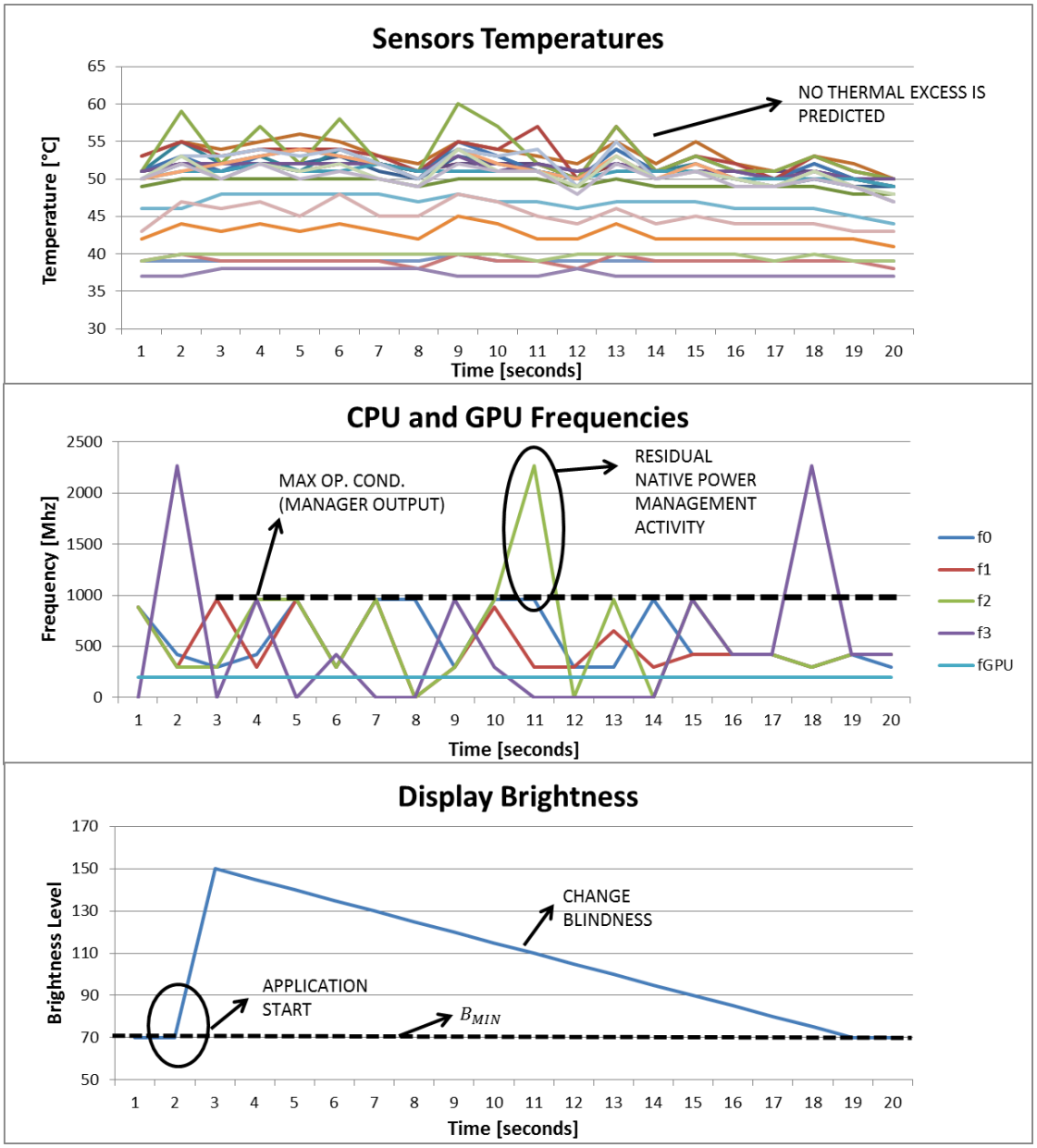

Figure 3.5: Behavior of the proposed manager

in a safe range. The second plot reports the behavior of CPU and GPU frequencies. Once the application is launched (at 2 sec), the Application Observer detects it running in foreground and selects the corresponding required operating conditions. No thermal management is activated in this case, so the manager always selects $\mathbf{F}_{k+1} = \mathbf{F}^{req}_{k+1}$. The black dotted line in the second plot represent the required CPU operating frequency, which is the maximum bound set by the manager. Indeed, the Ondemand governor at the Kernel level is free to switch frequency at a higher rate. Moreover, when a CPU core frequency is equal to zero, then it means that mpdecision has switched the core off. At 11^{th} second we can notice that the frequency of CPU 2 exceeds the bound imposed by the manager. This is due to the combination of Thermal Engine, mpdecision, and Ondemand governor activities, but it does not affect the temperatures much. Finally, the third plot shows the behavior of display brightness. When an application is launched, the required level is set (150 in this case). After that, change blindness is exploited and brightness is progressively reduced by 5 every seconds until the predefined minimum value β^{min} is reached (70 in this case).

In Figure 3.6 we show what happens when the manager detects a thermal emergency (e.g. the threshold T_{thr} is exceeded by some sensor predicted temperature). While executing a sample trace of the popular game Angrybirds, we monitor the temperatures and the CPU maximum frequency (e.g. the output of the manager) in two different cases. In the first case (on the left) the thermal threshold for the target application is set at 70^{o}C, in the second case, the threshold is 60^{o}C. For simplicity, only the temperature of the two sensors which detect thermal excess is reported, namely T9 and T10. In both cases, the maximum operating conditions are modulated by the manager in order to keep the predicted temperature lower than the threshold. The threshold can be also tuned to provide different performance tradeoffs.

Figure 3.7 reports our results in terms of power savings. Table 3.5 shows the configurations used for various applications in our evaluation of the proposed power-thermal manager. We record and replay event traces for each application and measure power consumption respectively with the default Thermal Engine and with the proposed user-centric manager. In both cases, mpdecision is active and the Ondemand governor is set. Power consumption is monitored by sampling the battery voltage and current

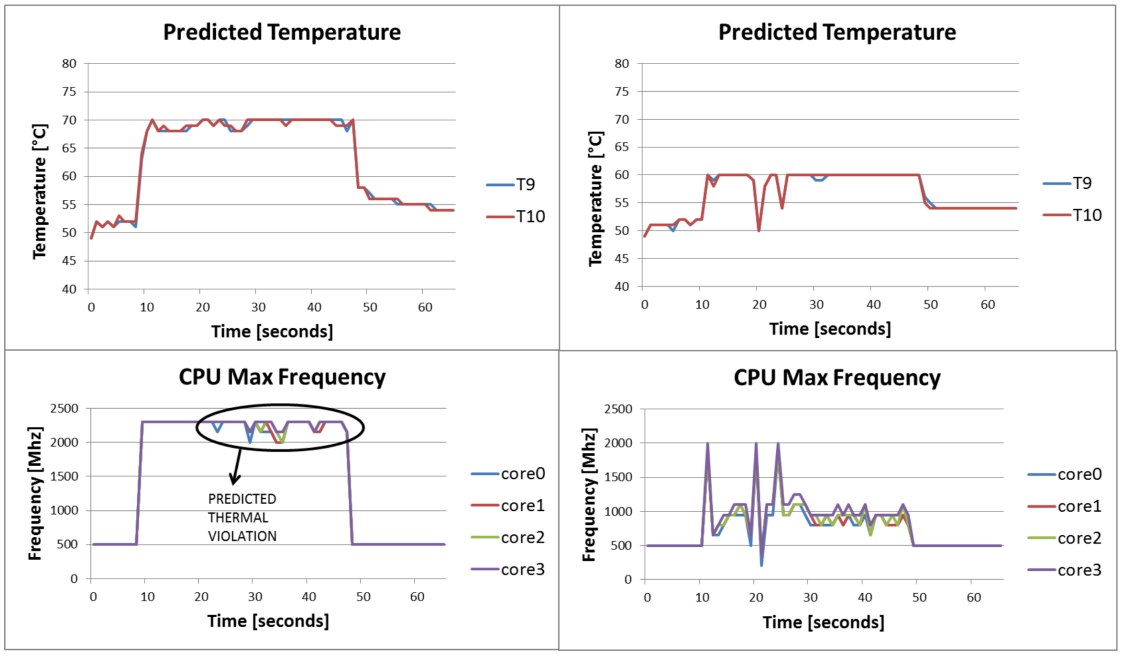

Figure 3.6: Example of the proposed manager handling thermal emergenices

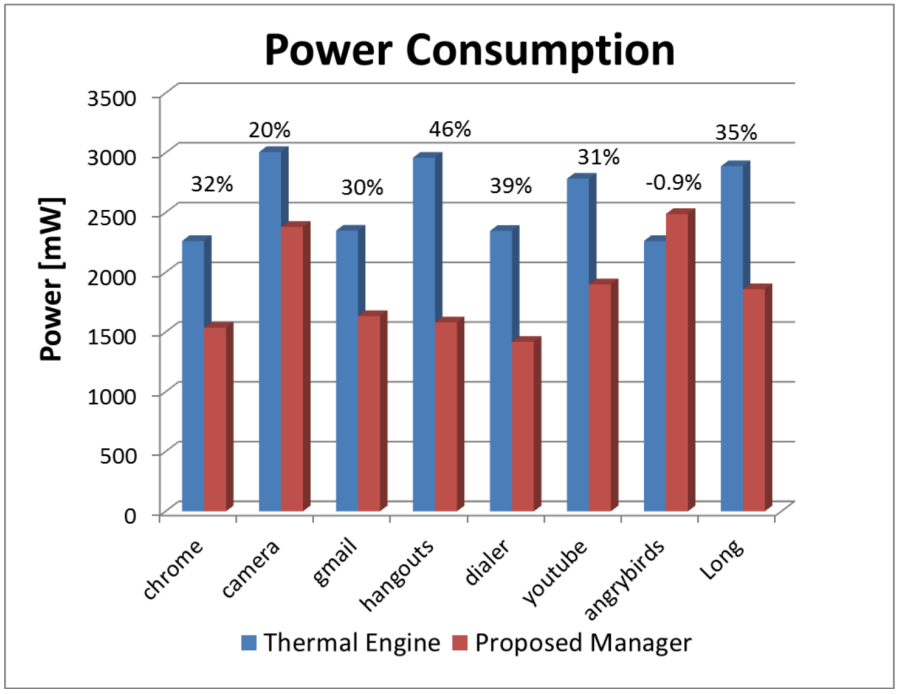

Figure 3.7: Power savings resulting from the use of the proposed manager

Table 3.1: Configuration values used for various applications in the proposed manager

Application	CPU freq. (MHz)	GPU freq. (MHz)	Display brightness
Chrome	1000	200	150
Camera	1500	350	150
Gmail	800	200	70
Hangouts	900	200	100
Dialer	1300	200	55
YouTube	1500	450	150
Angry Birds	2260	450	255
Default	500	200	100

values at the sysfs interface. In devices where this is not available, it is required to use an external power monitor. The traces are for a single application with 1 minute duration. Then we also compare power consumption on longer traces which combines all the considered applications (labeled as "Long" in the bar graph), which have 10 minutes duration. The values of power consumption in Figure 3.7 are average values.

For all the considered applications, our manager provides a power saving w.r.t. the standard power management, up to 46% in the case of Hangouts. The only exception is the case of Angrybirds. In this case the user expectation is very high, so the required operating conditions are set at the maximum. In the case of Long traces, our manager provides up to 35% average power saving w.r.t. the standard power management.

3.5.1 Improvements with the Custom Governor

In this experiment, we intend to evaluate the proposed custom governor in terms of power savings and we show that it is able to achieve further energy efficiency with respect to the standard ondemand governor. As described in Section 3.4.3, the custom governor selects the maximum frequency if the Process ID of the currently scheduled process matches the one of the foreground application. Otherwise it selects the minimum frequency.

To demonstrate the portability of our framework, we decided to execute the fol-

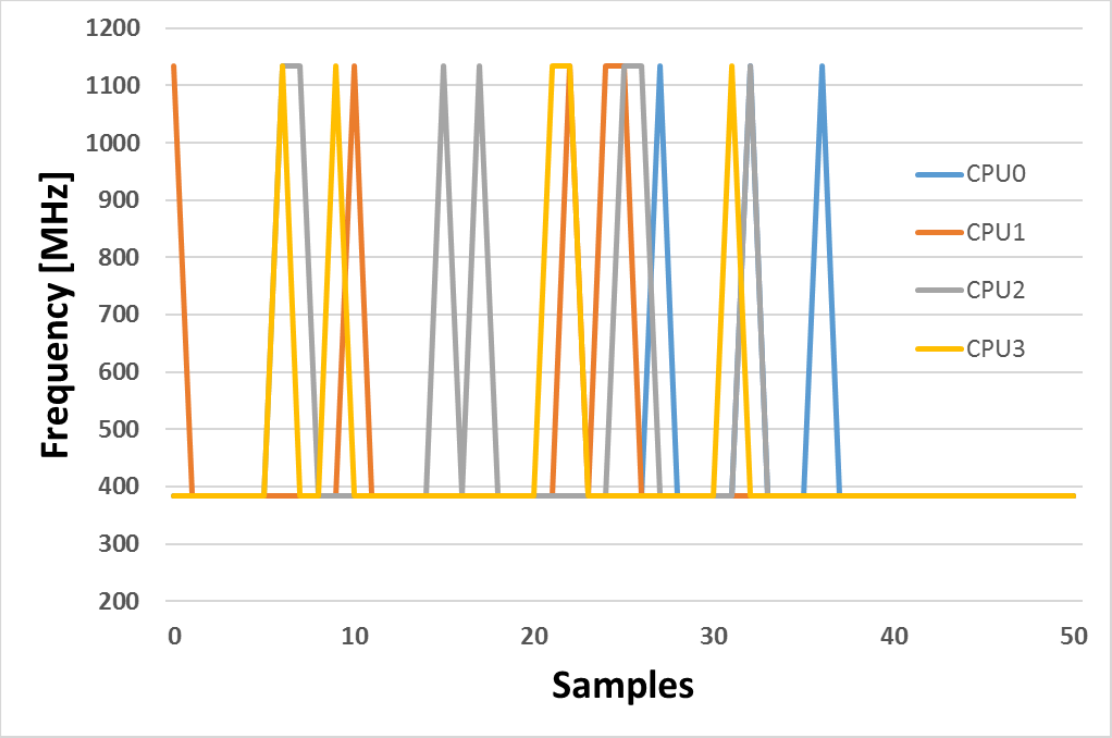

Figure 3.8: Trace of CPU frequencies when using the custom governor

lowing set of experiments on a different platform. We chose the APQ8064 Development Tablet. This device has a quad core Krait CPU with frequency settings from 380 MHz up 1.67 GHz per core. The platform also contains an Adreno 320 GPU. The CPU Cores and the GPU have independent voltage/frequency (V/f) settings and fixed operating V/f points. The operating system of the tablet is Android 4.1.2 (Jelly Bean), with Linux kernel 3.4.0

Figure 3.8 shows the details of the operation of the custom governor. The reported is the operating frequency of the 4 cores of the target platform, together with the upper bound imposed by the user space manager, while executing 5 seconds of the browser application. The frequency is sampled every 100 ms. The frequency is switched to the maximum only if the PID of the scheduled task matches the one of the foreground application (the browser).

To compare the governors in terms of energy efficiency, we first record execution traces of different applications: Angrybirds, Facebook app, Gallery, and the browser. The duration of traces is 2 minutes for Angrybirds and the browser, and 1 minute for

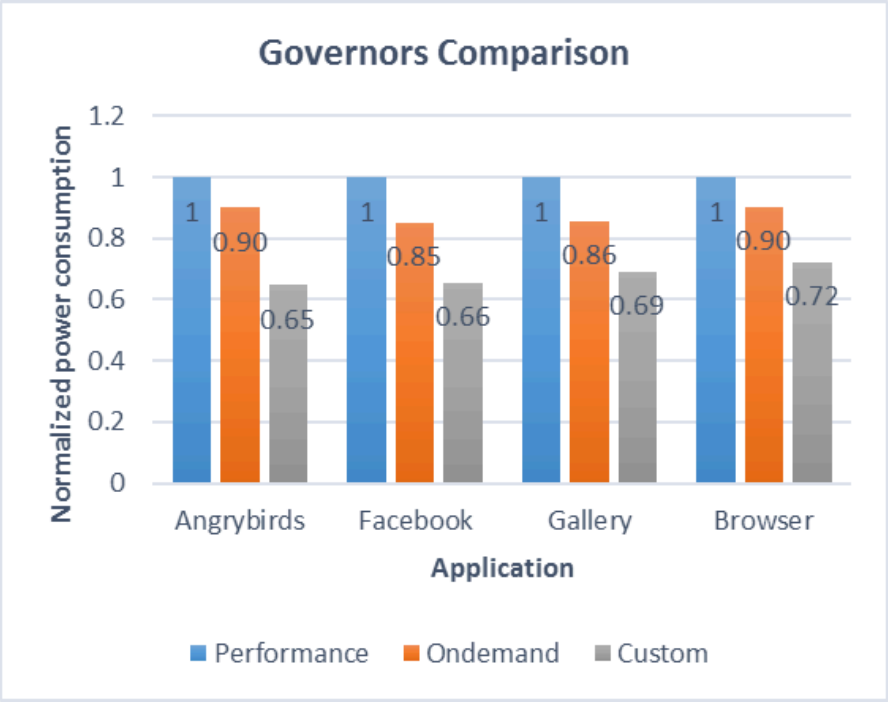

Figure 3.9: Comparison of power savings of different governors with the proposed custom governor

Facebook and Gallery. The recorded traces involve a sequence of interactions such as opening a new page, slide the next picture, open a different tab, scrolling and zooming. Then we replay the trace while keeping the user space manager and the app observer active in all scenarios, and only changing the CPU frequency governor. We use respectively our custom governor, the ondemand governor and the performance governor. The latter essentially sets a constant maximum frequency and it leads to the highest power consumption. For practical reasons, results are normalized with respect to the power consumption obtained with the performance governor. To better capture the difference in power consumption given by governors, in this experiment the mpdecision daemon is deactivated, and all the 4 cores are turned on and configured with the same governor.

Figure 3.9 compares the values of average power consumption for the recorded app traces when applying different governors. The duration of the trace does not change when different governors are used and the average power (which is periodically sampled) gives a comparison in terms of energy as well. The values of power consumption obtained with the performance governor for the traces are 3.87 W, 3.77 W, 2.77 W,

3.37 W, respectively. The result show that the combination of user space manager and custom governor achieves a further 27% of power savings with respect to using the ondemand governor.

3.6 Conclusion

In this chapter we proposed and presented a ready-to-use solution for user-centric proactive power and thermal management. The solution reduces power consumption, thus increasing battery lifetime subject to application-specific performance requirements for user experience. This is a complementary problem with respect to the one solved in Chapter 2, which focused on maximizing performance subject to a target battery lifetime. The approach proposed in this chapter is able to adapt to maximize user experience and reduce power consumption while keeping temperature in a safe range. Temperature prediction is based on a novel observable-based thermal model which achieves a mean accuracy of 1.17^{o}C. The proposed technique has been implemented on a real Android device and tested on a set of real life applications. The experimental results show that our manager achieves up to 46% reduction in application-specific power consumption and up to 35% reduction in average power consumption, when compared to standard power and thermal management. Chapters 2 and 3 focus on the fine-grain time scale characteristic of workload changes, and are part of the short term controller represented in Figure 1.1. In the next chapter we investigate reliability emulation and management in the long term controller, and its interaction with the short term controller.

Chapter 3 contains material from "User-centric joint power and thermal management for smartphones", by Pietro Mercati, Vinay Hanumaiah, Jitandra Kulkarni, Simon Bloch and Tajana Šimunić Rosing, which appears in Proceedings of the 6th International Conference on Mobile Computing, Applications and Services (MobiCASE), 2014, Austin, TX, 2014. [108]. The dissertation author was the primary investigator and author of this paper.

Chapter 4

Reliability Emulation and Management

We presented solutions for power and thermal management which consider user experience in a complementary way. However, they do not consider the impact of temperature on reliability degradation explicitly. With CMOS scaling beyond 14nm, reliability is a major concern for IC manufacturers. Reliability-aware design has a non-negligible overhead and cannot account for user experience in mobile devices. An alternative is Dynamic Reliability Management (DRM), which counteracts degradation by adapting the operating conditions at runtime. In this chapter, we propose a techinique which leverages the major gap between the time scales of workload variations and reliability loss, thus suggesting the multi-rate structure comprised of a long term controller and a short term controller. We formulate DRM as an optimization problem that accounts for reliability, temperature and performance. We develop an optimal policy for multicores using convex optimization, and show that it is not feasible to implement on real systems. For this reason, we propose Workload-Aware Reliability Management (WARM), a fast DRM technique adapting to diverse workload requirements to trade reliability and user experience, implemented and tested on a real Android device. It approximates the solution of the convex solver within 18% in the worst case, while executing more than 400x faster. WARM integrates a Thermal Controller that allocates tasks to meet thermal constraints, since degradation strongly depend on temperature. We show that WARM is within 5% of temperature constraints in 87.5% more cases

than the state-of-the-art. Our task allocation strategy achieves up to 1 year lifetime improvement for a multicore platform and 100% of performance improvement on cluster architectures, such as big.LITTLE, while still guaranteeing the reliability target.

4.1 Introduction

The world of integrated circuits today is facing an era of incredible development. Modern ICs are manufactured in the 14nm technological node [8]. Mobile devices, such as smartphones and tablets, integrate different processing elements (CPUs, DSPs, accelerators, GPUs) and execute applications and programs with varied quality of experience requirements for the user. The stop in supply voltage scaling and the increase in power density have caused the increase of chip temperature and the impact of degradation mechanisms [156].

As the technology scales, the impact of mechanisms such as Time Dependent Dielectric Breakdown (TDDB), Negative Bias Temperature Instability (NBTI) and Hot Carrier Injection (HCI) becomes dramatic [29]. Degradation worsens under voltage and temperature stress and it is influenced by environmental conditions, such as ambient temperature, and workload variations.

Degradation mechanisms are described by the Mean Time To Failure (MTTF) of devices [76], [153]. A common practice is to assume that all the devices of the same kind have the same MTTF. However, with scaling, the mean lifetime of processors becomes shorter and the distribution of lifetimes becomes larger, so it is less and less accurate to assume the same MTTF. The result is the production of devices whose lifetime is more difficult to predict [27]. This has an impact on warranty costs and on trust and reputation of companies. Design techniques cannot completely address these problems, due to workload variability, frequent user interactions and changing environmental conditions. Imposing high design margins results in a loss of performance and higher costs.

The last decade has seen a great development of mobile systems. Modern smartphones and tablets support graphics, wifi communication, web browsing and multimedia, thanks to powerful systems-on-chips, e.g. [6], [77]. They run a great variety of workloads with different performance requirements and they are subject to variable

voltage/frequency stress [159]. Since processor degradation depends on temperature and voltage, a runtime control is needed to correctly manage the reliability of a device over time [58]. Since many degradation mechanisms depend exponentially on temperature, thermal control is a key requirement. Reliability of processors can be estimated by monitoring voltage and temperature, and providing these values to a mathematical model. Alternatively, recent work presents sensors for monitoring degradation [147], and embeds them in prototypes [160], [148].

Dynamic Reliability Management (DRM) is a technique to trade off degradation and performance at runtime, to meet a target lifetime. Reliability is defined at any point in time as a real number between 0 and 1 corresponding to the probability of not having failures. Usually, a reliability threshold is defined a priori. If the estimated reliability at the target lifetime is greater than the threshold, the lifetime constraint is met. Reliability can be modeled as a function of technological parameters, voltage, temperature and time [177]. In DRM, reliability is periodically estimated and per-core operating conditions are controlled to limit the degradation source (i.e. temperature and voltage) [153], [176], [110].

A recent DRM technique uses arrays of local controllers that independently set the voltage of the associated core [110]. The novelty of this approach is to discriminate between highly critical (H) and less critical (L) tasks, depending on their impact on user experience. Each local controller works in two time scales: Long Intervals (LI) and Short Intervals (SI). The Long Term Controller (LTC) monitors the degradation status and calculates average voltage constraints that guarantees the lifetime requirement. The Short Term Controller (STC) changes the voltage and frequency frequently so that (i) the average voltage over a LI is below the constraint and (ii) when high performance are required, the voltage can be boosted for a limited time. This approach only observes temperature, but does not control it, leading to suboptimal decisions. Moreover, the approach only exploits voltage and frequency, but does not leverage task allocation/migration.

All modern operating systems like Linux, Windows and iOS have dedicated components for CPU power management, to target energy efficiency and high performance. Linux uses *governors* to control operating conditions. Governors are kernel

modules, part of the *cpufreq* driver, which is interfaced with hardware regulators enabling voltage and frequency switching [125]. Android, the most popular operating system for mobile devices, based on the Linux kernel, can select between different governors, targeting goals such as providing maximum performance or saving energy. For example, the *powersave* governor forces the processor to run at minimum frequency, while the *performance* governor forces it to run at maximum frequency. A more complex governor, the *ondemand* [125], samples the CPU utilization at a given rate and scales the frequency accordingly. Standard governors are not aware of running applications, and cannot make distinctions based on their priority. They have no DRM capability, but previous work shows that they can implement user experience-aware per-core DRM with low overhead [105]. Android also implements mechanisms to allocate and migrate tasks.

In this chapter for the first time we formulate an optimization problem for temperature and workload-aware reliability management and solve it with a multilevel controller employing convex optimization. It comprehensively manages reliability and temperature subject to diverse performance requirements of applications. We then show that convex solvers are not suited for system requiring response times on the order of milliseconds, because of their computational overhead. For this reason we propose **Workload-Aware Reliability Management** (WARM), a heuristic that efficiently solves the optimization problem by leveraging a cascade of controllers that act on different time scales. The fundamental contribution of WARM with respect to previous work in [110], [105], [176] is the addition of a thermal controller that manages temperature by leveraging task allocation on a multicore platform. This is extremely important, given that most degradation mechanisms affecting reliability have a strong and exponential dependency on temperature. WARM leverages RelDroid, an infrastructure for the online emulation of reliability degradation. RelDroid enables the design of workload-aware dynamic reliability management on real mobile devices with accurate reliability models. Our framework captures the effect of variable workload and environmental conditions and allows to emulate longer degradation in a short time scale. We implement the framework on a real Android device and exploit it to enable workload-aware dynamic reliability management. Our results show that WARM approximates the solution of

the optimal policy within 18% error in the worst case, while executing more than 400x faster. We also show that, since temperature is a major concern for degradation, WARM meets temperature constraints within 5% in 87.5% more cases than the state-of-the-art.

4.2 Related Work

The International Technology Roadmap for Semiconductors (ITRS) [8] recognizes reliability due to aging as a primary concern for integrated circuits. Uncertainties in reliability can lead to performance, cost, and time-to-market penalties and can lead to field failures that are costly to fix and damaging to reputation. In the near future, Time Dependent Dielectric Breakdown (TDDB) and Negative Bias Temperature Instability (NBTI) are a primary concern. TDDB (also referred to as oxide breakdown or dielectric breakdown) is a degradation mechanism that results in a low-impedance path through the gate dielectric of a transistor. Failures related to TDDB are manifest as abnormally high off-state leakage current, changes in circuit switching delay or even failure to switch (hard breakdown). It depends on temperature, voltage and oxide thickness [66], [167]. NBTI results in an increased absolute threshold voltage of p-channel MOSFETs, and hence a degradation in drain current and performance. Most research attributes the threshold shift to two mechanisms. The first mechanism involves interface traps and oxide charge formation due to negative gate bias at elevated temperatures. The second mechanism involves breaking of Si-H bonds at the Si/SiO2 interface by a combination of electric field, temperature and holes. It results in dangling bonds or interface traps at that interface and positive oxide charge that may be due to H^+ [69]. NBTI has been the subject of extensive research [127]. In this work, we focus on TDDB, which is shown to have a much faster degradation rate compared to NBTI [60]. However, the proposed solution can be applied to any degradation mechanism can be modeled with a reliability function.

Traditionally, aging is handled at design time, with the adoption of high design margins under the assumption of worst-case conditions. However, it will not be possible any longer for designer to take into account a worst case design window, because this would jeopardize the performance of circuits too much [8]. A more promising approach

lowers the design margins and exposes degradation at the software stack to manage it at runtime [58].

Runtime control of operating conditions and task allocation have been successfully used in the recent years for Dynamic Power Management (DPM) and Dynamic Thermal Management (DTM). The latter is a technique aiming at keeping the operating temperature of the chip in a safe range. Researchers developed different techniques from simple reactive ones, to more elaborated approaches like model predictive control [150], [21]. Today DTM is employed in almost all commercial devices and can be handled by all operating systems.

Dynamic Reliability Management (DRM), instead, is a technique where a processor can dynamically trade performance to adapt aging. DRM is different from DTM in that it allows to meet a target lifetime. This is introduced in [153], where a microarchitectural reliability model is the reference for dynamic adaptation. The authors report how DTM and DRM lead to different control behaviors. Work in [78] highlights the limitations of [153] of focusing on a short time scale and of using benchmark applications for experiments. In turn, it proposes a PID-based DRM algorithm which exploits DVFS for peak performance improvement under high demand. The DRM here exploits a linear extrapolation to project lifetime total damage. They evaluate with macro-level user-collected processor usage profiles. This work assumes that the future workload is equal to the previous one and is sensitive to sudden workload variations. Zhuo et al. [177] proposed a process variation and temperature-aware reliability model for TDDB, which can estimate processor reliability from temperature and voltage history. In [176] the authors presented a DRM framework that extends this model to periodically predict the future value of reliability at the target lifetime. Based on the difference between the predicted reliability and the target one, the controller sets a maximum operating voltage. This approach is less sensitive to workload variations compared to previous work because it exploits a confidence-based workload estimation. Since the policy sets the maximum voltage, it is not able to guarantee speed bursts for high performance demanding tasks, causing user experience degradation. Recent work in [152] explores the tradeoff between performance and reliability in multicore processors by introducing the throughput-lifetime product (TLP) and proposes dynamic reliability variance manage-

ment (DVRM) to enhance multicore lifetime. The approach in [119] employs an efficient Bayesan classifier to detect reliable configurations online and uses architectural adaptation to select the one with best performance thanks to a performance prediction model. These papers present results deriving from simulations. For this reason they assume workload and platform models. Workload models are known a priori, but they may not be available for general purpose architecture. This limits the feasibility of implementation of the proposed approaches on a real platform. DRM is not available in commercial devices today.

Recent work proposes hardware degradation monitors. In [24] the authors propose the design of a wear out detection unit for automatic compensation. Reference [147] presents in-situ sensors for NBTI and TDDB. These devices are based on a ring oscillator driven by transistors under stress. The frequency of the ring oscillator varies depending on the impact of degradation. Work in [148] uses the output of these sensors in a control loop for managing dynamically the impact of NBTI degradation. The authors of [164] present Radic, a novel built-in sensor for reliability analysis. based on this, they deploy an aging adaption system to prevent failures. These publications effectively counteract degradation, but they do not consider the existence of different workload requirements to achieve good user experience. This is achieved by the technique in [110], which is a workload-aware DRM technique for multiprocessors based on a two-level controller. This technique monitors system reliability on a long time scale and adapts operating conditions to workload quality requirements on a short time scale, preserving from user experience degradation. This is shown to outperform state-of-the-art techniques, as it provides full performance to critical applications. This work focuses on TDDB, exploiting the model presented in [176] for simulating the presence of degradation sensors. It does not assume a priori knowledge of workload, but leverages a runtime binary characterization of workload requirements. Thus it is feasible to implement.

A number of recent publications propose reliability-aware scheduling techniques. Work in [67] proposes a task allocation technique for MPSoC which maximizes lifetime subject to performance constraints. The authors of [25], instead, focus on NOC-based platform and devise a scheduling algorithm for joint energy and reliability optimization. In these publications, however, lifetime is considered as an objective function and max-

imized, rather than as a constraint. Work in [127] formulates and solves the problem of energy efficient, aging-tolerant task allocation for a variability affected platform running rate-constrained multimedia applications, adapting to system degradation conditions. In this work, reliability is a constraint rather than an objective function. Such a formulation is more meaningful from a manufacturer perspective [8]. These publications do not exploit DVFS and the existing power management infrastructure. Also, the scheduler of operating systems such as Linux is a very critical section. The implementation of scheduling techniques is highly challenging in real devices and the results presented in the referenced works come from simulations.

Alternative DVFS and task scheduling approaches have been proposed to mitigate thermal hotspots, also improving the system reliability. In reference [138], the thermal behavior of a multicore processor is first modeled through a state-space system, which accounts for the thermal coupling between cores. Next, the model is reduced into a representation suitable to formulate an Integer Linear Problem (ILP). The ILP's goal is to allocate N tasks onto N cores while maximizing the per-core frequency and meeting a thermal constraint. A similar approach is proposed in [61]. In this paper, an heterogeneous multicore processor running a queue containing N non-identical tasks is considered. The authors use a more detailed thermal model which leads to a non-linear optimization problem. These publications are effective in controlling temperature, but are not aware of degradation, which may lead to either reliability violations and performance degradation.

Another class of related work focuses on management for soft errors (for examples, bit-flips caused by radiations). Such errors can compromise data integrity and accuracy of computations, but they do not affect aging. Work in [174] investigates the effects of energy management using DVFS on real-time embedded systems. The authors find that, for critical applications, DVFS may increase the fault rate dramatically. Based on this, the authors of reference [173] propose two effective scheduling algorithms for real-time tasks that maintain a very low fault rate, without impacting on energy efficiency. Our work is orthogonal, as we target mobile devices and degradation mechanisms that induce circuit aging.

Reliability management for embedded systems is the subject of very recent

work. Reference [43] presents a hierarchical manager that exploits Q-learning to jointly control temperature and energy consumption. The proposed strategy is also demonstrated to improve the lifetime reliability. Work in [42] proposes a simplified temperature model to develop a gradient-based efficient heuristic to determine multicore operating conditions with the goal of minimizing energy consumption and maximizing the system lifetime. The presented strategies are effective in improving system reliability, but they cannot account for user-experience.

Previous work in [110] proposed a two time-scale DRM framework and a following publication described its implementation as a Reliability Governor on a real Android device [105]. These techniques have three main limitations, which may lead to suboptimal decisions. First they do not control temperature, but only voltage and frequency. This is very critical because degradation mechanisms have an exponential dependency on temperature. Second, they scale voltage and frequency, but cannot leverage task allocation/migration. This is an effective strategy to balance temperature in a multicore. Third, they have not been compared to an optimal approach.

In previous chapters, reliability is not considered. In this chapter, for the first time we formulate and solve an optimization problem that accounts for reliability and thermal constraints, while meeting application-specific performance requirements for quality of experience (QoE). We then present WARM, a fast hierarchical heuristic that approximates the optimal solution, and implement it in the Android software stack. We add a thermal controller to WARM that uses task allocation and migration to effectively balance degradation among different cores and achieve higher performance. In the Android implementation, we decouple reliability emulation from reliability control. We implement an application manager that enables taking short term decisions based on the currently scheduled task. Finally, in experimental results, we compare our technique on a real Android device against state-of-the-art governors on a broad range of applications. We achieve up to 1 year reliability lifetime improvement for a multicore platform, and up to 100% of performance improvement on cluster architectures.

4.3 Mathematical Models

The main degradation mechanisms affecting integrated circuits are Time Dependent Dielectric Breakdown (TDDB), Negative Bias Temperature Instability (NBTI), Hot Carrier Injection (HCI), Electromigration (EM) and Thermal Cycling (TC). The average time before the failure of a device is denoted as Mean Time To Failure (MTTF). Models have been developed for MTTF for each degradation phenomenon, which show a strong dependence on temperature. For example, the MTTF for TDDB is described by Equation (4.1):

$$MTTF_{TDDB} = A_0 \exp -\gamma E_{ox} \exp \frac{E_a}{kT} \tag{4.1}$$

Where A_0 is a constant determined empirically, E_{ox} is the electric field across the dielectric, γ is the field acceleration parameter and E_a is the activation energy. The MTTF for NBTI is described by Equation (4.2):

$$MTTF_{NBTI} = A_0 \left(\frac{1}{V}\right)^{\gamma} \exp \frac{E_a}{kT} \tag{4.2}$$

Where γ is the voltage acceleration factor and V is the applied voltage. The MTTF for HCI is described by the Eyting model, expressed in Equation (4.3) for N-channel devices:

$$MTTF_{HCI} = BI_{sub}^{-N} \exp \frac{E_a}{kT} \tag{4.3}$$

Where I_{sub} is the peak substrate current during stressing, N is a material dependent constant and B is a scale factor, function of technological parameters. Temperature is a parameter in all the previous equations and the dependency of MTTF on temperature is exponential, so it is very critical for degradation.

MTTF for each degradation mechanism is related to a reliability function as expressed by Equation (4.4). Compared to MTTF, reliability is a function of time, so it is more suited for the purpose of dynamic management [176]. When considering the effect of multiple mechanisms acting together, multiple reliability functions can be combined into a single one [153]. In this work we focus on TDDB, as it is considered

to be one of the key sources of degradation [8]. We describe the reference reliability model for TDDB [177] that is used in our implementation.

$$MTTF = \int_0^\infty R(t) \tag{4.4}$$

4.3.1 TDDB Reliability Model

Time Dependent Dielectric Breakdown (TDDB) is a major concern for modern integrated circuits, given the important dependence it has on gate oxide thickness [8]. As technology scales, the gate oxide layer is thinner, increasing the risk of breakdown and shortening devices lifetime. Because of its non reversibility and increasing impact, it is a very representative degradation phenomenon. The oxide breakdown time to failure is inherently a statistically distributed quantity. TDDB time is modeled as a random variable with a Weibull probability distribution function. The reliability of a single transistor subject to oxide degradation can be expressed as [154]:

$$R(t) = e^{-a\left(\frac{t}{\alpha}\right)^{\beta}} \tag{4.5}$$

where t is the time-to-breakdown, a is the device normalized area with respect to the minimum area, and α and β are respectively the *shape parameter* and *scale parameter* of the Weibull distribution (sometimes also referred to as *Weibull slope*). The scale parameter α represents the characteristic life, which is the time where 63.2% of devices fail, and it depends on voltage and temperature. The shape parameter β, instead, is a function of the critical defect density, which in turn depends on oxide thickness, temperature and applied voltage. In [44] the shape parameter is shown to vary linearly with the oxide thickness x, so $R(t)$ can be expressed as follows.

$$R(t) = e^{-a\left(\frac{t}{\alpha}\right)^{xb}} \tag{4.6}$$

In the remaining of this chapter, for simplicity, we will refer to b as shape parameter. It is constant for a given oxide thickness, and depends only on voltage and temperature. Due to process variations, the oxide thickness is actually a distributed

quantity. Therefore, the single i^{th} device reliability can be expressed as a conditional probability:

$$R(t|x_i) = e^{-a\left(\frac{t}{\alpha}\right)^{x_i b}} \tag{4.7}$$

Based on this, the reliability of the entire chip R_c can be then expressed as the product of all of the single device reliabilities as:

$$R_c(t|\mathbf{x}) = \prod_{i=1}^{m} R(t|x_i) = e^{\sum_{i=1}^{m} -a_i\left(\frac{t}{\alpha_i}\right)^{b_i x_i}} \tag{4.8}$$

where $\mathbf{x}$ is the vector of all oxide thicknesses and m is the number of transistor of the chip. For eliminating the dependence on $\mathbf{x}$, a m-dimensional integral would be required. This problem has high complexity, as m could be millions. The work in [177] proposes a way to reduce the complexity of this problem, while still including process variation effects on oxide thickness. It is based on the observation that different regions of the chip share similar temperatures. A *block* is defined as region of the chip with almost the same temperature. Given this definition, the reliability functions of the single devices belonging to the same block have the same scale and shape parameters. Given that N is the total number of blocks, $R_c(t|\mathbf{x})$ can be rewritten as:

$$R_c(t|\mathbf{x}) = e^{\sum_{j=1}^{N}\sum_{i=1}^{m_j} -a_{i,j}\left(\frac{t}{\alpha_j}\right)^{b_j x_{i,j}}} \tag{4.9}$$

A further simplification is introduced by defining the *Block Level Oxide thickness Distribution* (BLOD). Collecting all the oxide thicknesses of all the devices in a block, it is possible to build a frequency histogram. The histogram, then, can be fitted to a Gaussian curve, which is the BLOD. Each BLOD has mean u_i and variance v_i. It can be noticed that, since it is unfeasible to actually measure oxide thicknesses and build a frequency histogram, means and variances of BLODs are random variables. However, the number of random variable in the problem is reduced from m (millions) to $2N$ (some units). The chip reliability can be approximated as:

$$R_c(t|\mathbf{x}) = R_c(t|\mathbf{u}, \mathbf{v}) \tag{4.10}$$

The simplification steps described in [177] allow to remove the dependence and express the chip reliability as a sum of double integrals in the space of means and variances of

BLODs, as:

$$\begin{aligned} R_c(t) =& 1 - N + \\ & \sum_{j=1}^{N} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} e^{-A_j g(u_j,v_j)} f_{u_j,v_j}(u_j, v_j) du_j dv_j \end{aligned} \tag{4.11}$$

where N is the number of blocks composing the chip, A_j is the normalized area with respect to the minimum of the j^{th} block, f_{u_j,v_j} is the joint distribution of means and variances of BLODs and $g(u_j, v_j)$ results from the simplification procedure and reads:

$$g(u_j, v_j) = exp\left(ln(\frac{t}{\alpha_j}) b_j u_j + (ln(\frac{t}{\alpha_j}))^2 b_j^2 v_j / 2 \right) \tag{4.12}$$

The joint distribution f_{u_j,v_j} can be expressed as the product $f_{u_j,v_j} = f_{u_j} \times f_{v_j}$ with good approximation, as shown in [177]. The distribution f_u is Gaussian, while for the variances we have:

$$v_j \sim v_{j,0} + \hat{a}\chi_{\hat{b}} \tag{4.13}$$

where χ is a chisquare distribution with $\hat{b}$ degrees.

Since the quantity R_c is static (meaning that, for how it is formulated, it considers that the same voltage and temperature are applied from time $t = 0$). To exploit the value of reliability in a control loop, it is necessary to consider temperature and voltage changes over time. This is possible by discretizing the time axis and calculating at each time step the system (dynamic) reliability as:

$$\begin{aligned} R_k =& R_{k-1} - \\ & (R_c(t_{k-1}, T_{k-1,k}, V_{k-1,k}) - R_c(t_k, T_{k-1,k}, V_{k-1,k})) \end{aligned} \tag{4.14}$$

where k indicates the generic k^{th} time instant, $T_{k-1,k}$ and $V_{k-1,k}$ are the temperature and voltage experienced by the system between the time instants $k-1$ and k, R_c is the static reliability. In this way, the system reliability is calculated as a recursive sum of progressive damages.

When applying this model, we consider each core of the multicore platform as a single block, thus $N = 1$. Equation (4.11) reduces to:

$$R_c(t) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} e^{-A_j g(u_j,v_j)} f_{u_j,v_j}(u_j, v_j) du_j dv_j \tag{4.15}$$

Equation (4.14) is used in our framework as a black box to asses the reliability status of the processors, given temperature and voltage values, using 4.15 to evaluate the static reliability R_c. The value R_k resulting from Equation (4.14) goes from 1 to 0 and is considered as a measure of the system degradation status. Equation (4.14), used to update the reliability value, is general and does not depend on a specific degradation mechanism. Therefore, our framework is valid for every degradation mechanism or combination of multiple mechanisms as long as it can be described by a function $R_c(t)$ such as that in Equation (4.15). For example, to extend the framework to include NBTI and HCI, we assume that we have models to describe the reliability functions associated with these mechanisms, respectively R_{NBTI} and R_{HCI}. As described in references [78], [79], the total system reliability function is given by the product of the functions associated with the single mechanisms as $R_c(t) = R_{TDDB}(t) \cdot R_{NBTI}(t) \cdot R_{HCI}(t)$.

4.3.2 Thermal Model

The heat propagation across a multicore processor is modeled using the heat diffusion relationship reported in Equation (4.16). In this equation, $T(\vec{r}, t)$ and $P(\vec{r}, t)$ are the temperature and power at location $r = (x, y, z)$ and time t. Parameter ρ is the material density, c_p is the material specific heat, and k_T is the material thermal conductivity [22]. The heat propagation across a multicore processor is modeled using the heat diffusion relationship reported in Equation (4.16). In this equation, $T(\vec{r}, t)$ and $P(\vec{r}, t)$ are the temperature and power at location $r = (x, y, z)$ and time t. Parameter ρ is the material density, c_p is the material specific heat, and k_T is the material thermal conductivity [22].

$$\rho(\vec{r})c_p(\vec{r}, t)\frac{\delta T(\vec{r}, t)}{\delta t} = \Delta \cdot [k_T(\vec{r}, t)\Delta T(\vec{r}, t)] + P(\vec{r}, t) \tag{4.16}$$

Equation (4.16) can be spatially and temporally discretized. Let $T[k] \in \Re^N$ be the temperature at instant k for N locations and $P[k] \in \Re^M$ be the known powers of M sources (e.g. the power consumed by each of the cores). Let also $T_s[k] \in \Re^S$ be S observable temperatures (i.e., the temperatures measured via sensors), and A, B, C transformation matrices. Then, the equivalent discrete-time state-space linear time-invariant (LTI) system can be expressed as in Equations (4.18) and (4.17) [22].

$$T[k+1] = AT[k] + BP[k] \tag{4.17}$$

$$T_s[k] = CT[k] \tag{4.18}$$

The representation of Equations (4.17) and (4.18) is useful to deploy effective control strategies aimed at meeting a thermal constraint while improving performance and reducing the power consumption by operating DVFS and task scheduling [21].

4.3.3 Power Model

For a core, the power is the sum of two contributions: dynamic and leakage. The dynamic power can be modeled through Equation (4.19) where α an C are the activity factor and the switching capacitance. As the frequency f depends linearly on V_{dd}, the dynamic power can be approximated as a cubic function of frequency, similarly as in [149].

$$P_{dyn} = \alpha C V_{dd}^2 f \simeq a f^3 \tag{4.19}$$

The leakage power can be modeled through Equation (4.20) where the coefficient b is a technology dependent constants, channel length and width; the coefficient k is the Boltzmann constant, the electron charge, and the threshold voltage; and I_{gate} is the gate leakage current that can be assumed constant.

$$P_{lkg} = V_{dd}(bT^2 \exp(\frac{k}{T}) + I_{gate}) \tag{4.20}$$

A simplified model, that accounts for both dynamic and static power is shown in Equation (4.21). The strength of this model is that it can be easily fit to real measurements.

$$P = P_{dyn} + P_{leak} = a \cdot f^3 + b \cdot f \tag{4.21}$$

4.4 Optimal Controller Architecture

In this section, we describe the structure of the optimal controller. We divide the problem into Long Intervals (LI), in the order of days it takes for reliability to change, and Short Intervals (SI), in the order of milliseconds for scheduling decisions.

Figure 4.1 shows the structure of an optimal dynamic reliability controller. It consists of an Optimal Long Term Controller (OLTC), which selects the target average voltage and temperature for each core for the next LI, and an Optimal Short Term Controller (OSTC), which determines task allocation and frequency levels at each SI. The

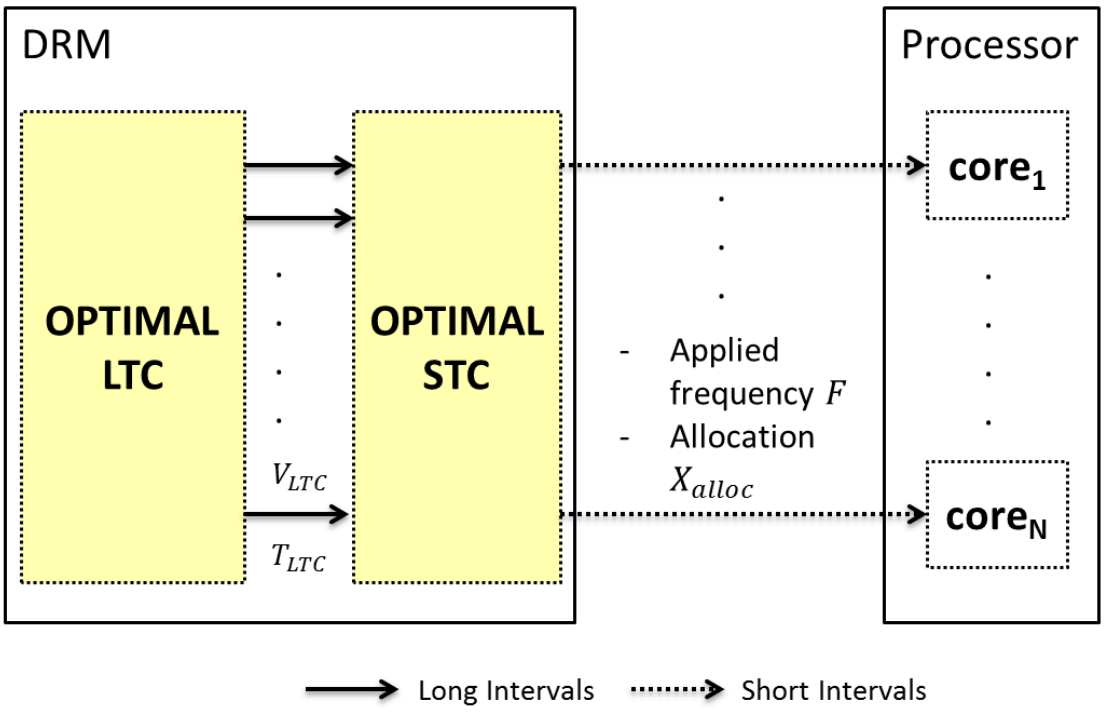

Figure 4.1: Block diagram of optimal DRM controller

OLTC activates at the beginning of a LI and provides each core with a voltage and a temperature reference values, T_{LTC} and V_{LTC}. Then, the OSTC activates at each SI and determines voltage/frequency levels and task allocation so that the average temperature and voltage in the LI are respectively lower than T_{LTC} and V_{LTC}, The OLTC problem for finding T_{LTC} and V_{LTC} is formulated as follows.

$$\min_{T_{LTC},V_{LTC}} \quad \|R_p(t_{life})\| \tag{4.22}$$

$$\text{s.t.} \quad R_p(t_{life}) \geq R_{target} \tag{4.23}$$

In this problem, $R_p(t_{life})$ indicates the predicted reliability at the target lifetime [110]. Based on such formulation, the OLTC finds the pair of values T_{LTC} and V_{LTC} which are used as constraints in the OSTC problem. The constraint on reliability is met if, at the end of each LI, the average observed temperature and voltage are lower than constraints (T_{LTC}, V_{LTC}) [110].

For the formulation of the OSTC problem, we assume that the target device has N cores, each one executing a task j ($j = 1, \ldots, N$) that requires a frequency $f_j^*[k]$ at the short interval k. For each core i, its average frequency over a LI, labeled as f_{LI_i}, must not be greater than the value f_{LTC_i}. This corresponds to the maximum frequency that is possible to obtain at voltage V_{LTC_i}. This is a reasonable assumption given that processors supporting DVFS usually have predefined voltage-frequency operating points. Moreover, its average temperature over a LI should not be greater than the reference T_{LTC_i}.

We also assume such a system to work alongside the native system scheduler, which selects at most N tasks. The problem is then to allocate N tasks onto N cores and select the frequency $f_i[k]$ of the core i at each SI k. If the actual tasks are less than the cores, we consider a number of idle tasks to sum up to N. Similarly as in [110], some tasks are labeled as highly critical (H) for user experience, thus they require to execute at a frequency as close as possible to $f_j^*[k]$. These can be, for example, tasks belonging to the foreground application. All others are labeled as less critical (L). Given these assumptions, at each SI, the following optimization problem is formulated.

$$\min_{F,X_{alloc}} \quad \|F - X_{alloc} \cdot F^*\|^2 \tag{4.24}$$

$$\text{s.t.} \quad F[k] \leq F_{ref_{HH}} \tag{4.25}$$

$$T[k] \leq \bar{T}_{ref_{HH}} \text{ for } k = k_0, ..., k_0 + L \tag{4.26}$$

$$T[k] \leq T_c \tag{4.27}$$

In the problem above, $F[k]$ is the vector of core frequencies $F[k] = [f_1[k], \ldots f_N[k]]'$ at time instant k, where $k = k_0, ..., k_0 + L$. F^* is the vector of required frequencies

$F^*[k] = [f_1^*[k], \dots f_N^*[k]]'$ at instant k. X_{alloc} is a matrix which elements x_{ij} assume value 1 if core i executes task j and 0 otherwise. The matrix $X_{alloc} \in \mathbb{R}^{N \times N}$ of elements $x_{i,j}$ has only one element equal to 1 on each column and row. This is because every core executes one task. The solution to the problem is given by matrix X_{alloc} and vector F. The term $F_{ref_{HH}}$ is a vector with the dimension equal to the number of cores, in which the generic element $f_{ref_{HH}}$ is determined as in Equation (4.28).

$$f_{ref_{HH}} = \begin{cases} f_{ref} & \text{if task is } L \\ f_{MAX} & \text{if task is } H \end{cases} \tag{4.28}$$

Similarly, the term $\bar{T}_{ref_{HH}}$ is a vector with the dimension equal to the number of cores, in which the generic element $T_{ref_{HH}}$ is calculated as in Equation (4.29).

$$T_{ref_{HH}} = \begin{cases} T_{ref} & \text{if task is } L \\ T_c & \text{if task is } H \end{cases} \tag{4.29}$$

In the above equation, the values of f_{ref} and T_{ref} are updated at each SI, based on the rules shown in Equations (4.34) and (4.31) respectively. The goal of Equations (4.28) and (4.29) is to relax the constraints of the optimization problem for H tasks, so that the solver can find a solution that provides a higher frequency for H tasks. We use the thermal model of Equations (4.17) and (4.18). We also assume to have a power and thermal sensor on each core. Therefore, $A, B, C \in \mathbb{R}^{N \times N}$ while $C = I$.

The problem has real (i.e. F) and binary (i.e. X_{alloc}) decision variables. For a fixed X_{alloc}, the problem is convex, because frequency and power can only have positive values. The problem is solved for all the possible instances of X_{alloc} to choose the one that provides the best solution.

As illustrated in [30], an optimization problem belongs to the class of convex problems if the objective function and the constraint functions are convex. To prove this, we assume $X_{alloc} = X'$ is fixed. First of all, we prove that the objective function $f_0(F) = |F - X'F^*|^2$ is convex. This is clearly true, as the function is a square. Then, constraint in Equations (4.25) is linear, which is a subclass of convex functions. Constraints in (4.26) and (4.27) are also convex with respect to frequency. This can be verified by substituting Equations (4.19) and (4.20) and considering that frequency is always greater than zero.

To find the solution, the OSTC iterates through each possible X_{alloc} and computes the optimal F. Then, it returns the pair of X_{alloc} and F that provides the lowest value for $\|F - X_{alloc} \cdot F^*\|^2$.

4.5 WARM Controller Architecture

Convex solvers are too computational expensive to be used in runtime management policies, since the system should take control decisions in the order of milliseconds. For this reason, we develop WARM, a heuristic solution that approximates the optimal solution, but is more than 400x faster. WARM consists of three components:

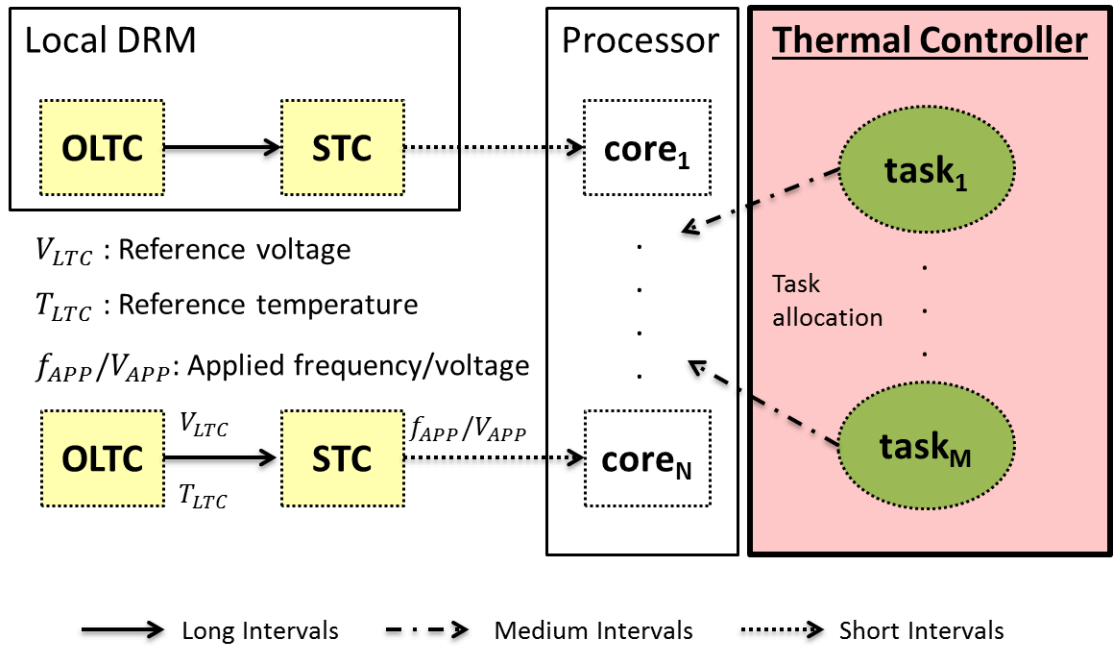

Figure 4.2: WARM block diagram

The first is a set of local and independent **Short Term Controllers (STCs)** that find F and switch the operating frequency of the cores at each SI. The second is a **Thermal Controller (TC)**, which determines the allocation of tasks for a Medium Interval (MI), sub-second or a few seconds. This is a key component, since degradation depends exponentially on temperature. On the top of these two actions, a set of **Optimal Long Term Controllers (OLTCs)** based on convex optimization estimates the degradation status of

the cores and sets the operating condition constraints for a LI (i.e., T_{LTC} and V_{LTC}). In the following, we explain the behavior of each component more into details.

4.5.1 Long Term Controller

The Optimal Long Term Controller (OLTC) activates at each Long Interval, samples data from aging sensors, monitors the degradation status, and calculates the average temperature and voltage. The OLTC predicts future reliability with these values, using the technique described in [110]. This is determined by assuming a predicted constant voltage and temperature for the remaining lifetime. Since reliability loss occurs on a long time scale, we consider a LI to be on the order of days. Based on this, it solves the problem presented in Equation (4.22) using convex optimization and provides a reference voltage/frequency V/f_{LTC} and a reference temperature T_{LTC}, which are the inputs for STC. The constraint on reliability is met if the average applied voltage V_{LI} is less or equal to V_{LTC} and the average temperature T_{LI} is below T_{LTC}, for each core, at the end of the LI. Since the OLTC activates in the order of days, the use of convex optimization represent a negligible overhead.

4.5.2 Thermal Controller

Previous work in [110], [105] is based on only a long term controller and a short term controller, and it has two main limitations. First, it cannot balance temperature across different cores of a multicore platform, because it cannot exploit task allocation. Second, it cannot enforce the constraint on T_{LTC}, because it does not control temperature directly. To solve these problems, WARM has a centralized Thermal Controller (TC) that monitors core temperatures and updates the values of average temperature T_{avg} and reference temperature T_{ref} with Equations (4.30) and (4.31).

$$T_{avg}(i) = \frac{(T_{avg}(i-1) * (i-1) + T(i))}{i} \tag{4.30}$$

$$T_{ref}(i) = min\left(\frac{(T_{LTC} * t_{LI} - T_{avg} * i)}{(t_{LI} - i)}, T_c\right) \tag{4.31}$$

In these equations, i indicates the i^{th} SI inside a LI, $T(i)$ is the temperature at the i^{th} SI, t_{LI} is the duration of a LI (measured in SIs), T_{LTC} is the reliability-induced

constraint on average temperature and T_c is the core critical temperature. Given this, the TC takes decisions at a coarser time granularity. The activation periods for the TC are called Medium Intervals (MI) and are in the order of tens of SI, representative of temperature changes over time. When the TC activates, it determines the task allocation by assigning the tasks with higher priority (e.g. H tasks) to cooler cores. This is equivalent to determining the matrix X_{alloc}. Then, it forces the frequency f_L to assign to L tasks for the next MI to the minimum, in case the current temperature T gets higher than T_{ref}. This helps reducing the thermal stress for the next MI, thus reducing the value of T_{avg} for the current LI. In this way, the TC limits the performance of L tasks to execute only if the temperature is out of a safe range from the point of view of reliability (e.g. relative to T_{ref}). By limiting the performance of L tasks, the TC spends a MI to lower the average temperature.

4.5.3 Short Term Controller

The Short Term Controller (STC) activates at each Short Interval (SI) and selects the voltage to apply at a fast time rate. A SI ideally corresponds to the scheduling tick of a real system. As already described, the frequency to apply for the execution of L tasks is selected at each medium interval following the rule specified in Equation 4.33. Then, the STC selects the applied frequency as described by Equation (4.32). In this equation, the value of f_L is selected by the rule in Equation (4.33).

$$f_{app} = \begin{cases} f_L & \text{if task is } L \\ f_H & \text{if task is } H \end{cases} \tag{4.32}$$

$$f_L = \begin{cases} f_{MIN} & \text{if } T > T_{ref} \\ min(f_{ref}, f_{req}) & \text{otherwise} \end{cases} \tag{4.33}$$

Here, the first case with f_{MIN} is enforced by the TC. For the second case, f_{req} is the required execution frequency of the task, while f_{ref} is updated at each SI with the rule specified in Equation (4.34). Moreover, the value of f_H corresponds to f_{req} for the current task. Finally, f_{ref} is computed as in Equation (4.34).

$$f_{ref}(i) = min\left(\frac{(f_{LTC} * t_{LI} - f_{avg} * i)}{(t_{LI} - i)}, T_c\right) \tag{4.34}$$

Where f_{LTC} represents the reliability-induced constraint on frequency f_{avg} is updated at each SI similarly as T_{avg}. If the system is running into a power-critical scenario that requires the intervention of power management, the applied frequency f_{app} can be lowered, so the target reliability would still be met. If the task executed is H, this would occur at the cost of QoE degradation, but in any case the proposed DRM would not exacerbate the power consumption.

In this work we assume without loss of generality that applications are either H or L. However, the controller can operate correctly even if quality requirements change at a fast rate even for the same application, as long as it is slower the STC activation rate. For example, an H application could have an L phase (like the menu screen of a mobile game). This could be identified online by integrating the DRM with the app-phase recognition engine proposed in [87] to enable further reliability improvement.

4.6 Android Implementation

All the components of WARM have been implemented in the Android software stack to execute on real devices. Figure 4.3 shows the block diagram of the proposed implementation. WARM consists of three subcomponents: the Application Monitor, RelDroid and the WARM reliability manager. In the following we describe each component into details.

Application Monitor: The Application Monitor consists of the configuration file (App Monitor Config in the figure), the App Monitor and the App Driver. The User can optionally fill a list of favorite applications, which is saved into the configuration file. This is because not all the applications may be critical for the user, due to subjective judgment. The Monitor periodically checks the applications currently active in foreground, and if it finds a matching in the list of favorite applications, it outputs the ID of the corresponding process. This is passed to the kernel space through the driver, and it is stored in a shared variable.

RelDroid: RelDroid implements the infrastructure for monitoring reliability.

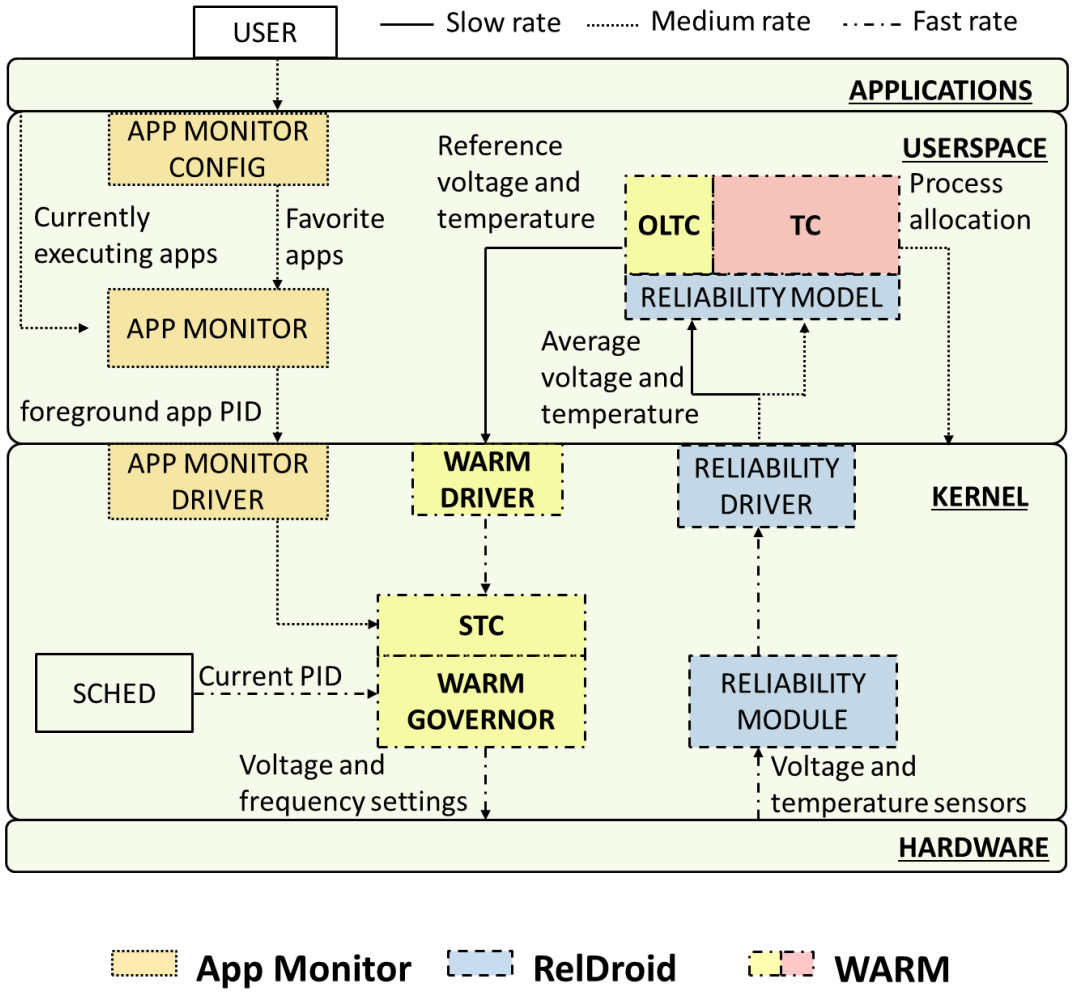

Figure 4.3: WARM Implementation

Since commercial devices do not have degradation sensors, reliability needs to be emulated online through temperature and voltage readings, which are the input for the model discussed in Section 4.3. In the kernel space, the Reliability Module samples the values of voltage and temperature at each scheduling tick and updates the average values. In the userspace, the Reliability Model activates periodically (at a user-defined rate) and reads the average voltage and temperature values from the module through a dedicated Reliability Driver. Finally, the Reliability Model implements the model presented in Section 4.3 to update the values of reliability and writes the values to a log file.

WARM: WARM consists of the **Optimal Long Term Controller** (OLTC), the

Thermal Controller (TC) and the **Short Term Controller** (STC). The LTC selects the reference average voltage and temperature for the next Long Interval (LI) with a PID based control. The TC activates at each Medium Interval (MI) and switches the allocation of active processes on cores to balance temperature. This is accomplished by the *set_affinity* mechanism, which automatically increases the affinity of tasks to run on a specific core. Both the LTC and the TC are implemented in C language and cross compiled with Android NDK toolchain to run on the target ARM architecture. Finally, the Short Term Controller (STC) is implemented as a *cpufreq* governor, called WARM Governor. To implement this, we modified the code of the *ondemand* governor and replaced the original algorithm with the rule described in Section 4.5.3. The kernel and userspace components of WARM communicate with each other through the WARM driver.

4.7 Experimental Setup

Our Android test platform is the Odroid XU3 development board, shown in picture 4.4. This board has a Samsung Exynos 5422 Octa core based on ARM big.LITTLE architecture, with a Cortex-A15 2.0Ghz quad core cluster and Cortex-A7 quad core cluster. It also has a Mali-T628 MP6 GPU, supporting OpenGL ES 3.0/2.0/1.1 and OpenCL 1.1 Full profile. The platform has a 2MB L2 cache and a 2GB LPDDR3 RAM at 933MHz (14.9GB/s memory bandwidth), PoP stacked. The OS is Android 4.4.4 with Linux kernel 3.10.9.

The device has 4 integrated voltage/current/power monitoring sensors implemented on the PCB and driver support. They allow to measure the voltage, current and power respectively of LITTLE cluster, big cluster, GPU and memory subsystem. The LITTLE and big cluster have DVFS capability and can switch voltage and frequency using the *cpufreq* utility between predefined operating points. Figure 4.5 shows the voltage-frequency curves for the two clusters. This has been obtained by changing the frequency of the cores and sampling the integrated voltage sensor. The platform has 4 temperature sensors for the big cluster and 1 for the GPU, which exact placement is unknown, but no sensor for the LITTLE cluster. We perform an experiment to associate

Figure 4.4: Odroid XU3 board

a sensor to each big core. To do this, we selectively run a power virus application with one big core active at a time for 5 seconds, and observe which sensor records the highest temperature. We also implement a virtual platform leveraging thermal and power models to simulate the control policies. This is required due to the complexity of convex solvers. Figure 4.6 shows the block diagram of the virtual platform implementing a control loop. The trace of required frequency F^* is randomly generated and provided as an input. These are used to compute the power consumption of each core with the derived power model. The power is then used in the state-space model to predict the temperature of each core. Finally, the control policy adapts the frequencies to meet the set of constraint of the management problem.

To simulate temperature, we employ the state space model used in [130]. For power, we derive the model by fitting the relation in Equation (4.21) with real exper-

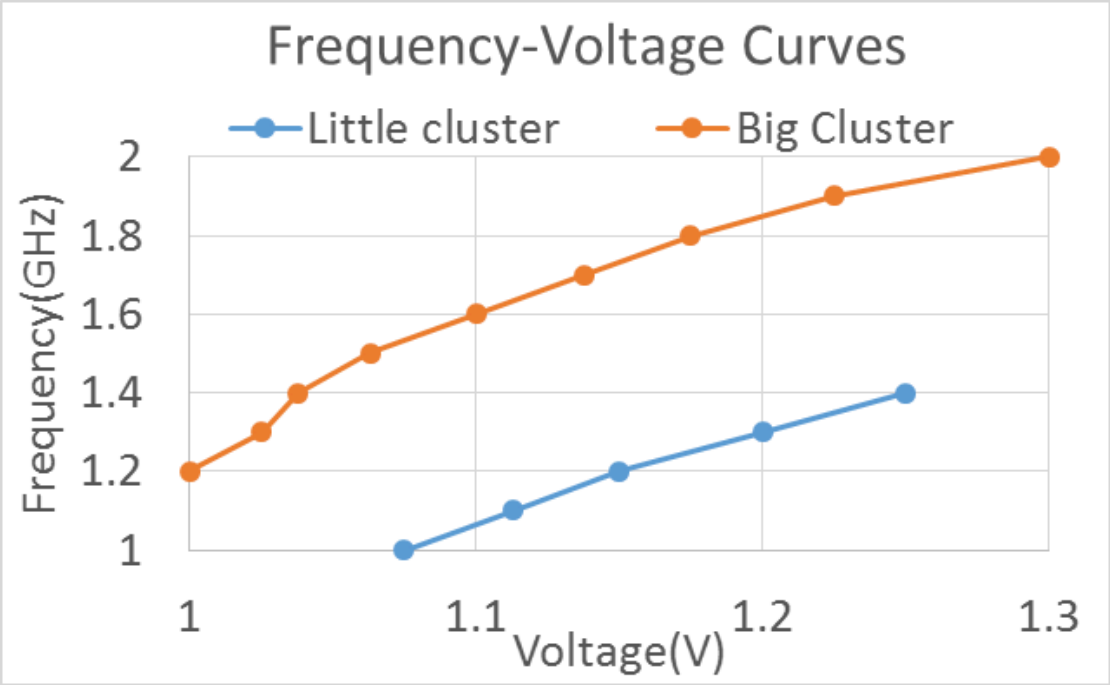

Figure 4.5: Voltage-frequency ranges

imental data. Such model accounts for both dynamic and leakage contributions. The fitting procedure estimates the parameters a and b by applying a least square algorithm to train power and frequency traces. The virtual platform has been implemented in Matlab version R2014a (8.3.0.532) and executed on a Lenovo T440s Thinkpad. The laptop has a 4th Gen Intel Core i5-4300U processor (3MB Cache, up to 2.90GHz). The virtual platform in our evaluation has 4 cores and is configured based on measurements from the Cortex A15 cores of the ARM big.LITTLE architecture. We specify and solve convex optimization with CVX [56], [55].

4.8 Experimental Results

In this section we report our experimental results. First, we illustrate how the Optimal Long Term Controller behaves. Then, we compare the optimal and the proposed WARM short term control in a simulation environment. We also show the benefits of WARM as compared to state-of-the-art techniques in [110], [176]. Then, we discuss the overhead associated to the implementation of WARM in a real operating system.

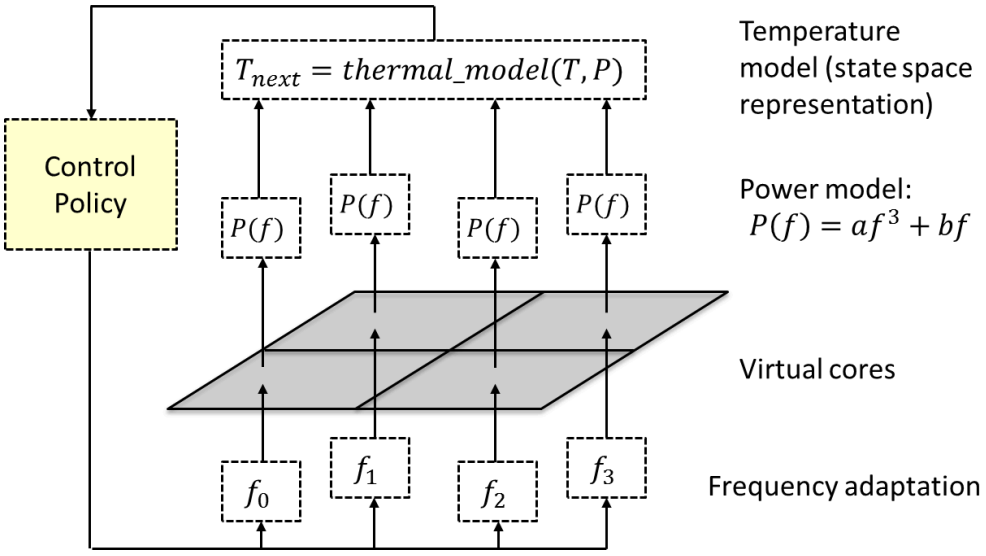

Figure 4.6: Block diagram of virtual platform

On the real system, we highlight the benefits of the TC migration compared to the technique in [105]. Finally, we evaluate the performance and reliability trade-offs of WARM compared to other CPU governors. The results presented in subsections 4.8.A-C are obtained through simulations on the virtual platform presented in the previous section. The results in subsections 4.8.D-F, instead, are obtained on the real Odroid XU3 platform.

4.8.1 Optimal Long Term Controller

The Optimal Long Term Controller activates at each long interval, solves the problem expressed in Equation (4.22), and provides values V_{LTC} and T_{LTC} to the STC. The problem is solved using convex optimization. In this experiment we implement the solver in the Matlab virtual platform for a single core, and activate the OLTC to meet a final target reliability of 0.8 at a target lifetime of 5 years. The activation rate for the OLTC is 30 days. Figure 4.7 shows in the first plot the controlled reliability curve for the target core, where the final value is above the target. In the second and third plot are shown the voltage and temperature respectively. Both the value of controlled and average are reported. For the two time intervals highlighted in the figure, we assume that the device is active, but not used, so that temperature and voltage are at the minimum

value. In this case the device leaves a reliability margin unexploited, and the OLTC reacts by providing higher targets in the subsequent intervals.

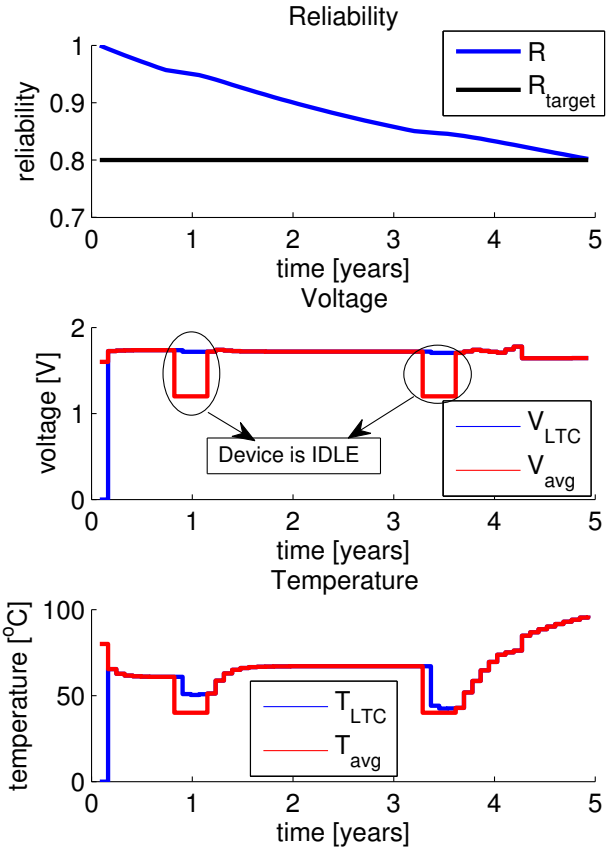

Figure 4.7: Behavior of Optimal Long Term Controller

4.8.2 Optimal Vs. Heuristic

With the virtual platform described in Section 4.6, we compare optimal and WARM policies. We generate two random input traces: a trace of required frequencies f_{req} and a trace of required quality flags H/L. We then provide such traces as input to the simulator. Figure 4.8 shows the average temperature and average frequency of the four cores over the time period of the simulated long interval. The values for T_{LTC} and f_{LTC} for this simulation are respectively 40^{o}C and 1400 MHz (reported with a black

straight line in the figure), for a LI with the duration of 200 SIs. The figure show that both policies are effective in meeting the reliability-induced average constraint in that the average temperature and frequency are below the constraint at the end of the LI. Moreover, we evaluate the performance of the two policies using the metric defined in Equation (4.35) [110].

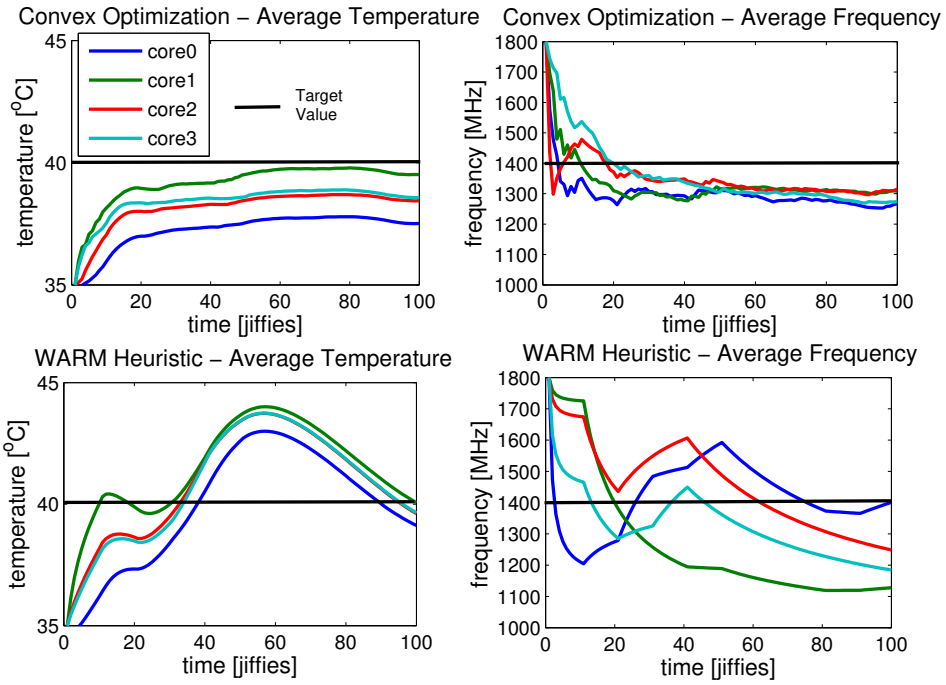

Figure 4.8: Comparison between optimal and heuristic policies

$$\delta(q) = \frac{\int_0^{t_{LI}} f_{app}(t)dt}{\int_0^{t_LI} f_{req}(t)dt} \tag{4.35}$$

Where q can be for H or L alone, or $both$ indicated with a $*$. Metric δ measures how close the performance provided by the policy is close to the required one, distributed across the whole long interval. The closer it is to 1, the better.

Table 4.1 reports the comparison of such values for the two policies respectively. For this experiment, we execute the two policies (Optimal and WARM) on a trace of required frequencies equal to the maximum on a LI with 200 SIs. Then, we vary the average target temperature T_{LTC} and frequency f_{LTC} and the percentage of H tasks in the trace. The results in the table allow us to conclude that the proposed heuristic is

effective in approximating the results given by convex optimization. When the convex approach gives better score, the heuristic is in at most 18% from the convex approach in the worst case. Also, we verify that, on the simulation platform, the heuristic is more than 400X faster than the convex optimization approach. This confirms that the heuristic is feasible to implement on a real system for runtime management.

Table 4.1: Comparison between convex and heuristic approaches

	$T_{LTC} = 45$, $f_{LTC} = 2000$		$T_{LTC} = 60$, $f_{LTC} = 1700$		$T_{LTC} = 120$, $f_{LTC} = 1400$		
	Opt	WARM	Opt	WARM	Opt	WARM	
$\delta(H)$	na	na	na	na	na	na	0% H
$\delta(L)$	0.696	0.692	0.778	0.743	0.700	0.700	
$\delta(*)$	0.696	0.692	0.778	0.743	0.700	0.700	
$\delta(H)$	0.831	0.655	0.917	0.835	1	1	10% H
$\delta(L)$	0.660	0.624	0.753	0.732	0.664	0.664	
$\delta(*)$	0.688	0.631	0.776	0.747	0.700	0.700	
$\delta(H)$	0.720	0.654	0.825	0.813	1	1	30% H
$\delta(L)$	0.689	0.665	0.776	0.747	0.582	0.582	
$\delta(*)$	0.698	0.662	0.791	0.768	0.710	0.710	
$\delta(H)$	0.717	0.633	0.819	0.746	1	1	50% H
$\delta(L)$	0.674	0.629	0.759	0.691	0.499	0.499	
$\delta(*)$	0.696	0.631	0.789	0.718	0.743	0.743	
$\delta(H)$	0.718	0.625	0.813	0.728	1	1	70% H
$\delta(L)$	0.638	0.644	0.724	0.686	0.415	0.415	
$\delta(*)$	0.690	0.631	0.782	0.713	0.812	0.812	

4.8.3 Comparison with State-of-the-Art

In this section we provide a comparison of the proposed WARM technique against the state-of-the-art techniques in [176] and [110] to highlight the advantages of WARM. The two techniques are both effective in guaranteeing the predefined target lifetime, but they have the following limitations. The technique in [176] cannot distinguish between applications with different quality requirements, and only fixes a maximum bound on operating voltage and frequency. For this it may cause degradation of user experience. We denote this technique as F_{max}. The technique in [110] can distinguish H and L applications, but temperature is only observed. It may incur into violations of the constraint on average temperature. This would also affect performance, as the control in the LTC would lower the constraint on average voltage/frequency for future long intervals. We denote this technique as T_{obs}. Figure 4.9 shows a comparison

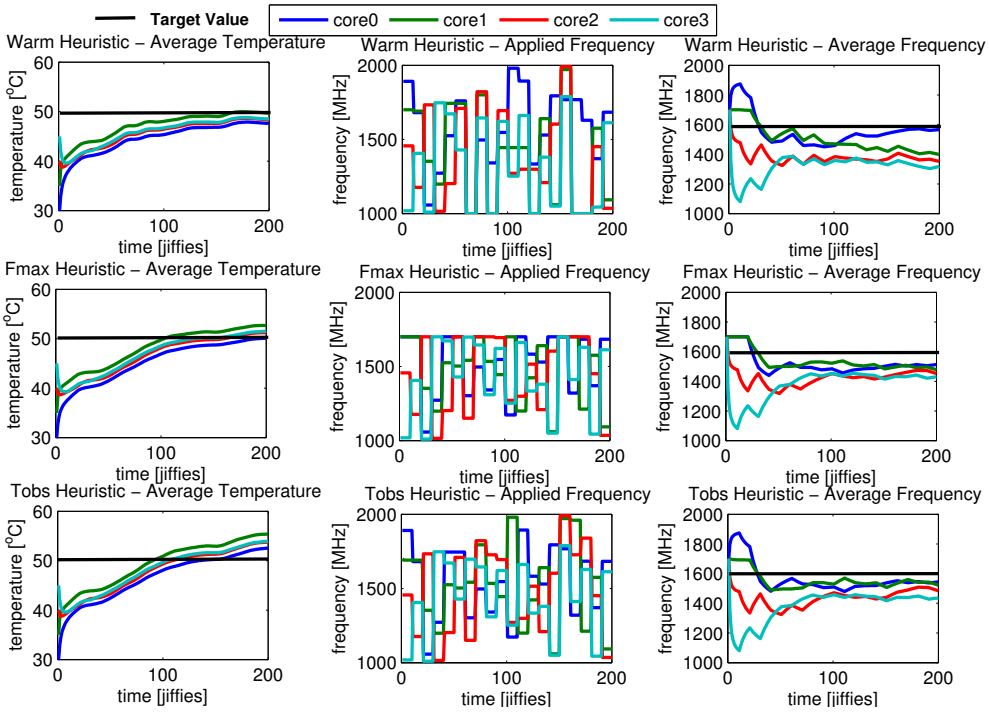

Figure 4.9: Comparison between WARM heuristic and state-of-the-art policies Fmax [176] and Tobs [110]

of the three techniques with on a random trace of tasks with 10% of H tasks [110]. In this experiment, we set $T_{LTC} = 45^oC$ and $F_{LTC} = 1600$, on a simulated long interval

of 200 short intervals (indicated as jiffies, which correspond to scheduling ticks). The results show two key advantages of the new heuristic over the previous two. First, F_{max} cannot meet the desired performance whenever the required frequency for H tasks is higher than F_{LTC}. This would results in degradation of user experience. Second, both F_{max} and T_{obs} violate the constraint on temperature, as the value of average temperature at the end of the long interval is higher than T_{LTC}. Finally, we show that our proposed technique can adapt to temperature variations. Looking at the graphs of applied frequencies (central column) we observe that they are different for WARM and T_{obs}. This is because WARM employs task migration to exploit higher performance from cooler cores while maintaining the temperature below the limit critical for reliability.

In the next experiment, we analyze temperature violations more into details. We execute a trace of required frequencies equal to the maximum for a LI with 200 SIs, and we keep the average target frequency equal to the maximum $f_{LTC} = 2000MHz$. Then we vary the average target temperature T_{LTC} and the percentage of H tasks. For each policy, we count the number of cores for which, at the end of the LI, the average temperature is within 5% from the target. On a total of 40 cases, WARM succeeds in 35 of them, which is 87.5% better than the state-of-the-art. The detailed results are reported in Table 4.2.

4.8.4 Implementation Overhead

Table 4.3 reports a detailed evaluation of all the overheads involved with the implementation of WARM on the target Odroid board. All the overheads are measured with the WARM infrastructure executing with a single little core active, executing at minimum frequency. All results are presented on an average of 100 samples, with a standard deviation lower than 5%. In the following, we provide a description of each of the actions profiled. We get the values of temperature sensors from the original device driver *thermal_exynos.c*. The values are stored in a variable that is shared with the Reliability Module. From the moment in which the value is recorded, until it is available to the Reliability Module, 160 cycles are elapsed. Similarly, the integrated voltage sensors are read in 170 cycles. In the WARM Governor itself, the STC algorithm runs for 74 cycles. The action of switching frequency, which is performed by any governor, lasts

Table 4.2: Average target temperature violations

$T_{LTC}[^oC]$	40	50	60	70	80	90	100	110	
WARM	1	0	0	0	0	0	0	0	0% H
Fmax	4	4	4	4	4	4	1	0	
Tobs	4	4	4	4	4	4	1	0	
WARM	1	0	0	0	0	0	0	0	10% H
Fmax	4	4	4	4	4	4	1	0	
Tobs	4	4	4	4	4	4	1	0	
WARM	1	1	0	0	0	0	0	0	30% H
Fmax	4	4	4	4	4	4	1	0	
Tobs	4	4	4	4	4	4	1	0	
WARM	1	0	0	0	0	0	0	0	50% H
Fmax	4	4	4	4	4	4	1	0	
Tobs	4	4	4	4	4	4	1	0	
WARM	1	0	0	0	0	0	0	0	70% H
Fmax	4	4	4	4	4	4	1	0	
Tobs	4	4	4	4	4	4	1	0	

for 16824 cycles (corresponding to 8.4us on average). All the previous overheads are obtained by sampling the ARM cycle counter in the kernel space.

In user space, we record the execution time of the TC routine, with which, the medium term temperature targets are updated and tasks are migrated. The time elapsed is 1.5 ms. Considering that the activation rate of the TC is 1 second, this represents only a 0.15% of time overhead. Also, every second the application manager passes to the kernel space the ID of currently application active in foreground. This operation takes 10214 cycles (corresponding to 5.1us on average). Similarly, we record the LTC execution time, which results in 342ms. Considering that the LTC activates in the order of days, this is a very low overhead. In userspace, timing overheads are measured using the *gettimeofday* function. In this work we target mobile CPUs, which today can have 8

cores (like the ARM big.LITTLE processor in the Odroid XU3). Given the low overhead of the implemented solution, reported in Table 4.3, it is possible to implement it on every mobile device.

Table 4.3: Implementation overheads

Overhead	Metric
Temperature sensor reading	160 cycles
Voltage sensor reading	170 cycles
Frequency change	16824 cycles
STC algorithm	74 cycles
Passing *process ID*	5 us
LTC execution	342 ms
TC execution	1.5 ms

4.8.5 Benefits of Task Migration

In this subsection we present the benefits of including the Thermal Controller and its task migration in the reliability management policy with respect to the previous technique proposed in [105]. To this aim, we present the effects of both inter- and intra-cluster migration in the ARM big.LITTLE architecture.

For intra-cluster, we execute a single CPU-intensive task on a big core. To isolate only the effect of temperature control through migration, we keep the frequency constant to the maximum value and we keep only two cores active (core4 and core7). First, we employ the technique in [105], which is static (i.e. it cannot leverage reliability-aware migration). The right plot of Figure 4.10 shows the reliability curves for the two cores, which result to be unbalanced. In this and following experiments, to derive the reliability curves in a reasonable experimental time we activate the LTC every 10 seconds. Then, in the model we update reliability as if 30 days are passed [105]. We repeat the same experiment while activating the migration policy. Thanks to this, WARM is able to migrate the task between the two cores when the temperature exceeds the limit critical for reliability. The reliability curves for this case are shown in the left plot of

Figure 4.10. The migration policy of WARM is able to keep the degradation among cores more balanced, thus a more efficient utilization of the performance budget of a multicore platform. Therefore, reliability-aware intra-cluster migration by itself may increase the lifetime of a multicore platform of more than 1 year, for a final reliability of both 0.8 and 0.6. For inter-cluster, we execute the Vellamo Metal benchmark, labeled

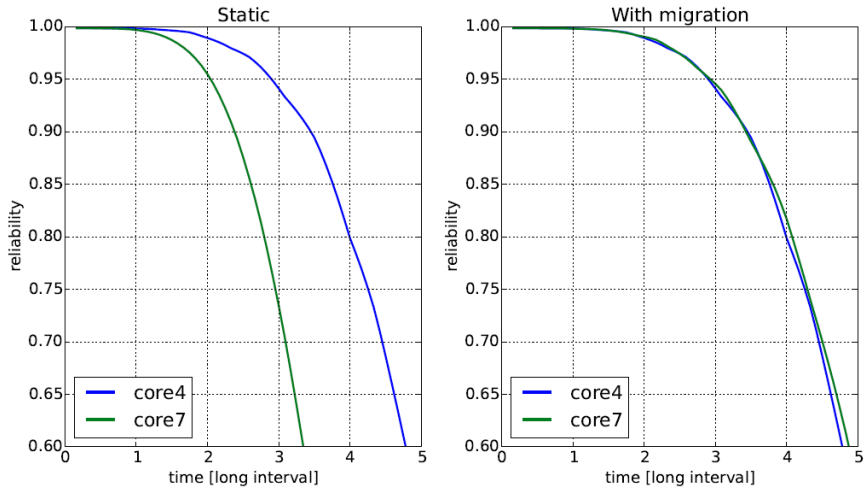

Figure 4.10: Effect of intra-cluster migration

as an H application. This is a benchmark evaluating the performance of the CPU that provides a final score which we use for comparison. We first execute the benchmark with the technique from [105]. Since this technique has static allocation, by default it executes the benchmark int the master cluster (which is the LITTLE one). We repeat the experiment while activating WARM, which can also leverage task migration. WARM, thanks to the application manager, recognizes that Vellamo Metal is a H application and automatically places it in the big cluster. Figure 4.11 reports the results. The plot on the right shows the reliability curves for the most degraded core of the big cluster. The plot on the left reports the final score obtained with the Vellamo benchmark. The result is that the technique from [105] leaves a reliability margin unexploited, at a significant cost in terms of performance. The plot on the left reports the scores for the Vellamo Metal in the two cases. The proposed technique, in this case, achieves 100% of performance improvement.

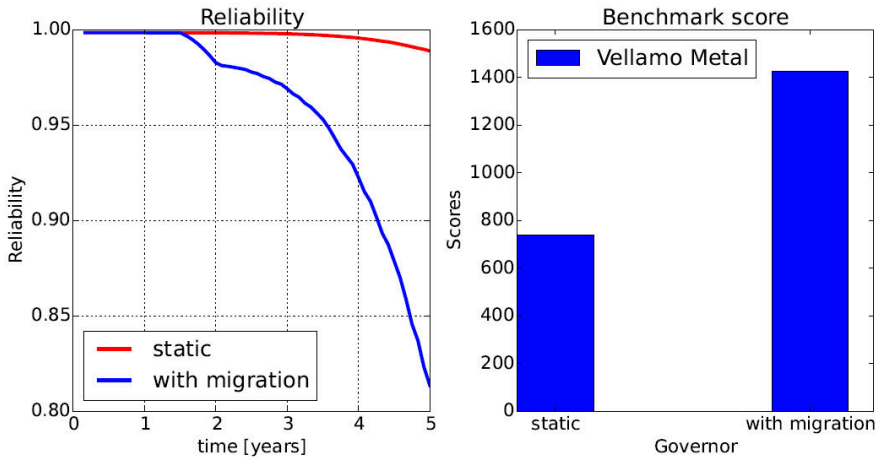

Figure 4.11: Effect of inter-cluster migration

4.8.6 Comparison with Standard Governors

In this subsection we show the comparison of our WARM governor against standard governors. First, we want to show that the performance provided by WARM for critical applications is comparable to that provided by the *performance* governor. To this aim, we execute a set of popular benchmarks for Android: AnTuTu, Vellamo Browser, Vellamo Metal, GeekBench and CFBench. Such benchmarks provide a score at the end of execution that we use as a comparison metric, similarly as in [111]. We first execute the benchmarks respectively with performance (giving a highest score) and powersave governor (giving the lowest score). Finally, we execute it with WARM. Figure 4.12 shows the benchmark scores (normalized as the increase with respect to the score obtained with the powersave governor) obtained with the three configurations. In all cases, the score obtained with WARM is in 4% of that obtained with performance governor. In the last experiment we show an example of how the implemented WARM technique behaves with real applications when compared to standard governors. For this experiment we run the Antutu benchmark with different frequency governors, among which our WARM reliability governor. For the execution with the reliability governor, Antutu is labeled as an H application. Together with Antutu, we execute a background program that forks and allocate a periodic task on each core, labeled as L. This simulate the presence of background activity on a mobile device. Figure 4.13 shows the reliability

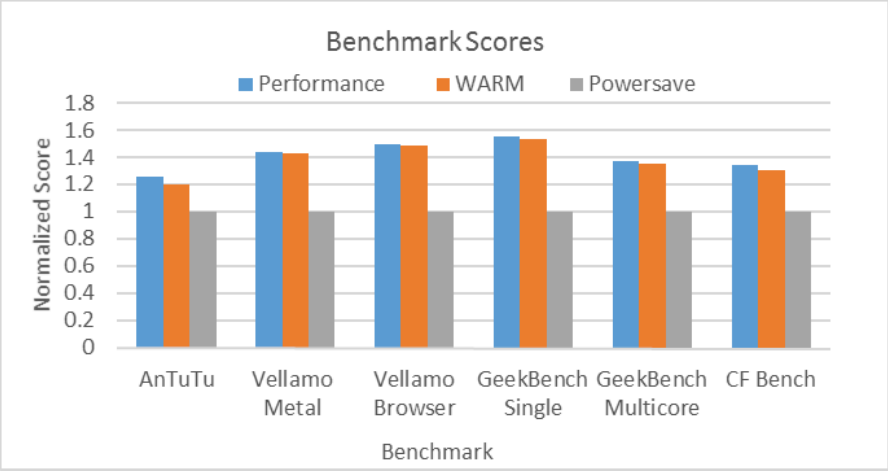

Figure 4.12: Performance of WARM governor

curves in the case of powersave (blue), reliability (pink), ondemand (light blue), interactive (green) and performance (red) governor. Performance and powersave governor give respectively the longest and shortest lifetime. The WARM Governor is able to meet the target of reliability of 0.8 before the target lifetime of 5 years (corresponding to 60 long intervals of 30 days each). Ondemand and interactive both fails at meeting such constraint.

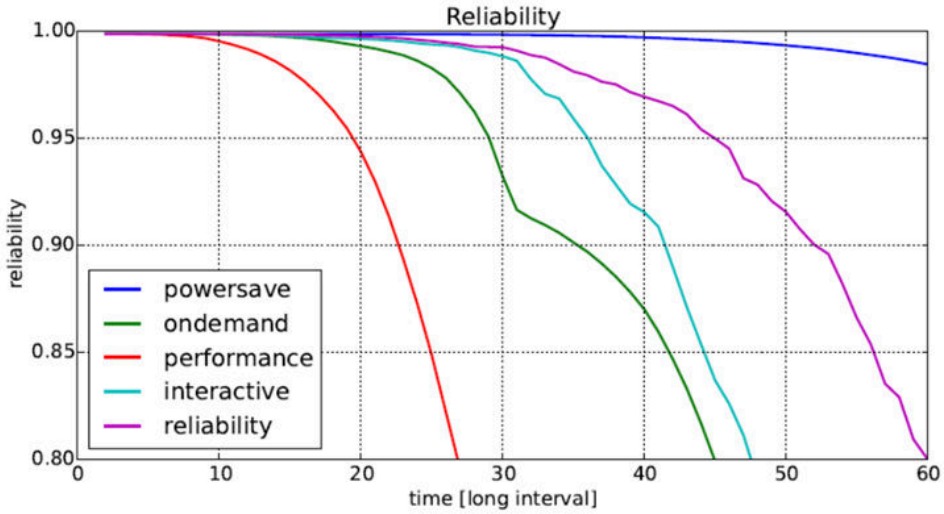

Figure 4.13: Reliability with different governors

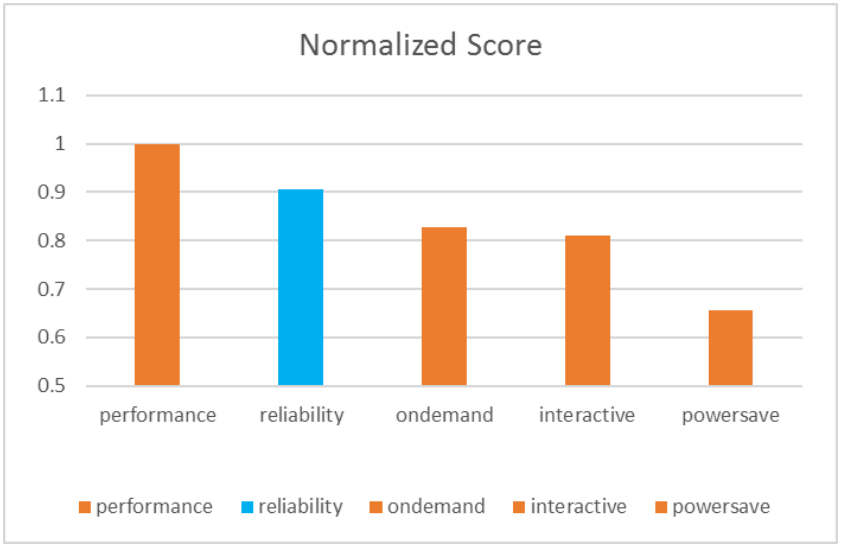

Figure 4.14: AnTuTu scores with different governors

Figure 4.14 shows the corresponding scores obtained with each governor, normalized to the maximum, obtained with the performance governor. The result shows that WARM not only achieves the target reliability, but also provides performance very close to the maximum.

4.9 Conclusion

In this chapter, we presented a framework for online reliability emulation and leverage it to develop a dynamic reliability management solution. The proposed solution exploits the long time scale of reliability changes and explores the interaction between long term and short term controller. This builds onto the short term solutions presented in Chapters 2 and 3 for power and thermal management. We develop an optimal controller for comprehensive temperature, performance and reliability management that leverages the CVX convex solver. We also show that convex solvers are not suited for implementation on a real device, due to computational overhead. Motivated by this, we develop WARM, a multilevel heuristic controller that approximates the solution of the optimal within 18% in the worst case, while executing more than 400x faster. WARM leverages task allocation to control temperature, while exploiting voltage and

frequency scaling to provide maximum performance to critical applications. We show that, since temperature is a major concern for degradation, WARM meets temperature constraints within 5% in 87.5% more cases than the state-of-the-art. Also, WARM task allocation achieves up to 1 year lifetime improvement for a multicore. We show that it can achieve up to 100% of performance improvement on cluster architectures, while guaranteeing the reliability target. In the following chapter, we include variability in power, performance and degradation rate which arise from fabrication process inaccuracy and increase with CMOS scaling.

Chapter 4 contains material from "WARM: Workload-Aware Reliability Management in Linux/Android", by Pietro Mercati, Francesco Paterna, Andrea Bartolini, Luca Benini and Tajana Šimunić Rosing, which appears in IEEE Transactions on Computer-Aided Design of Integrated Circuits and System [106]. The dissertation author was the primary investigator and author of this paper.

Chapter 4 contains material from "Workload and user experience-aware dynamic reliability management in multicore processors", by Pietro Mercati, Andrea Bartolini, Francesco Paterna, Luca Benini and Tajana Šimunić Rosing, which appears in Proceedings of the 50th Annual Design Automation Conference (DAC '13). ACM, New York, NY, USA [110]. The dissertation author was the primary investigator and author of this paper.

Chapter 4 contains material from "A Linux-governor based Dynamic Reliability Manager for android mobile devices", by Pietro Mercati, Andrea Bartolini, Francesco Paterna, Luca Benini and Tajana Šimunić Rosing, which appears in Proceedings of Design, Automation & Test in Europe Conference & Exhibition (DATE), Dresden, 2014 [105]. The dissertation author was the primary investigator and author of this paper.

Chapter 4 contains material from "An On-line Reliability Emulation Framework", by Pietro Mercati, Andrea Bartolini, Francesco Paterna, Luca Benini and Tajana Šimunić Rosing, which appears in Proceedings of the 2014 12th IEEE International Conference on Embedded and Ubiquitous Computing (EUC '14). IEEE Computer Society, Washington, DC, USA [109]. The dissertation author was the primary investigator and author of this paper.

Chapter 5

Variability Emulation and Management

Variability is a major challenge for integrated circuits, that strongly affects power, temperature and reliability. In the previous chapters we assumed that power consumption, performance and degradation rate are at their nominal value. This is clearly not the case in today's devices. This chapter first presents VarDroid, a low-overhead tool to emulate power and performance variability on real platforms, running on top of the Android operating system. VarDroid enables analyzing the effect of variability in power and performance while capturing the complex interactions characteristic of mobile workloads, thus relating to user's quality of experience. We present use cases to show the utility of VarDroid to test applications, device and OS robustness under the effects of variability. Our results show that a variability-agnostic OS can incur in a performance penalty of up to 60% and a power penalty of up to 20%. Then, we use online variability emulation to develop a variability-aware OS scheduling algorithm that assigns the workload to the cores and sets the power/performance tradeoffs to meet the mobile processor's lifetime constraints while adjusting to variability and improving the overall performance. We implement our algorithm in Android OS, integrating it with VarDroid on a mobile phone and show that it achieves up to 160% performance improvement over the state-of-the-art while meeting the lifetime constraints.

5.1 Introduction

Variability in integrated circuits refers to the deviation of the actual value of a design parameter from its nominal value, such as transistor channel length and threshold voltage. Variability is a key concern in modern integrated circuits, as it worsens with scaling of transistor dimensions [27]. Sources of variation are static and dynamic. Static variations are caused by manufacturing process imprecision, causing ICs to have frequency, power consumption and degradation rate different from their nominal values [141], [113], [127]. Dynamic variations, such as voltage and temperature, impact IC over time [153], [176], increasing aging phenomena, such as Negative Bias Temperature Instability (NBTI), Electromigration (EM), Hot Carrier Injection (HCI) and Time Dependent Dielectric Breakdown (TDDB) [75]. In multicore processors, transistor-level static variability determines power and performance distributions across cores of the same model. Consequently, cores have a power consumption, operating voltage/frequency and degradation rate which is different from nominal right after fabrication [28], [70].

Given that dynamic variations play an important role, a key idea is to leverage sensors to expose variability to higher levels of the software stack (e.g. the operating system) to manage it at runtime [58]. This strategy is referred to as Dynamic Variability Management (DVM). DVM can leverage common hardware sensors such as performance counters and temperature sensors, or more sophisticated ones, such as degradation sensors and frequency sensors (which are devices capable of monitoring path delays) [90], [35], [175]. The latter, however, are only available on prototypes and research platforms. Simulators have been developed to help designers make chips more robust to variations [141], [113]. These tools can be integrated into the design flow to estimate the resulting impact of variability. However, they cannot account for dynamic variations and cannot capture real workloads behavior. Moreover, they require a highly detailed architectural description, which may not be always available. It is required a tool to emulate the impact of variability and to test DVM strategies on real commercial devices [148].

Dynamic Variability Management (DVM) techniques have been proposed to counteract frequency variations [128], [157]. However, these techniques do not account

for processors lifetime requirements, with the risk of violating them. Moreover, they account only for static variations and do not account for dynamic temperature variations. Dynamic Reliability Management (DRM), instead, aims at guaranteeing a target reliability within a predefined target lifetime. The works in [153], [176] counteracts aging at runtime by Dynamic Voltage and Frequency Scaling (DVFS). In [110] and [105] the authors present a DRM technique for homogeneous multiprocessors which is workload-aware, thanks to a *borrowing strategy* and the use of on-chip sensors for reliability monitoring, and they implement it on a real Android device. However, these DRM techniques have the following fundamental limitations: (i) they cannot counteract unbalanced degradation among cores of the same multiprocessor, (ii) they cannot take advantage of diverse frequency of the cores and (iii) they neglect potential performance improvement under temperature variations. Moreover, they do not account for variability when allocating task.

In this chapter, we describe VarDroid, a low-overhead framework to emulate the effect of performance and power variability on real Android devices, to capture the effects of environmental changes and interaction with users, which is not possible with simulators. Thanks to this, VarDroid can estimate the impact of variability on power and quality of user experience. Also, the Variability-induced Power Breakdown (VIP) allows us estimate the ratio of dynamic and leakage power due to variability. We present the assumptions of our work, the details of the implementation, and use-cases to demonstrate the feasibility and ease of use of our framework. Figure 5.1 better explains the motivation and goal of VarDroid. At the transistor level, variability mainly affects channel length and threshold voltage. At the circuit level, this has an impact on path delay and power consumption. As a preliminary study, we implement a circuit consisting of AND and OR gates using 32-nm technology with HSPICE simulator. For each transistor, we consider a variation in both channel length and threshold voltage which follows a Gaussian distribution with 3σ equal to 10% of the original value, similarly as in [10]. A Monte Carlo simulation with 5000 iteration produces the distributions of path delay (which affects operating frequency) and leakage power. The figure shows that such distributions are also well fitted by Gaussian curves. Finally, at the processor level, previous work [28], [141], [113] (on the left) focuses on simulating the effect of

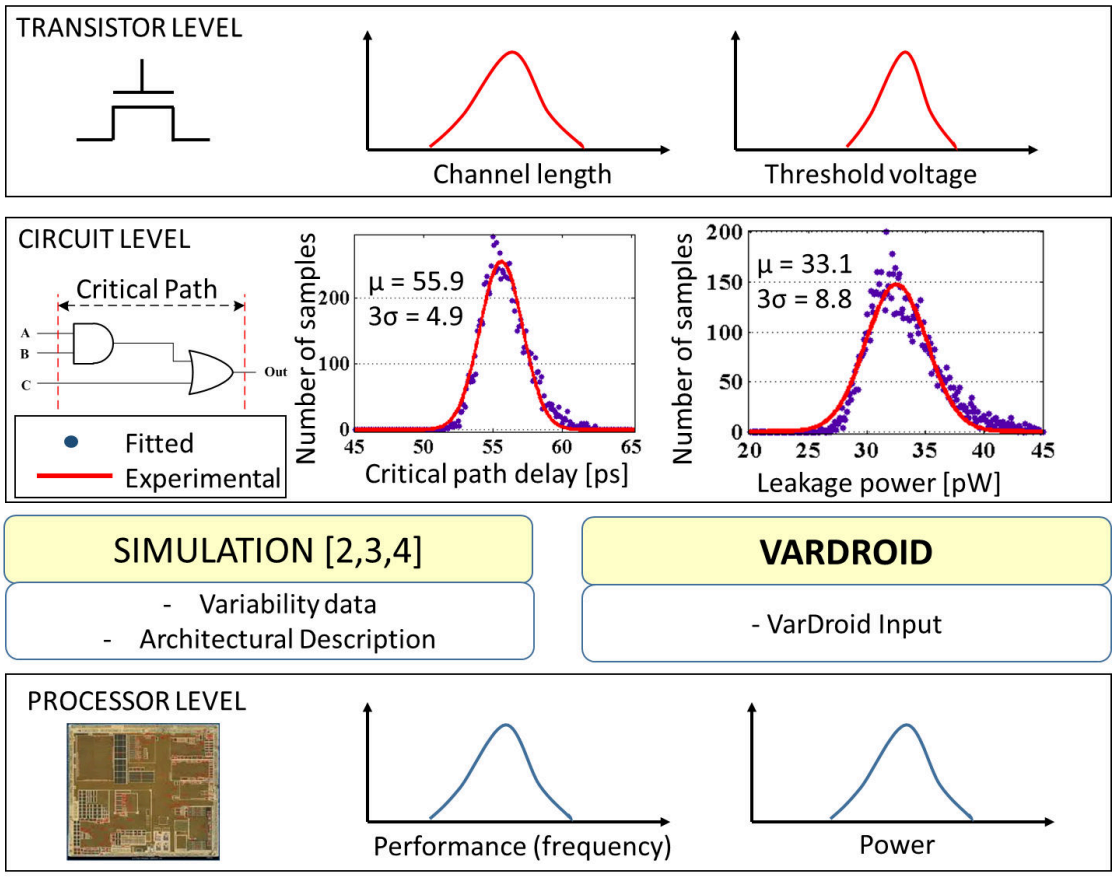

Figure 5.1: VarDroid context and motivation

variability on a multiprocessor starting form detailed data on transistor parameters and propagates it to the processor level. In the end, it achieves Gaussian distributions for power and performance (i.e. operating frequency) for a large number of processors. In our work, instead, we emulate variability on a real device, by injecting perturbations in the Operating System (denoted as *VarDroid input*). Thanks to this, we do not require detailed low-level information about transistor variability. Nevertheless, we will show that we still obtain Gaussian distributions for power and performance, thus matching the results of state-of-the-art simulator and low level models. To clarify this, we describe models for power and frequency as a reference. In our experiments, we use VarDroid to investigate the robustness of applications with respect to variability, the impact of variability on user experience and the tolerance of current OS migration policy. Our results show that performance of applications may drop by up to 35% in presence of variability

and that a variability-agnostic OS can incur in a performance penalty of up to 60% and a power penalty of up to 20%.

Using the proposed variability emulation framework, we present a novel Dynamic Variability Management policy for the comprehensive runtime management of degradation rate variability, frequency variability, frequency degradation over time, application-specific quality requirements for user experience, lifetime constraints, ambient temperature variation. For the latter, our DVM policy features **T-boost**, a novel strategy that allows to improve performance during periods of high temperature. Our solution uses sensors for frequency and aging to improve variability control robustness, through a variability-aware OS scheduling and frequency scaling algorithm. The goal is to maximize performance against variations, while meeting the target reliability for each core within its predefined target lifetime.

We implement our policy on a real Android device and show its effectiveness against comparison techniques. The results show that our policy achieves up to 160% performance improvement on variations-affected platforms relative to state-of-the-art. We implement the proposed task allocation strategy leveraging non-invasive Linux features on a real Android device and evaluate its effectiveness against state-of-the-art approaches through an extensive set of experiments. To the best of our knowledge, this is the first comprehensive sensor-based dynamic variability management technique for mobile devices.

5.2 Related Work

5.2.1 Variability Emulation

In the last decade, variability has been the subject of numerous publications, from device-level simulation up to approximate computing for applications. Architectural-level variability simulators have been developed, such as VARIUS [141], VAM [113], and VARIUS-NTV [81]. These tools can be integrated with simulators such as GEM5, but they require detailed architectural description. Garg et al. [51] propose a framework to analyze the impact of variability at the system level to help designers determining the tradeoff between performance and clock domain granularity. These techniques extract

random values for architectural parameters and inject variability in the configuration of the simulator. The main drawback of such an approach is that simulations are very slow, as a few seconds of clock time for a real device could take hours.

An alternative approach in represented by variability emulation, which consists in injecting variations on a real device. In ERSA [92], errors are injected in architecturally visible registers. Similarly, in [132] a large fault injection campaign is conducted in the openSPARC T1 core logic. In [93], a combined hardware and software solution aims at identifying anomalous software behavior in presence of hardware faults. However, solutions based on hardware prototypes and FPGAs present low flexibility and portability. Work in reference [88] proposes VESPA, a variability emulation framework for SoC performance analysis. The proposed solution addresses the challenge of translating component-level characteristics into system-level performance distribution. The approach is evaluated for a 802.11 MAC processor and an MPEG encoder. Moreover, VESPA is used to assess the performance distribution of processors design prior to fabrication, given the design synthesis. Therefore is not effective in capturing real workload and the effects of real environment. Work in [165] proposes VarEmu. Implemented as an extension of the QEMU operating system, this can be used to evaluate variability-aware software techniques. VarEmu extracts timing and cycle count from the emulated code. This information is fed into a variability model with configurable parameters that determines energy consumption and fault variations in the virtual machine. VarEmu relies on models for power estimation, which have to be adapted to different platforms and may present accuracy problems. None of the previous work on variability emulation addresses mobiles and the crucial need of capturing interactive workload on variability-affected devices. Our approach is different in that we inject faults directly in the native OS of real mobiles in the form of a VarDroid input.

Recent publications on Dynamic Variability Management (DVM) propose scheduling and DVFS algorithms for embedded platforms [127] and for Android-based devices [107]. However, these solutions do not account for real workload and real user interaction. Emulation in Android has been addressed before for reliability [105], [109]. Work in [105] implements a kernel module to virtualize the presence of degradation sensors to emulate the effect of degradation mechanisms. Our work is orthogonal to [105],

in that we focus on performance and power variability, while the technique in [105] aims at emulating degradation. At this time, no publication presents power and performance variability emulation capable of running on real mobiles. In this work, we develop and implement a framework for emulating the effects of performance and power variability on real Android devices, while capturing real mobile workloads. Finally, we show how to use it to test system robustness against variability.

5.2.2 Variability Management

The approach of exposing and managing variations at runtime has been exploited in the recent years through Dynamic Thermal Management (DTM) [21] and Dynamic Reliability Management (DRM) [153], [176], [110]. DTM techniques monitor temperature and change dynamically the operating conditions to avoid overheating, but they do not guarantee a target lifetime constraint. DRM techniques, instead, manage temperature and voltage to guarantee a predefined target lifetime.

DRM approaches in [78], [176] monitor aging with a voltage and temperature dependent reliability model, trade off performance and reliability to meet a target lifetime by limiting the maximum operating voltage thanks to a PID controller. These approaches do not consider user experience. Moreover, they target only a single core scenario. In [110] and [105] the authors present a DRM technique for homogeneous multiprocessors which is workload-aware, thanks to a *borrowing strategy* and the use of on-chip sensors for reliability monitoring, and they implement it on a real Android device. However, this technique does not take into account frequency and degradation rate variations across cores of the same multicore platform, resulting in reliability unbalance. In our experiments we showed that this could lead from 40% to 120% of performance loss in a variation affected platform.

Work in [128], [157] proposes static solutions for the parallel streaming workload task allocation problem. However, they do not account for independent tasks with different execution requirements. In this case a dynamic control is more effective. In [127] the authors consider aging-aware workload allocation for homogeneous multicores, but they focus on the NBTI phenomena and simulated multimedia applications. We instead consider TDDB as mechanism of degradation. In fact, work in [60]

shows that TDDB exhibits a much faster degradation rate with respect to NBTI. The technique in reference [163] keeps the temperature of the cores below a threshold in order to maximize the lifetime. However, such a formulation does not give guarantees on the actual lifetime of a device and can heavily penalize performances. Most of the reliability-aware techniques in literature do not consider different degradation rates among the cores of a platform due to variability. Furthermore, they do not make use of aging sensors [148], [175] and frequency monitors [35]. The publications reviewed here consider as target platforms single core processors or homogeneous multiprocessors. However, mobile devices today are equipped with heterogeneous MPSoCs. Finally, none of these approaches accounts comprehensively for frequency and degradation rate variation, ambient temperature variation, lifetime constraints and different workload requirements.

5.3 VarDroid Variability Models

Variability in integrated circuits manufacturing has two components: D2D (die-to-die) and WID (within-die) [141]. WID variations can be further separated into systematic and random variations. At the microarchitectural level, the elements of key importance for variability are the transistor threshold voltage V_{th} and the effective channel length L_{eff} of transistors. More recently, C2C (core-to-core) variations have been considered [28]. Following this model, low-level variations in V_{th} and L_{eff} result in operating frequencies and power consumption which are distributed among cores. Since V_{th} and L_{eff} can be considered normally distributed with good approximation, power consumption and critical path delay (thus, operating frequency) can be considered normally distributed among cores as well [141]. Power is the sum of dynamic and leakage contributions. Dynamic power has a well-known dependency on frequency and voltage [85]. Leakage power, instead, depends on leakage current, which can be modeled as in Equation (5.1).

$$I_{leak} = b \left(\frac{kT}{q} \right)^2 \exp \left(\frac{q(V_{off} - V_{th})}{\eta kT} \right) \tag{5.1}$$

Where q is the electron charge and k is the Boltzmann constant. The terms b, η

and V_{off} are empirically determined parameters. As shown in [141], the knowledge of V_{th} variance can be exploited to assess the impact on chip leakage power. As a result, the value of the ratio between perturbed and nominal leakage can be expressed as in Equation (5.2).

$$\frac{P_{leak}}{P_{leak_0}} = \frac{V_{leak}}{V_{leak_0}} = \exp\left(\frac{1}{2}\left(\frac{q\sigma}{\eta kT}\right)^2\right) \tag{5.2}$$

Where P_{leak_0} and I_{leak_0} are the nominal values of leakage power and current. This indicates that the variation in leakage depends on the standard deviation σ of V_{th}. As for critical path delay, this is related to the switching delay of an inverter gate, which is expressed by Equation (5.3).

$$T_g = a\frac{L_{eff}V}{\mu^*(V - V_{th})^\alpha} \tag{5.3}$$

Where a is an empirically determined parameter, α is typically 1.3 and μ^* is the mobility of carriers. Given the critical path delay T_g, which depends on gate switching delays, the maximum frequency at which a processor can be clocked is given by its inverse $f_{MAX} = 1/T_g$.

In this work, we refer to Equations (5.1), (5.2) and (5.3) to characterize VarDroid configurations of the target platform. This is done assuming a diverse impact of variability in L_{eff} and V_{th}, expressed as the ratio μ/σ of the underlying Gaussian distribution. Finally, at the processor level, variability results in Gaussian distributions of maximum frequency (e.g. performance) and power. VarDroid naturally captures dynamic variations, such as workload and environmental conditions (such as SoC and ambient temperature), given that it is implemented on real devices.

5.4 VarDroid Architecture

Figure 5.2 shows the block diagram of the proposed implementation. It has two main components: the **VarDroid Engine** and the **VarDroid Monitor**. The **VarDroid Engine** is the software element which perturbs the system to emulate the effect of variability and has four components: the input configurations PERT, SCHED and INJ,

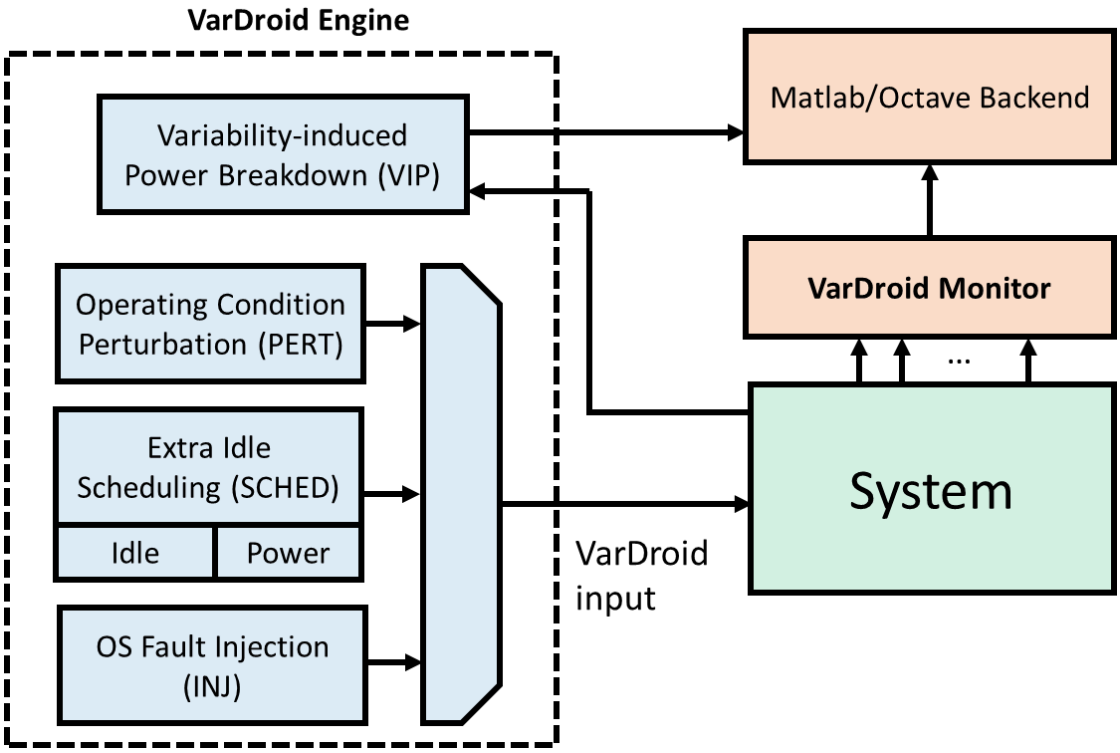

Figure 5.2: VarDroid block diagram

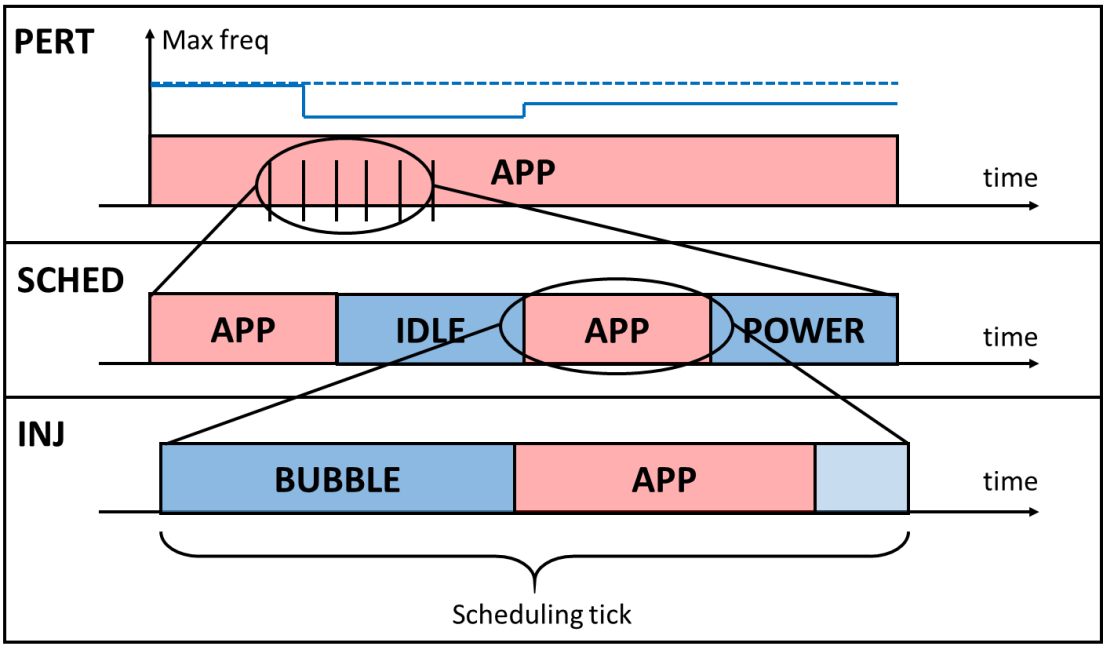

Figure 5.3: VarDroid input configuration

and the Variability-Induced Power breakdown emulator (VIP). The components are described into details in the following.

Operating conditions perturbation (PERT): The operating frequencies of cores are limited by PERT to mimic the effect of performance variability. We generate a Gaussian distribution for f_{MAX}, extract and assign randomly one value to each core. In this way, operating conditions are limited as if the multicore was affected by variations on performance. This emulates a scenario in which L_{eff} and V_{th} of transistors have variability, which reflects in f_{MAX} variability (i.e. performance), as described by Equation (5.3). The operating condition perturbation is implemented at the userspace level, and exploits the *sysfs* fields exposed by the *cpufreq* driver that allows to set an upper bound on operating frequency. The selected f_{MAX} represents the input of PERT to the system. Figure 5.3 shows that PERT perturbs the behavior at a high level (i.e. system level), by selecting an upper bound for operating conditions.

Extra scheduling (SCHED): SCHED forces the scheduling of extra periods to interleave the normal execution of applications. It is implemented as a program that alternates periods of execution and sleep (implemented with *nanosleep*). The effect of SCHED is varied by tuning the *nanosleep* duration, which affects the SCHED duty cycle. SCHED can be configured with an *Idle* application for performance variation (a program executing a long sequence of NOPs) and a *Power* application for power consumption variation (a sequence of cpu-intensive operations). In the first case, the emulated scenario is similar to the one achieved with PERT, in which L_{eff} and V_th variations result in frequency variations. In the second case, the scenario includes power variability, both dynamic and leakage. Figure 5.3 shows that SCHED occupies some scheduling intervals with spurious applications. Both PERT and SCHED are implemented as C programs and cross-compiled with Android NDK toolchain to run on the target platform.

OS fault injection (INJ): INJ is implemented by inserting in the OS kernel source code execution *bubbles* that perturb performance. The *bubble* is a fixed sequence of instruction which is inlined in the kernel source code. A single bubble executes a sequence of 20 NOPs (for the current design of VarDroid) at the beginning of each scheduling tick. The impact of variability can be tuned by choosing the number of bub-

bles (referred to as *bubble length*). INJ also emulates the scenario in which variability in L_{eff} and V_{th} results in performance variability. The fault injector is implemented as an inlined function which is invoked in the *scheduling_tick()* function in the scheduler source code (*core.c*). Figure 5.3 shows that INJ occupies part of the scheduling interval itself with spurious instructions. We denote the VarDroid input ν in the three configurations respectively as the percentage of frequency degradation over f_{MAX} for PERT, the nanosleep duration for SCHED and the bubble length for INJ.

The three input configurations have complementary benefits. PERT and INJ can emulate performance variations, but are not effective in emulating power variations. INJ emulates performance variations at a finer granularity compared to PERT, but requires recompiling the Linux kernel. PERT and SCHED, instead, can be more easily ported, as they are userspace programs. Note that the three input configurations inject variability at different layers of the software stack, and consequently at different time granularity, as Figure 5.3 shows.

Variability-induced Power Breakdown (VIP): Beside the three input configurations, VarDroid features VIP, a framework to estimate the proportion of dynamic to leakage power depending on input variation on performance. The main motivation for the design of VIP is that on a real platform, we can only measure the total power consumption, but we cannot distinguish the dynamic and leakage contributions. This is important, because the variation on parameters L_{eff} and V_th determines a variation in the ratio between dynamic and leakage power. It is well known that dynamic power P_{dyn} is directly proportional to frequency, and VarDroid establishes a relation between the input variability ν and f_{MAX}. Since f_{MAX} depends on V_{th}, this can be derived from ν as well. The leakage power P_{leak} depends exponentially on V_{th}. Overall, this allows deriving the ratio P_{dyn}/P_{leak} as ν varies, as expressed by Equation (5.4).

$$\frac{P_{dyn}}{P_{leak}} \approx \sigma \frac{f(\nu)}{\exp(g^{-1}(\beta f(\nu)))} \tag{5.4}$$

Where σ, β are fitting parameters. The function g can be derived from Equation (5.3), and it is so that $g(V_{th}) = f(\nu)$, while $f(\nu)$ can be fitted experimentally. As ν increases, the ratio should also increase, as this reflects a scenario in which V_{th} is higher than nominal. VIP computes a ratio, thus it does not depend on which power is actually

consumed and measured. We will better characterize VIP in the results section.

The **VarDroid Monitor** infrastructure has been implemented by modifying the one described in [105]. It has a kernel sampling function, a monitor driver to transfer data to the user space and a monitor daemon to regulate the data transfer. The monitor infrastructure is able to get one sample per each scheduling tick (10ms in our setup) and transfers data to the userspace with low overhead. Finally, the Matlab backend allows parsing and plotting of results.

5.5 Vardroid Experimental Evaluation

We implemented VarDroid on the Odroid XU3 board. This board has a Samsung Exynos 5422 Octa core based on ARM big.LITTLE architecture, with a Cortex™-A15 2.0Ghz quad core cluster and Cortex™-A7 quad core cluster [3]. The platform has a 2MB L2 cache and a 2GB RAM. The OS is Android 4.4.4 with Linux kernel 3.10.9.

5.5.1 VarDroid Validation

The key idea for validating Android is to show that when varying the VarDroid input (for each one of the three configuration) we obtain a Gaussian distribution of frequency and power consumption, similar as in [28]. This proves that a variation of the VarDroid input reflects a scenario in which transistor parameters L_{eff} and V_{th} are subject to variability.

For validating VarDroid, we conduct a series of experiments using single-threaded microbenchmarks. The *cpu-bound* executes a series of ALU operations without accessing memory, the *L2-bound* executes a series of loads from the L2 memory and the *mem-bound* similarly executes a series of loads from the main memory. To avoid the effect of migrations, microbenchmarks exploit the *set affinity* mechanism to run on a specific core.

Performance Variability: In the following experiments, we keep only one LITTLE core active, running at fixed maximum frequency. Analogous results can be obtained for big cores. Figure 5.4 shows the normalized execution time of the three microbenchmarks for the three VarDroid configurations. The execution time of the bench-

marks increases as the VarDroid input increases. This means that VarDroid effectively makes the processor behave as if it had a maximum frequency lower than the nominal.

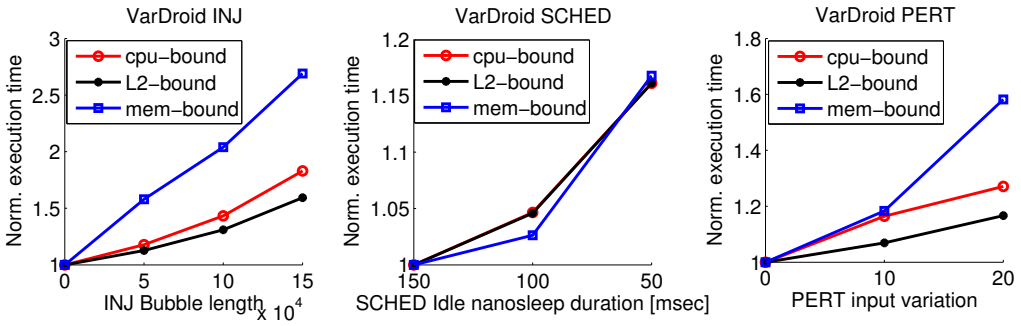

Figure 5.4: Microbenchmarks execution times

To further validate this behavior, we run the *cpu-bound* benchmark with random VarDroid input extracted from a normal distribution, in the three configurations. In Figure 5.5, the plots on the left show the histogram of VarDroid input. Then we collect the execution times and build a histogram, shown in the plots on the right. In the case of SCHED, we employed a smaller number of samples and a version of the *cpu-bound* benchmark of longer duration. The resulting shape match with the low-level models discussed from Section 5.3 and the example of Figure 5.1. In this way VarDroid emulation shows results as if the benchmarks were executed by different instances of the same processor, affected by performance variability. No variation in execution time is present when VarDroid is disabled.

Power Variability: Next, we verified the effect of VarDroid on power variability. In Figure 5.6 we show the histogram of random input for the SCHED Power configuration (e.g. *nanosleep* duration) and the corresponding histogram of average power consumption. In this experiment, we are measuring only the power consumption of the LITTLE cluster, but a similar result can be obtained for the big cluster. One sample of power consumption is computed by averaging the power over a period of 120 seconds, with a given SCHED power input. Also in this case, the histograms match the behavior expected from low-level models of Section 5.3.

In Figure 5.7, we perform the same measurement while executing a sequence of *cpu-bound* benchmarks. The second plot shows the power variability, while the third plot shows the execution time variability. The variation in the execution times is very

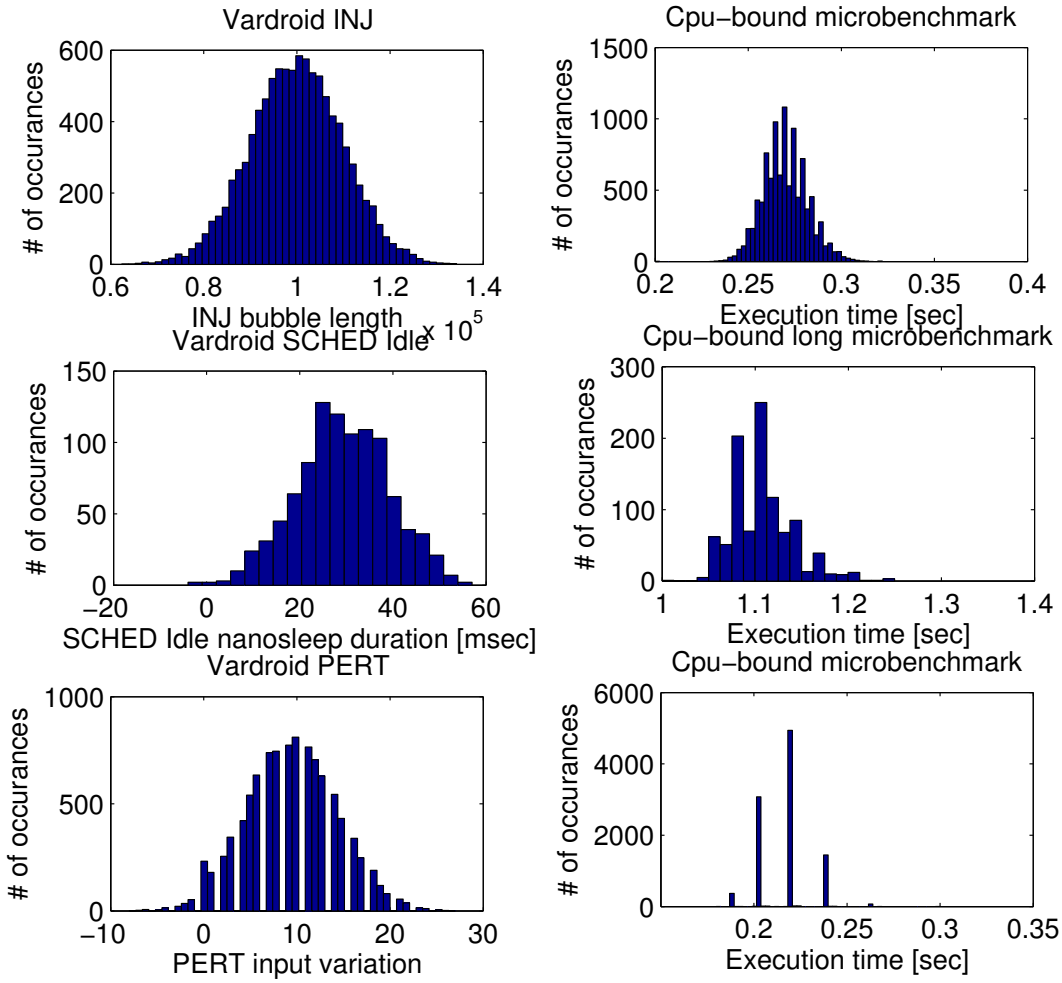

Figure 5.5: VarDroid performance distribution

low (in the 2% of the mean value). This means that the SCHED power setting can be effectively used to decouple power and performance variability effects.

Finally, Figure 5.8 shows the qualitative behavior of the VIP ratio. For simplicity, the relations involved in Equation (5.4) are assumed linear. Dynamic and leakage power are scaled in the [0,1] interval and the final ratio is normalized with respect to the minimum. The final behavior is that of a log-normal distribution. As reported in [85] the log-normal shape is generated by the exponential increase in leakage current, thus matching with the model described by Equations (5.2) and (5.3). Note that the distribution of the VarDroid input can be tuned to achieve a desired power and performance variation.

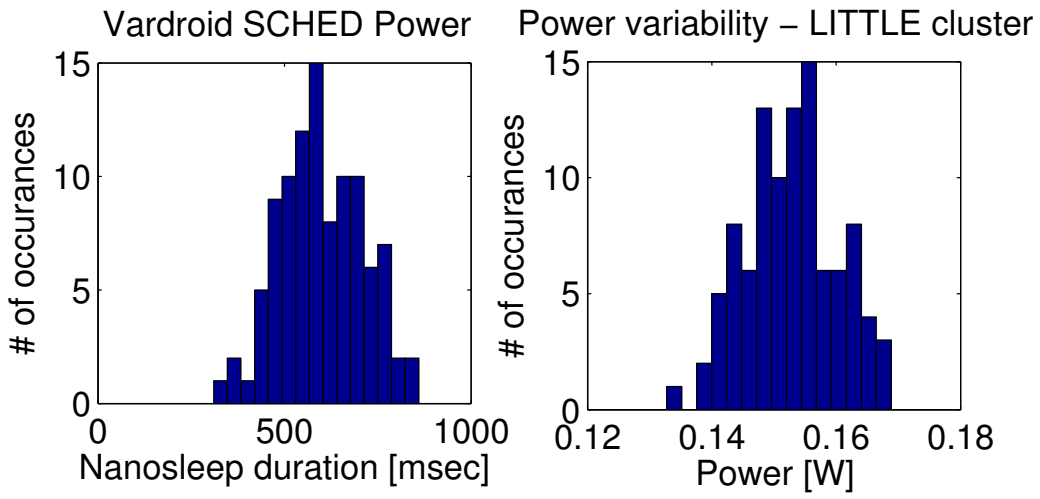

Figure 5.6: SCHED power variability of little cluster

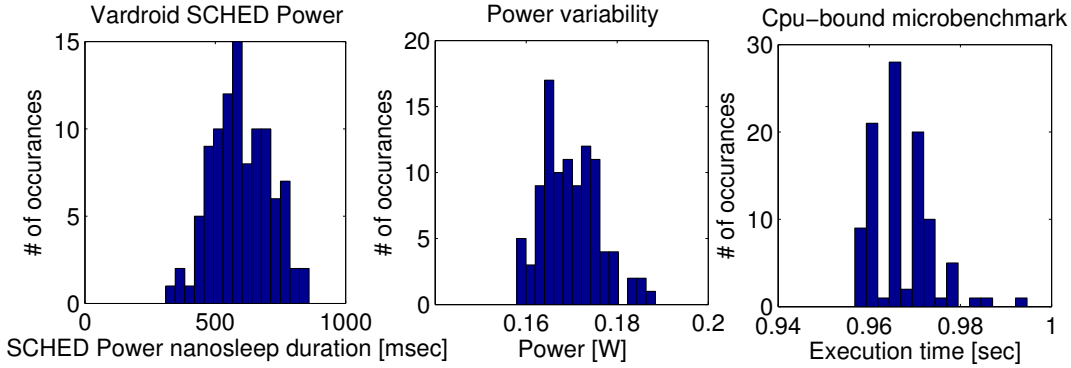

Figure 5.7: Power variability with cpu-bound benchmark

5.5.2 Use Cases

In the following experiments, we present use cases in which we can exploit VarDroid. The proposed emulation framework can be used to expose weaknesses of applications and operating systems, also relative to user experience, and to develop and tune dynamic variability management strategies accordingly.

Application Robustness: VarDroid can be used to investigate the impact of variability on real application performance and power consumption. In Figure 5.9b, we present the scores obtained with the mobile benchmark Antutu v5.6.1 [2] with increasing impact of INJ. Scores of mobile benchmarks are a common metric to compare the

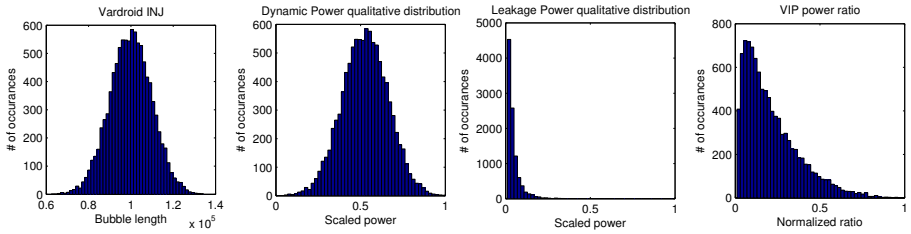

Figure 5.8: Variability-induced Power breakdown distribution

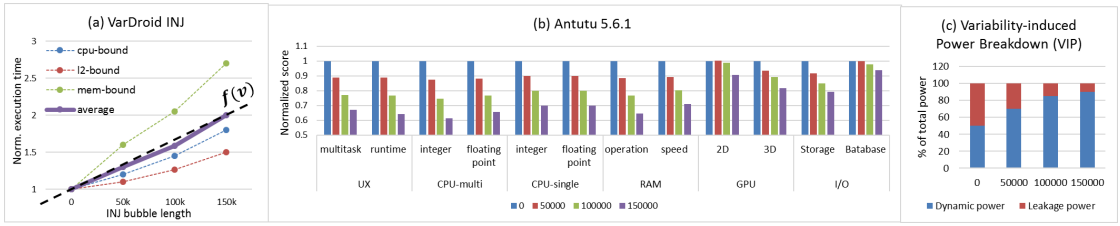

Figure 5.9: Performance degradation and VIP in Antutu benchmark

quality of execution of different devices. As expected, all the scores decrease as the INJ input increases, by up to 35%, also in accordance with the results obtained on microbenchmarks. Therefore, VarDroid can be used by application developers to test the performance of their apps in a real device affected by variability. Figure 5.9c shows the ratio between dynamic and leakage power obtained with VIP. The base case is assumed to have 50% of dynamic and 50% of leakage power, in accordance with ITRS [8]. In this case, the function f is derived from the results in Figure 5.3 from INJ microbenchmarking, also shown in Figure 5.9a. As expected, the ratio P_{dyn}/P_{leak} increases with the VarDroid input.

User Experience: Since VarDroid is implemented on a real device, it can capture real workload dynamics and user interaction, therefore it can be used to assess the impact of variability on user experience. For this experiment, we record and replay a 10 seconds-long touch event trace of a popular Android 3D game (Temple Run [7]), in order to reproduce the same workload [54]. Then, we execute the trace with different INJ configurations, and measure Frame per Seconds (FPS) by monitoring the SurfaceLinger service. Indeed, it is well recognized that FPS in 3D gaming is a good metric for quality of experience [131]. In particular, the closer to 60fps (which is the maximum allowed by Android), the better. The results in Figure 5.10 show the different FPS traces. As

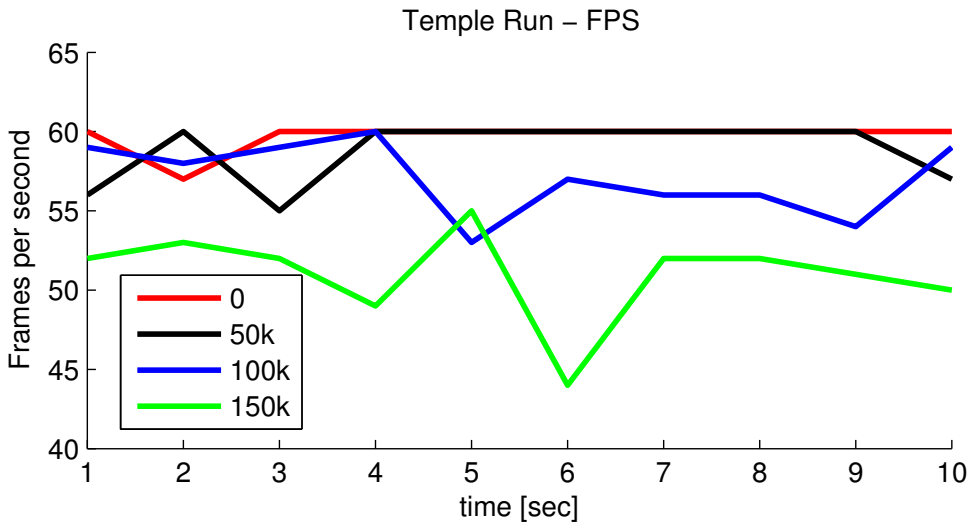

Figure 5.10: VarDroid impact on FPS

expected, the FPS lowers as the INJ input increases. In particular, in a scenario of severe variability impact (bubble length of 150k), user experience is heavily affected, as FPS drops below 50.

Variability Tolerance of the OS: VarDroid can be used to detect limitations and bugs of existing OS design. In this experiment, we focus on the workload migration policy employed in ARM big.LITTLE architectures [3]. In such architecture, a hysteresis mechanism regulates the task migration between LITTLE and big cores, with a high a and a low utilization threshold. When the high threshold is exceeded, the task is migrated to the big core, and is returned to the LITTLE core if utilization decreases below the low threshold. The limitation of such mechanisms is that migration thresholds are fixed. Because of this, the execution can incur significant performance penalties.

To show this, we implemented a single threaded cpu-bound synthetic workload that periodically alternates between three execution phases, respectively with low, medium and high utilization. Then, we execute it with only one LITTLE core and one big core active. The workload is initially allocated on the LITTLE core and execute low and medium phases. The high phase instead triggers the migration to the big core. We execute the program with increasing INJ input, and measure execution times. Figure 5.11a shows the normalized execution times of the medium phase. Results show a penalty of up to 60%. A lower migration threshold in this case would have avoided such

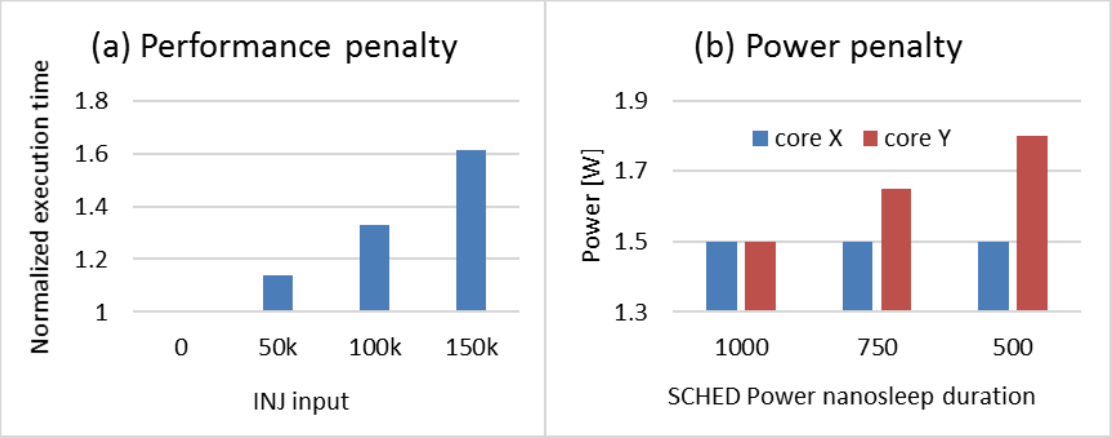

Figure 5.11: Migration (a) performance and (b) power penalty

penalty. The variability tolerance of the migration policy in this case could be improved by changing the migration threshold dynamically. In a scenario in which platforms can actually detect the variability at runtime thanks to dedicated sensors [90], thresholds should be adapted dynamically.

As a final example, we make the case in which the LITTLE core can choose to migrate the workload between two big cores, a core X of the big cluster with nominal power consumption (which is 1.5W from our measurements) and a core Y with higher power consumption due to variability. Results in Figure 5.11b show that a variation-agnostic migration policy can incur in a power penalty of up to 20%.

5.6 Dynamic Variability Management

In this section we employ the VarDroid framework to design and implement a novel Dynamic Variability Management technique. We first explain how variations are modeled for this specific framework, then we present our Dynamic Variability Management and show how it counteracts the various kinds of variations, while meeting the target lifetime.

We target two different kinds of variations which are present in mobile multiprocessors. **Frequency variation** is caused both by static variations in the manufacturing process and by degradation phenomena. The result is that cores of the same multiprocessors actually have different achievable frequency over time.

Static variations affect the way in which the propagation delay degrades. Tools for static variability characterization have been proposed [113], on top of which it is possible to build higher level models [128] for operating frequency, leakage and dynamic power. Static variations have been shown to impact on per core performance [70].

We consider multicores affected by Time Dependent Dielectric Breakdown. The effect of TDDB on transistors propagation delay is modeled in reference [39] as a time dependent gate-to-source resistance R_{BD}. As TDDB takes place, R_{BD} decreases together with the voltage on the gate of the transistor, and the propagation delay increases. The increase in propagation delay, then, causes the achievable operating frequency to decrease. Figure 5.12(a) shows the increase of propagation delay for devices with different degradation rates, but not affected by static variations (e.g. they have the same starting point). In Figure 5.12(b), we highlight the case of devices with the same degradation rate, but affected by static variations (e.g. different starting points). A realistic case, then, will show the combination of both effects.

Recent work in [35] proposes low-area and low-overhead sensors that can be integrated on multicore processors to monitor the achievable frequency. The availability of such devices allows to expose variability to the software stack by detecting the core achievable frequency, and enables runtime management.

Degradation rate variation causes cores of the same multiprocessors to have different lifetimes and different reliability. This is due both to variability in the manufacturing process, to unbalanced workload allocation, and to the impact of environmental conditions, such as ambient temperature. Figure 5.13 shows an example which motivates the use of workload allocation to counteract degradation rate variability. In this example we have to allocate three equal tasks to three cores. In the plots, the x-axis is time, while the y-axis is core reliability. In the first case, no degradation rate variation is present, and the three cores have the same reliability loss. In the second case cores have different degradation rates. In particular, core 1 degrades faster, while core 2 degrades slower with respect to the case with no variations. In this case, a variability-agnostic allocation causes reliability unbalance. Our technique (third case), instead, aims to balance the reliability loss by allocating tasks.

Moreover, considering the model for TDDB presented in [176], we observe the

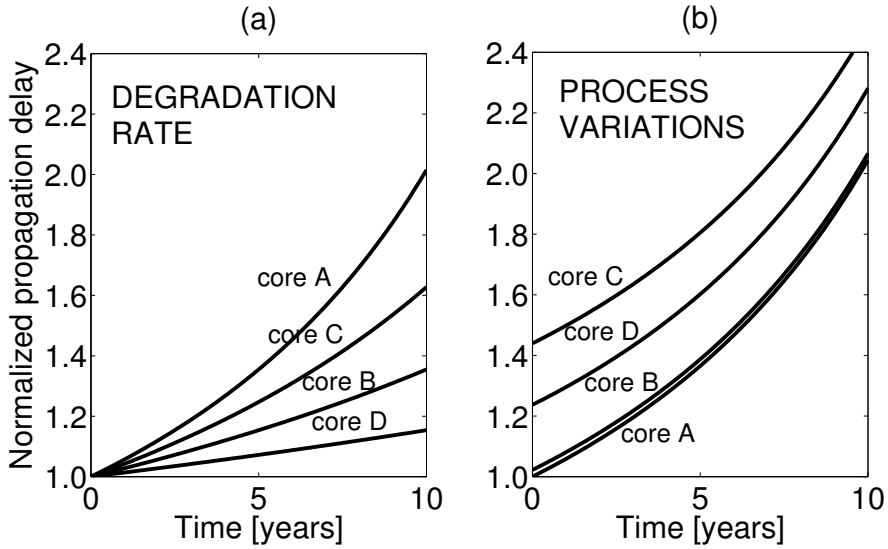

Figure 5.12: Propagation delay degradation under (a) different degradation rates and (b) static variations.

behavior of the scale parameter of the Weibull distribution of TDDB failures. The scale parameter indicates the *characteristic lifetime* of devices subject to TDDB. Figure 5.14 reports the behavior of the characteristic lifetime as temperature and voltage varies across the typical range of a mobile processor. As it can be noticed, when temperature is high, DVFS does not allow for significant reliability saving. This observation allows our technique to achieve even higher performance, as we will discuss in the next subsection.

Figure 5.15 shows the complete DVM framework. Our target platform is a heterogeneous multiprocessor, modeled as an array of cores $\Omega = [\omega_1, \omega_2, ..., \omega_{N_{CORE}}]$. Due to variations, cores have different degradation rates and operating frequencies. Each core is characterized by its *type* $[\sigma_1, \sigma_2, ..., \sigma_{N_{CORE}}]$. Each task can be assigned only to a specific type of cores. Each core has independent voltage and frequency settings. This is common in modern MPSoCs [31]. Moreover, cores have sensors for frequency and degradation monitoring. Tasks are classified based on their quality requirements in *Highly Critical* (H) and *Less critical* (L) as in [110]. H tasks require maximum performance to guarantee user experience.

At the top we have a *degradation monitor*. It activates every *reliability period*

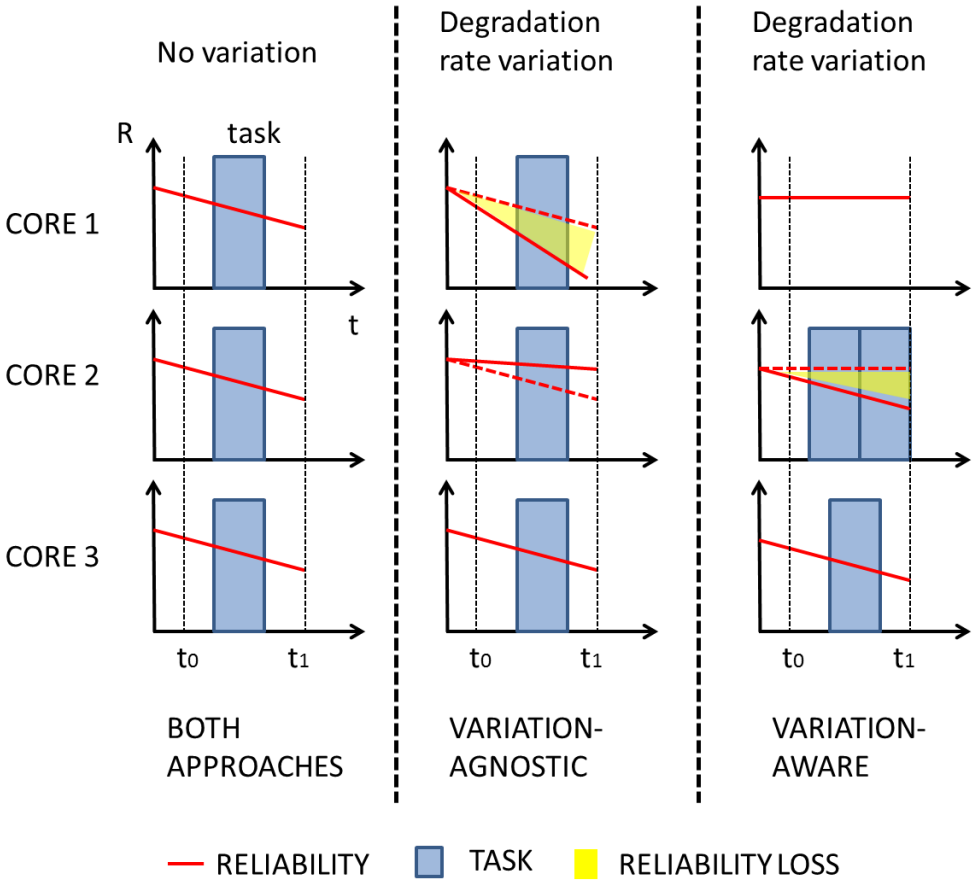

Figure 5.13: Comparison of variation-agnostic and variation-aware task allocation

(in the order of days) and it monitors the degradation status of each core by reading degradation sensors. Given a certain predefined target lifetime, it outputs a reference voltage V_{TARGET} for each core, as they have independent voltage settings. V_{TARGET} is the target average applied voltage for the next reliability period. A similar approach is used in [110] and [105] and has been shown to be effective in guaranteeing the target lifetime. We refer to these works for further details. Note that the effect of different degradation rates will result in different values of V_{TARGET} among cores.

V_{TARGET} is the input to the DVM algorithm, shown into details in Algorithm 1. At each scheduling period, it adjusts frequency and voltage of each core with the following criterion. Given V_{TARGET}, at each scheduling tick the system updates a reference value of voltage V_{REF} as:

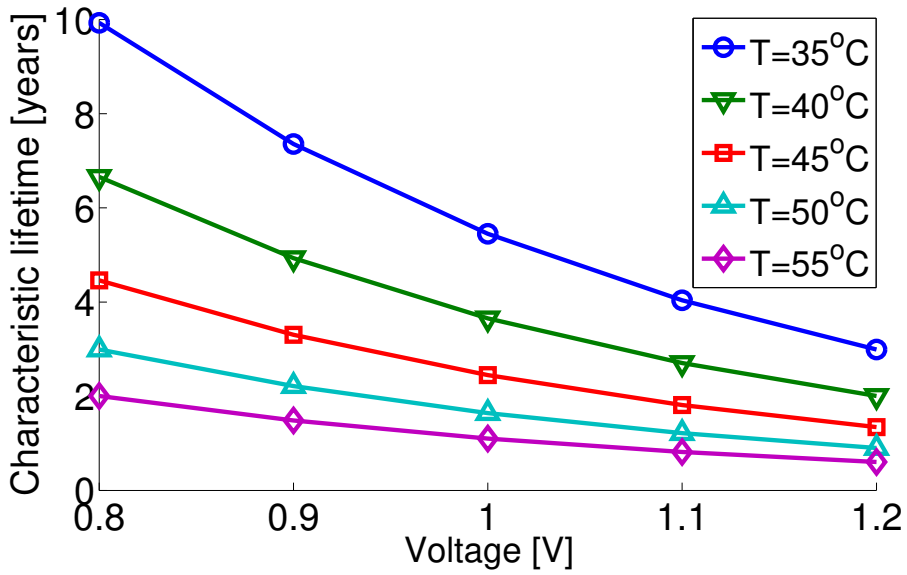

Figure 5.14: Behavior of characteristic lifetime

$$V_{REF} = \frac{V_{TARGET} \cdot \Delta_t - V_{MEAN} \cdot t}{\Delta_t - t} \tag{5.5}$$

Where V_{MEAN} is the mean voltage applied from the beginning of the reliability period, t is the time elapsed from the beginning of the reliability periodm and Δ_t is the duration. This voltage is the key to the *borrowing strategy* described in [110].

T-boost denotes our temperature variation-aware strategy, which is based on the observation that at high temperature lowering the voltage is less beneficial to reliability than at low temperature, as highlighted in the previous subsection. *T-boost* is defined by two temperature thresholds, T_{LOW} and T_{HIGH}. We observe that, from the reliability model defined in [176], running at a temperature lower than T_{LOW} allows to reserve a time budget t_{BOOST}, measured in scheduling periods, for running at maximum voltage/frequency when the temperature is above T_{HIGH}. Our DVM algorithm checks the value of temperature and increases t_{BOOST} by 1 tick if it is lower than T_{LOW}. If the temperature is over T_{HIGH}, it activates *T-boost* and lowers t_{BOOST} by 1 unit. When *T-boost* is active, maximum frequency is selected. Such an approach allows for long term reliability sprinting when the device experiences temperature variations.

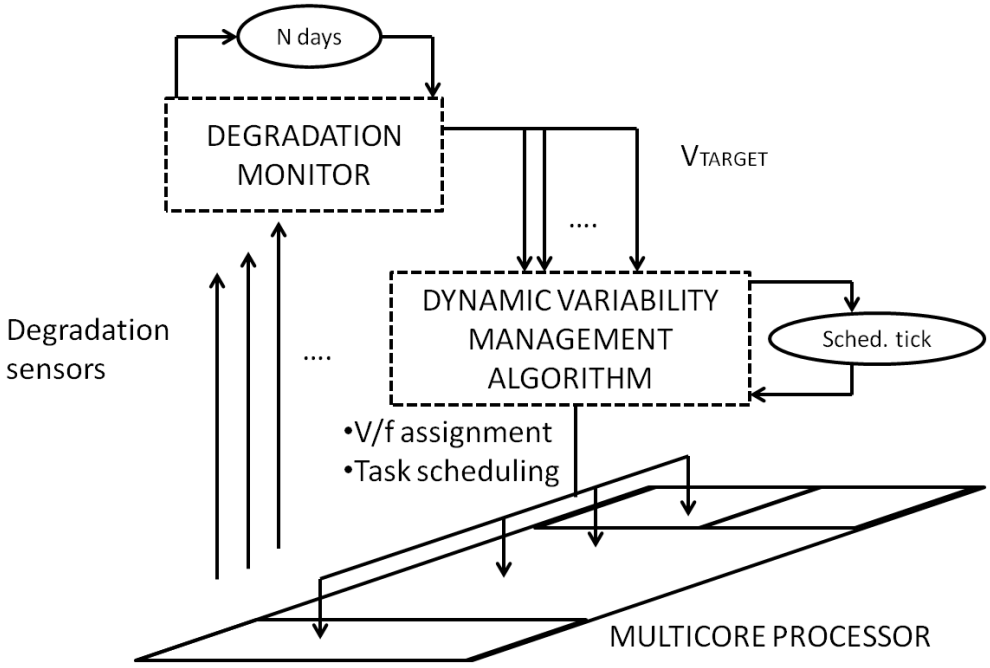

Figure 5.15: DVM framework

If *T-boost* is off, the algorithm selects minimum voltage/frequency if the core is idle, voltage V_{REF} and corresponding frequency if core is running a L task and maximum voltage/frequency if the core is running a H task.

Also, the algorithm takes the first task to be scheduled and considers whether it is H or L. In the first case, maximum performance is required, therefore cores are ranked with respect to their frequency variation. This is possible thanks to the presence of frequency monitors [35]. In this way, the H task is assigned to the core that at each scheduling period can achieve the higher frequency. In the second case, the algorithm tries to balance workload allocation with respect to reliability. In fact, in this case, cores are ranked with respect to V_{REF}. Since V_{REF} keeps track of the reliability borrowing, cores with higher V_{REF} have more reliability margin to spend. Such a formulation allows to counteract frequency and degradation rate variations, it allows to counteract temperature variations, it is lighweight and fully implementable at the OS level.

Algorithm 1 Dynamic Variability Management Algorithm

```
At each scheduling
for each kind of core σ* do
    for each core ω of kind σ* do
        if core is idle then
            V/f(ω) = V/f_MIN
        end if
        if T − boost OFF then
            if executing L then
                V/f(ω) = V/f_REF
            end if
            if executing H then
                V/f(ω) = V/f_MAX
            end if
        else
            V/f(ω) = V/f_MAX
        end if
        ————————
        if new task φ then
            if φ is L then
                rank_{V_REF}(Ω)
                Ω[1] ← φ
            end if
            if φ is H then
                rank_f(Ω)
                Ω[1] ← φ
            end if
        end if
        Update V_REF
    end for
end for
```

5.7 DVM Experimental Evaluation

Our test device is the Qualcomm MSM8660 mobile development smartphone. This device has Snapdragon S3 processor, with a dual Scorpion core and a Adreno 220 GPU, manufactured in 45nm technology. Cores have voltage range from 0.8V to 1.2V and frequency range from 0.81Ghz to 1.18GHz, with fixed V/f operating points. The two cores and the GPU have independent per-core DVFS capability. The smartphone has Android OS 4.0.3, with Linux kernel 3.0.8. Voltage and frequency settings for the GPU are not exposed to the user, therefore in the experiments we can only control voltage and frequency of the two Scorpion cores. To perform experiments, we have implemented testbench applications: Discrete Cosine Transform (DCT), Inverse Discrete Cosine Transform (IDCT) and Image Rotation (ROT), executing on test vectors. These computational kernels are broadly used in many digital processing and imaging applications (such as JPEG decoding) and are a representative, flexible and common workload, useful for performance comparison [128]. The applications have been cross-compiled

to run on the Android environment. Table 5.1 reports respectively the execution time (ET) at minimum and maximum frequency of each application.

Table 5.1: Testbench execution times

	DCT	IDCT	ROT
ET@f_{MIN}	15.13s	10.03s	13.07s
ET@f_{MAX}	5.22s	3.55s	4.16s

In order to implement this policy on an Android system, we have developed a high-level DVFS/scheduling daemon, running in the *userspace*. Figure 5.16 shows the block diagram of our implementation. The high-level daemon exploits the Linux **set-affinity** mechanism to allocate tasks to cores. This mechanism has been adapted to work on an Android environment. Userspace applications have no direct control on voltage and frequency, as these are changed by governors, which are modules of the *kernel space*. Therefore we made the DVFS/scheduling daemon able to communicate with the *userspace governor*, which is a governor that provides a built-in interface to the userspace. An application can issue commands to the userspace governor through the sysfs interface. In this way, the deamon is also able to switch voltage and frequency. The scheduling period is set at 1 second to approximate the Linux reassignment granularity [134]. The daemon reads temperature at each scheduling period from the Linux virtualized *thermal device*, which provides the mean temperature of the chip.

In our experiments, the time of arrival of tasks is modeled as a Poisson distributed random number with mean TA_{MEAN}, as in [63]. We observe four test cases, with different mean time of arrival of executing tasks TA_{MEAN} and different percentage of highly critical tasks $\%H$. The test cases are reported in Table 5.2. The experimental time is a single *reliability period*, with a duration of 500 seconds.

Table 5.2: Test cases

	case 1	case 2	case 3	case 4
$TA_{MEAN}[s]$	50	50	10	10
$\%H$	10	40	10	40

First we compare our policy against a DVM technique which is not subject to

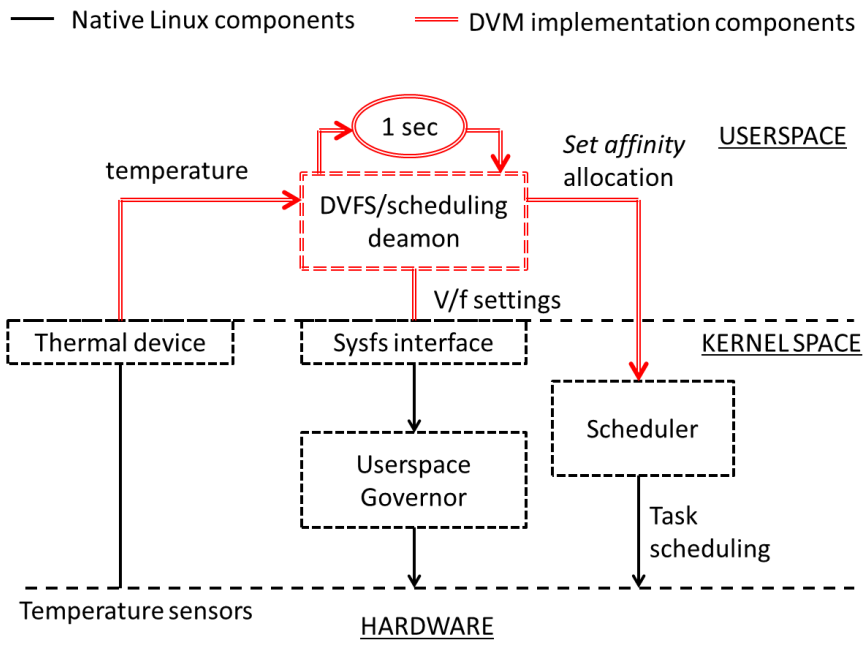

Figure 5.16: Block diagram of DVM implementation

lifetime constraints, such as the one in [128]. This technique performs energy-aware scheduling, by allocating workload on the most power efficient free core. Then, this technique always sets the maximum frequency, in order to provide high performance. Figure 5.17 shows the execution frequencies of the cores in case we apply our policy (*Lifetime Aware*) and in the case in which the *Lifetime Agnostic* policy is applied. Our policy, in the upper half of the figure, modulates the operating voltage/frequency to meet the lifetime constraints, and distributes workload among cores. The other policy, instead, tends to concentrate the workload on the most energy efficient core (core 0 in this example), with the result that the mean applied voltage in the reliability period violates the constraint imposed by V_{TARGET}, which is required to meet the target lifetime.

Then, we compare our DVM technique (*Variations-Aware*) to the dynamic reliability management technique in [110] (*Variations-Agnostic*), in three different scenarios: for a platform affected by variation of degradation rate, for a platform affected by frequency variation and for a platform with both effects. We show that even if both techniques guarantee the target lifetime, our technique achieves best performance because it always assigns tasks to the most performing core. For each case we measure

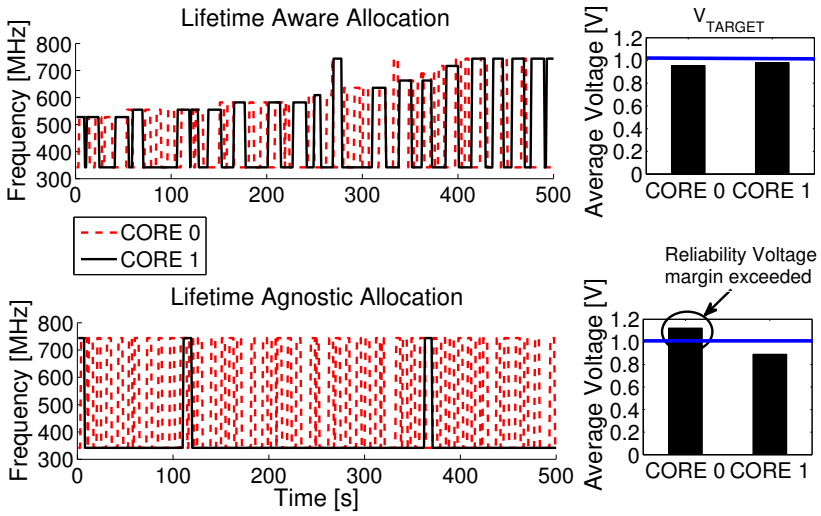

Figure 5.17: Instantaneous and average performance of the lifetime aware allocation and the lifetime agnostic allocation respectively

the mean execution time for the test applications in three different variability conditions, denoted by different values of the ratio σ/μ of the normal distribution characterizing the variations [114].

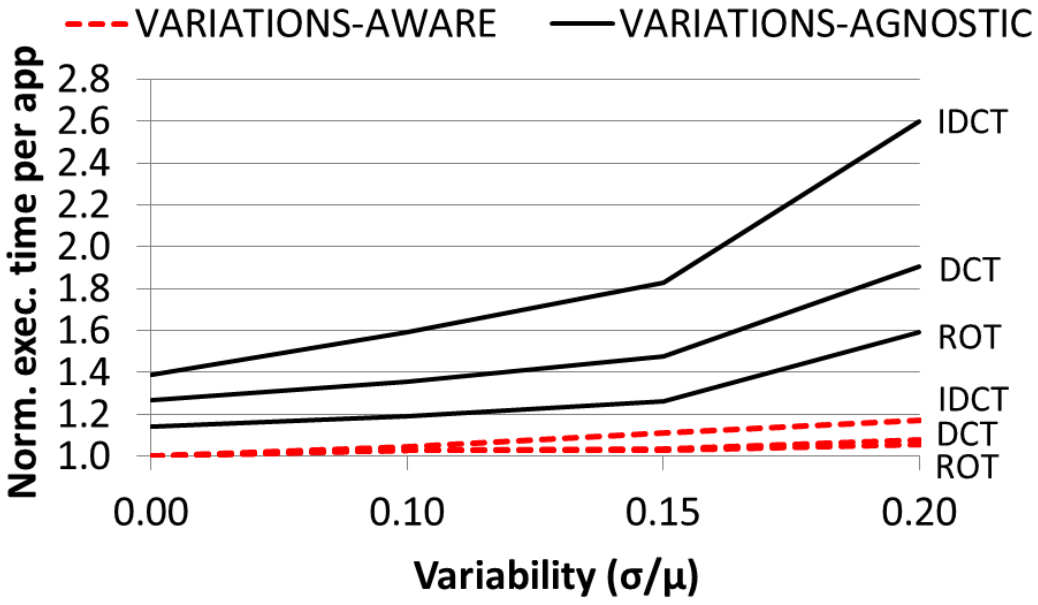

Figure 5.18: Robustness against degradation rate variability

Figure 5.18 shows the results for the comparison in case of a platform with variation on degradation rates, for the test case 1. For each value of the σ/μ ratio we execute the same trace of testbench applications and we collect execution times of each task, both in the *Variations-Aware* and in the *Variations-Agnostic* case. Given this, we compute the performance improvement as the difference between the normalized execution times obtained with the two techniques. The result is that *Variations-Aware* has up to 120% of performance improvement in the case with $\sigma/\mu = 0.2$. Even in the case in which $\sigma/\mu = 0$, our technique performs better. This benefit is given by the scheduling policy of algorithm 3 which assigns tasks to healthier cores first, which also provide higher performance. The variation-agnostic policy, instead, cannot schedule tasks, therefore it does not achieve an optimal result.

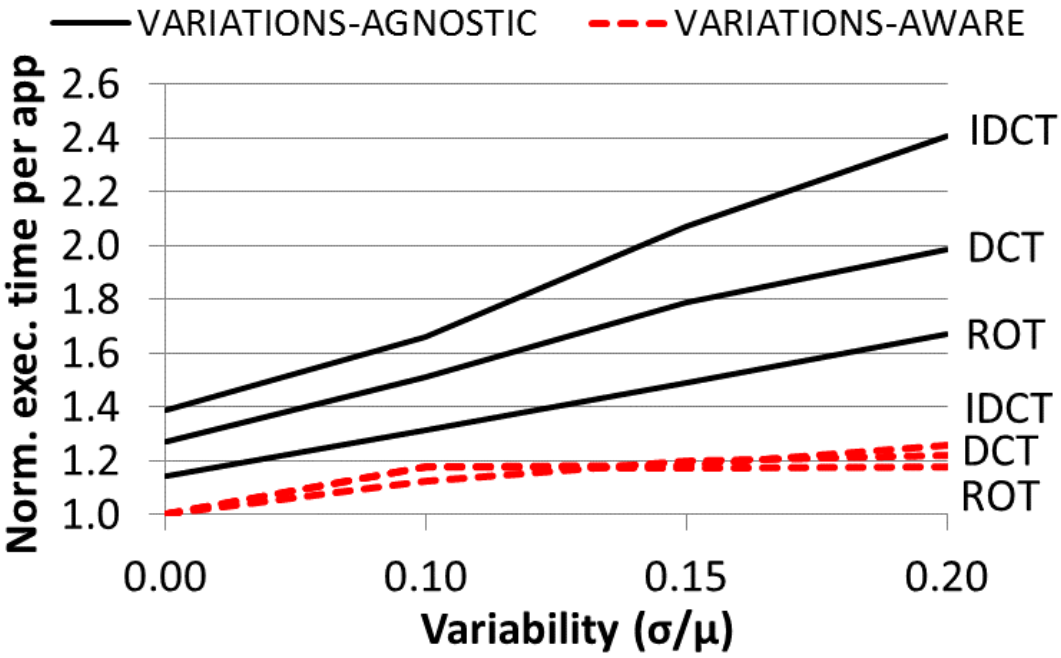

Figure 5.19: Frequency variability

Figure 5.19 shows the case in which the test platform is affected by frequency variations. In this case our variations-aware technique can rank cores with respect to their achievable frequency. Therefore, H tasks are always executed by the core which provides higher frequency.

Table 5.3 reports the mean execution for the IDCT application in the four test cases and for $\sigma/\mu = 0.2$. The higher performance improvement is achieved in case 1. When TA_{MEAN} is higher, in fact, the inefficiency of the variability agnostic technique is more evident, due to the absence of allocation capabilities.

Table 5.3: Normalized mean execution time for IDCT

$\sigma/\mu = 0.2$	case 1	case 2	case 3	case 4
VARIATION-AGNOSTIC	2.75	2.40	1.81	1.98
VARIATION-AWARE	1.11	1.02	1.13	1.14

Figure 5.20, shows the case in which the combined effect of the both variations on frequency and degradation rate are present. The figure also reports the performance improvement for IDCT. In this case, for $\sigma/\mu = 0.2$ we have a performance improvement of 40% for ROT up to 160% for IDCT.

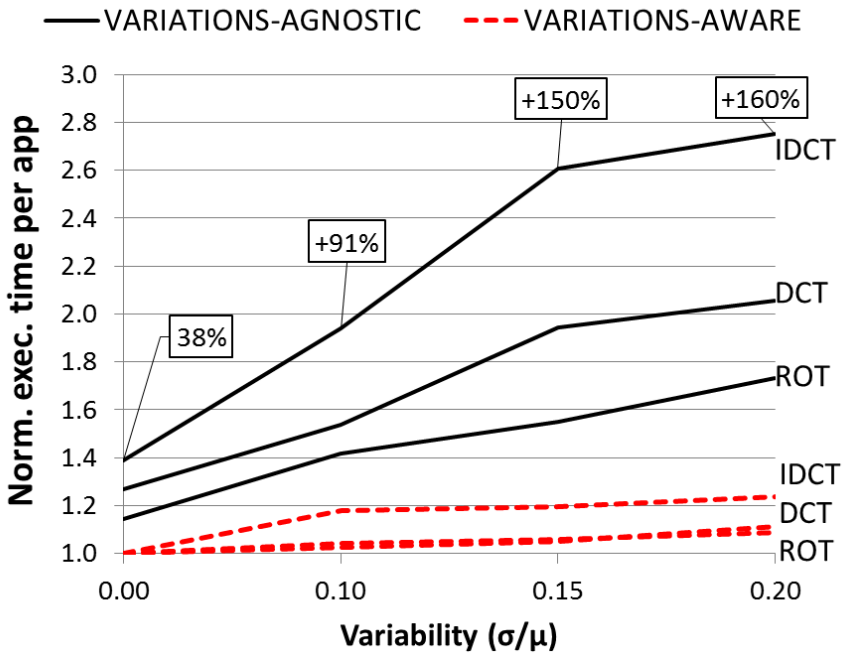

Figure 5.20: Robustness against the combined variations on frequency and degradation rate. Maximum performance improvement per variability level is reported

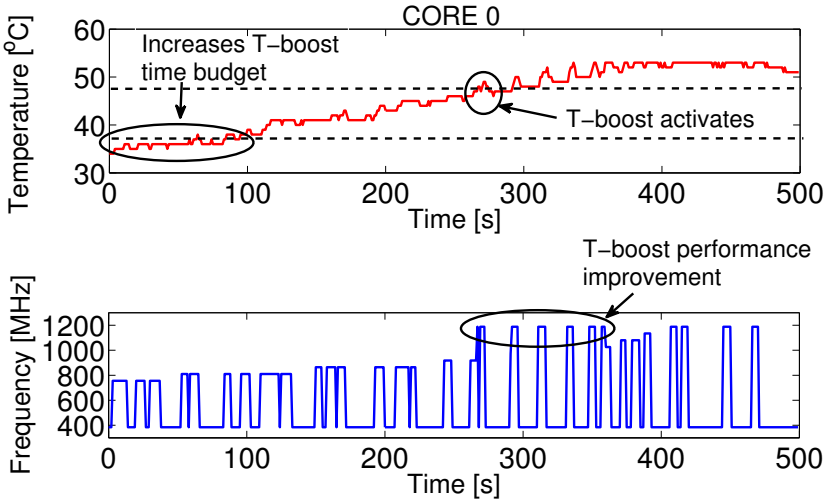

Figure 5.21: T-boost behavior

Figure 5.21, finally, shows the adaptive property of T-boost on temperature variation. In this experiment, T_{LOW} is set to 37^oC while T_{HIGH} to 47^oC. These values have been determined experimentally by measuring the temperature in various workload and environmental conditions. The plot on the top of the figure shows the behavior of temperature over time, while the plot on the bottom shows the execution frequency of CORE 0 (just one core is reported for the sake of clarity). At the beginning, temperature is below T_{LOW}, therefore the time budget t_{BOOST} increases. At 100 seconds, by using a 400w lamp placed at 10 cm to the phone, we start heating up the smartphone which leads to the increase the processor temperature. Figure 5.21 shows that when temperature goes above T_{HIGH}, then *T-boost* activates and the DVM algorithm provides maximum frequency. Note that in a given reliability period the processor experiences very high temperature, the V_{TARGET} for the next period will be lower. Moreover the range of temperature in which T-boost operates (defined by the choice of T_{HIGH} and T_{LOW}) is lower than processor critical temperatures, which are around 100^oC.

5.8 Conclusion

In the previous chapters of this dissertation we proposed power, thermal and reliability management for mobile devices. We assumed that power consumption, performance and degradation rate are the same for all devices and equal to their nominal value. Unfortunately, this is not true in general with the scaling of CMOS technology. In this chapter we also account for variability and integrate it in runtime management. We first presented VarDroid, a tool for power and performance emulation for mobiles, implemented on top of the Android operating system. It is validated with microbenchmarks for performance and power variability. We presented use cases to show how it can be used to analyze the robustness of applications against variability, evaluate the impact of variability on user experience and highlight the limitations of OS design on heterogeneous architectures. Using VarDroid for online emulation, we proposed a novel Dynamic Variability Management technique which leverages a per-core DVFS control along with a multicore task allocation to improve the performance of variability-affected mobile multicore processors while meeting lifetime constraints. The technique has been implemented on a real Android smartphone. The presented experiments show that the proposed DVM achieves up to 160% of performance improvement over state-of-the-art techniques, while meeting the lifetime requirements.

Chapter 5 contains material from “Dynamic Variability Management in Mobile Multicore Processors under Lifetime Constraints”, by Pietro Mercati, Francesco Paterna, Andrea Bartolini, Luca Benini and Tajana Šimunić Rosing, which appears in Proceedings of the 32nd IEEE International Conference on Computer Design (ICCD), Seoul, 2014 [107]. The dissertation author was the primary investigator and author of this paper.

Chapter 5 contains material from “VarDroid: Online Variability Emulation in Android/Linux Platforms”, by Pietro Mercati, Mohsen Imani, Francesco Paterna, Andrea Bartolini, Luca Benini and Tajana Šimunić Rosing, which appears in Proceedings of the 26th edition on Great Lakes Symposium on VLSI (GLSVLSI ’16). ACM, New York, NY, USA [112]. The dissertation author was the primary investigator and author of this paper.

Chapter 6

Conclusion

6.1 Thesis Summary

Today, mobile devices are experiencing an incredible development, and this trend does not seem to stop. They integrate heterogeneous computing units (CPU and GPU), display, speakers, networking and can execute a variety of applications, browsing, multimedia, gaming, beyond traditional phone calls. Unfortunately, despite such amazing development, mobile devices face a set of interrelated problems. To deliver quality experience, mobiles consume a significant amount of power, which can drain the battery in a few hours. High peak power increases the device temperature quickly, which is a source of discomfort for the user. Temperature stress also worsens the impact of transistors and interconnects reliability degradation mechanisms, such as Time Dependent Dielectric Breakdown (TDDB), Negative Bias Temperature Instability (NBTI), Hot Carrier Injection (HCI) and Electromigration (EM). Such mechanisms degrade the performance of circuits over time and eventually lead to hard failures, which are costly to fix and damaging to companies reputation. These problems worsen with the continuous scaling of CMOS technology and the consequent reduction of transistor and interconnect dimensions. This increases the parameter variability between circuits with the same nominal characteristics. As a consequence, devices that are supposed to have the same capabilities, actually have power, performance and degradation rates that are variable. Existing solutions for mobiles considers these aspects separately.

In this dissertation we presented the design and real implementation of a unified

framework for the comprehensive dynamic management of power, temperature, reliability and variability in mobile systems, subject to user experience requirements (see Figure 1.1). We propose a multi-rate **comprehensive management framework** that is composed by two subcontrollers, respectively the **long term controller** and the **short term controller**. This design is motivated by the fact that the time scale of reliability changes and variability effects (on the order of weeks and months) is very different with respect to the time scale of workload, power and temperature changes (on the order of seconds and milliseconds).

In Chapters 2 and 3 we presented solutions for power and thermal management which consider user experience in a complementary way. Chapter 2 aims at maximizing performance subject to a target battery lifetime. We experimentally verify that our strategy can still meets user experience requirements with a selected target battery lifetime extension of at least 25%. Chapter 3 reduces power consumption subject to application-specific performance requirements for user experience. In addition, it leverages a compact thermal model to perform joint power and thermal management. We achieved a 46% application-specific savings on power consumption and up to 35% savings in power consumption at the device level. The techniques presented in these two chapters are part of the short term controller.

Then, in Chapter 4 we presented WARM, a framework for online reliability emulation and leverage it to develop a dynamic reliability management solution. It leverages the major gap in the time scales of reliability changes and workload dynamics by exploring the interaction between long term controller and a short term controller. It integrates a Thermal Controller that allocates tasks to meet thermal constraints. This is required since degradation strongly depends on temperature. We show that WARM meets temperature constraints within 5% in 87.5% more cases than the state-of-the-art. Its task allocation achieves up to 1 year lifetime improvement for a multicore platform. WARM can achieve up to 100% of performance improvement on cluster architectures, such as big.LITTLE, while still guaranteeing the reliability target.

Power, thermal and reliability worsen with variability. Our work takes variability into account for runtime management. In Chapter 5 we first presented VarDroid, a framework for the online emulation of variability on real devices. We presented use

cases to show the utility of VarDroid to test applications, device and OS robustness under the effects of variability. Our results show that a variability-agnostic OS can incur in a performance penalty of up to 60% and a power penalty of up to 20 Then, we used VarDroid to develop a novel technique for dynamic variability management, which improves the long-short term controller interaction to improve performance. The proposed DVM solution uses sensors to monitor the variable operating conditions and the degradation rate. We implement our algorithm in Android OS on a mobile phone and show that it achieves up to 160% performance improvement over the state-of-the-art while meeting the lifetime constraints.

6.2 Future Work Directions

The problems of power, temperature, reliability and variability will be even more relevant for the Internet of Things era. Solutions developed for mobiles need to be adapted to the distributed nature of the IoT to guarantee scalability, modularity and extendability [57]. In the recent years, smart devices outnumbered human beings. There will be 50 billions of interconnected devices in 2020 [133], with an average of 6.5 devices per person. Recent studies say that the IoT market is expected to grow from $157.05 billion in 2016 to $662 billion in 2021, at a Compound Annual Growth Rate (CAGR) of nearly 33 percent during that time interval [9]. The IoT spans across applications in every field of human life: industry, environment, agriculture, smart cities, smart homes, healthcare and more [14]. In every application, the goal of IoT is to deliver Quality of Service (QoS) to the user, which can be defined in terms of accuracy, availability, stability and user satisfaction. The IoT faces dramatic challenges that put limitations to its growth. It is built by different devices coming from different vendors, so standardization is required for seamless integration. The large amount of data flowing also poses problems of interoperability, data collection and analysis. The communication of sensitive data creates problems related to security and privacy. A large part of IoT devices is powered by batteries that need to be frequently changed or recharged, making energy efficiency a primary requirement. Moreover, the large networks of devices composing the IoT require continuous maintenance for the replacement of defective parts,

with high associated costs [16].

Most of IoT applications are user-focused, hence delivering quality user experience is fundamental. The assumption of this dissertation is that user experience can be embedded in a certain required level of performance which is dictated by the level of expectation of the user with respect to different applications (see Chapter 3). In addition, we assumed that tasks belonging to the foreground application require higher performance with respect to background tasks, since foreground applications are those focusing user's attention (see Chapter 4 and 5). An automated framework for the online identification of user's requirements is a necessity for future IoT devices.

Both IoT and mobile devices integrate more heterogeneous components and can use multiple controls. The control algorithms can be improved by leveraging control theory for the management of multiple heterogeneous variables and for the design of digital controllers that guarantee stability and convergence [50].

Finally, energy efficiency of both IoT and mobile devices can be further improved with Approximate Computing [117]. Approximate Computing relies on the tolerance of systems and users to some loss of quality in a range of applications, and relaxes the need for fully precise operations, with the effect of enabling higher energy savings. Compared to the traditional design space, characterized by the two dimensions of energy consumption and performance, approximate computing introduces a third dimension, which is error (or, conversely, accuracy). Approximation has been extensively used in areas such as lossy compression and numeric computation [161] and today it is gaining a lot of success in many research areas, from circuit and hardware, to applications, compilers, OS and programming languages.

6.2.1 Dynamic Management of the IoT Infrastructure

Dynamic management of systems has been used extensively for energy, thermal and reliability management in traditional devices such as personal computers, mobiles and data centers [21], [110]. A flexible framework for IoT energy and reliability management is still missing. Future work should address these two key problems in IoT: energy efficiency and reliability. For this, it is necessary to formulate the problem of managing the IoT networks while reducing energy consumption subject to QoS and

reliability constraints. A framework to solve this problem should rely on model identification techniques to extract reference values and employ convex optimization to adjust control decisions. This requires developing models for reliability, power, quality of experience/service and control decisions for the IoT, which can be efficiently executed at runtime. All these models should be a function of a set of utilization metrics, which are parameters that strongly influence the power consumption of the system. These are, for example, the activity ratio of the device, the residency of processor power states, operating frequency, transmitting and receiving rates. Once the models are available, we can formulate an optimization problem in which we minimize power/energy consumption subject to quality of experience and reliability constraint in variability-affected devices.

6.2.2 Automated Detection of User Preferences

Maintaining a target quality of experience is the goal of IoT applications. Since the IoT is a user-centric paradigm, the target is defined as the minimum performance that meets user expectation and provides quality experience. For each application and for each IoT network, the quality should be modeled and expressed as a function of the relevant utilization metrics. To build such a function, we need user feedback at runtime. This can be obtained directly from users, with online evaluations on personal mobile devices.

An alternative is to perform automatic recognition of facial expressions and emotions. This task can be accomplished by a few recently developed algorithms. Work in [20], [100] presents a comparison of machine learning methods applied to fully automatic recognition of facial expressions. Work in [146] accomplishes emotion recognition by combining multiple visual descriptors with paralinguistic audio features for multimodal classification of video clips. The features extracted are combined through Multiple Kernel Learning and the clips are classified using an SVM into one of the seven emotion categories: Anger, Disgust, Fear, Happiness, Neutral, Sadness and Surprise. Solutions for automatic expression recognition have been implemented and release in the form of free software [101]. Facial expressions can be effectively used to infer user emotion and, therefore, to collect automatic feedback on the level of user satisfaction.

Work in reference [155] presents a mobile application for real time facial expres-

sion recognition running on a smartphone with a camera. It can automatically classify human emotions. Such information can be used to train a user experience model that can be leveraged for runtime optimization.

We also assumed that the desired battery lifetime is set by the user (see Chapter 2). Future work should automatically predict the next recharging opportunity [18], [135]. This can be achieved initially by deriving models based on existing datasets [144]. Such models can then be refined with online training and leveraged at runtime to predict the next battery recharge.

For temperature, we assumed that there are thermal threshold which constraint the optimization problems for runtime management. These thresholds implicitly account for skin contact temperature which can be a source of discomfort if it exceeds 45^{o}C. Future work should address an automatic solution to derive thermal threshold based on thermal model identification [130].

6.2.3 Optimization of Heterogeneous Control Variables

Modern processors for both IoT and mobile devices have heterogeneous control variables with different activation rate due to hardware limitations. For example, while frequency scaling is actuated in less than 1 millisecond, power gating for switching cores on and off might require tens or even hundreds of milliseconds. A single rate controller might not be able to adapt to fast changing workload requirements, if too slow, while it might loose optimization opportunity coming from slowly changing variable, if too fast. A promising solution is represented by the use of multi-rate controllers.

Multirate controllers are able to outperform single-rate linear time-invariant controllers thanks to their time-varying nature, but they might need powerful processors for online computation. For implementing on a real system, it is essential to obtain a computationally efficient multirate digital control scheme [91]. A multirate control structure accommodates multiple information available at different rates and implement the required control computations within the limited capabilities of online management in an operating system [53]. A control-theoretic approach is required for the distributed nature of the IoT and to guarantee scalability.

6.2.4 Approximate Computing

The availability of new hardware components with novel control variables offers new possibilities for dynamic management at the operating system level. For example, associative resistive memories have been recently employed to achieve in-memory approximate computing to improve energy efficiency [72]. Resistive configurable associative memories can be leveraged in computation with GPU and accelerators to quickly retrieve the result of computations and improve performance. These can be configured using selective voltage over-scaling to accept 1 or 2-bit hamming distance error and improve energy efficiency [73]. Also, the idea of approximate computing can be applied to storage. In hardware approximate storage can be implemented with relaxed retention time NVM [151] or fast read and write operation [140]. Therefore, approximation can be adapted at runtime by using hardware controls. Future work should first investigate the relation of approximation with power, temperature, reliability, variability and user experience. Intuitively, increasing the level of approximation helps reducing the power consumption and consequently temperature and reliability degradation. However, a high level of approximation might compromise user experience. Variability will impact the hardware components that tune approximation. The models that describe the impact of approximation can then be used to implement a comprehensive management solution on a broad range of applications. This can be achieved through the development of a novel approximate computing architecture for mobiles. The framework has two major modules which are part of the system software and enable approximate computing with existing mobile operating systems. The first module is a user perception management service, which performs user perception estimation and user perception-aware display management. It can be applied to both system-level and user-level programs. The second module is an approximate computing service, which dynamically adapts running applications to leverage the approximate computing opportunities offered by the underlying hardware and improve energy efficiency, while meeting the user experience target.

Bibliography

[1] Android ndk, https://developer.android.com/tools/sdk/ndk/index.html.

[2] Antutu benchmark, http://www.antutu.com/en/ranking.shtml.

[3] Arm white paper. 2014. big.little technology: The future of mobile.

[4] Google nexus 5, http://en.wikipedia.org/wiki/nexus_5.

[5] List of most downloaded android applications, http://en.wikipedia.org/wiki/list_of_most_downloaded_android_applications.

[6] Qualcomm snapdragon s4, https://www.qualcomm.com/products/snapdragon/processors/s4-s1.

[7] Temple run, https://en.wikipedia.org/wiki/temple_run.

[8] Itrs reports, december 2014, http://www.itrs.net. 2014.

[9] Internet of things market, http://www.marketsandmarkets.com/market-reports/internet-of-things-market-573.html. 2016.

[10] A. Agarwal, D. Blaauw, and V. Zolotov. Statistical timing analysis for intra-die process variations with spatial correlations. In *Computer Aided Design, 2003. ICCAD-2003. International Conference on*, pages 900–907, Nov 2003.

[11] F. Alam, P.R. Panda, N. Tripathi, N. Sharma, and S. Narayan. Energy optimization in android applications through wakelock placement. In *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014*, pages 1–4, March 2014.

[12] S. Alawnah and A. Sagahyroon. Modeling smartphones power. In *EUROCON, 2013 IEEE*, pages 369–374, July 2013.

[13] Bhojan Anand, Karthik Thirugnanam, Jeena Sebastian, Pravein G. Kannan, Akhihebbal L. Ananda, Mun Choon Chan, and Rajesh Krishna Balan. Adaptive display power management for mobile games. In *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*, MobiSys '11, pages 57–70, New York, NY, USA, 2011. ACM.

[14] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Comput. Netw.*, 54(15):2787–2805, October 2010.

[15] G. Bai, H. Mou, Y. Hou, Y. Lyu, and W. Yang. Android power management and analyses of power consumption in an android smartphone. In *High Performance Computing and Communications 2013 IEEE International Conference on Embedded and Ubiquitous Computing (HPCC and EUC), 2013 IEEE 10th International Conference on*, pages 2347–2353, Nov 2013.

[16] Bharathan Balaji, Jian Xu, Anthony Nwokafor, Rajesh Gupta, and Yuvraj Agarwal. Sentinel: Occupancy based hvac actuation using existing wifi infrastructure within commercial buildings. In *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*, SenSys '13, pages 17:1–17:14, New York, NY, USA, 2013. ACM.

[17] Barbara Ballard. *Designing the Mobile User Experience*. John Wiley & Sons, 2007.

[18] Nilanjan Banerjee, Ahmad Rahmati, Mark D. Corner, Sami Rollins, and Lin Zhong. Users and batteries: Interactions and adaptive energy management in mobile systems. In *Proceedings of the 9th International Conference on Ubiquitous Computing*, UbiComp '07, pages 217–234, Berlin, Heidelberg, 2007. Springer-Verlag.

[19] A. Banitalebi-Dehkordi, M. T. Pourazad, and P. Nasiopoulos. Effect of high frame rates on 3d video quality of experience. In *2014 IEEE International Conference on Consumer Electronics (ICCE)*, pages 416–417, Jan 2014.

[20] M. S. Bartlett, G. Littlewort, M. Frank, C. Lainscsek, I. Fasel, and J. Movellan. Recognizing facial expression: machine learning and application to spontaneous behavior. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 2, pages 568–573 vol. 2, June 2005.

[21] A. Bartolini, M. Cacciari, A. Tilli, and L. Benini. Thermal and energy management of high-performance multicores: Distributed and self-calibrating model-predictive controller. *Parallel and Distributed Systems, IEEE Transactions on*, 24(1):170–183, 2013.

[22] F. Beneventi, A. Bartolini, A. Tilli, and L. Benini. An effective gray-box identification procedure for multicore thermal modeling. *IEEE Transactions on Computers*, 63(5):1097–1110, May 2014.

[23] L. Benini, A. Bogliolo, and G. De Micheli. A survey of design techniques for system-level dynamic power management. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 8(3):299–316, June 2000.

[24] J. Blome, Shuguang Feng, S. Gupta, and S. Mahlke. Self-calibrating online wearout detection. In *Microarchitecture, 2007. MICRO 2007. 40th Annual IEEE/ACM International Symposium on*, pages 109 –122, dec. 2007.

[25] C. Bolchini, M. Carminati, A. Miele, A. Das, A. Kumar, and B. Veeravalli. Run-time mapping for reliable many-cores based on energy/performance trade-offs. In *Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), 2013 IEEE International Symposium on*, pages 58–64, Oct 2013.

[26] A. Bonetto, M. Ferroni, D. Matteo, A. A. Nacci, M. Mazzucchelli, D. Sciuto, and M. D. Santambrogio. Mpower: Towards an adaptive power management system for mobile devices. In *Computational Science and Engineering (CSE), 2012 IEEE 15th International Conference on*, pages 318–325, Dec 2012.

[27] S. Borkar. Designing reliable systems from unreliable components: the challenges of transistor variability and degradation. *Micro, IEEE*, 25(6):10 – 16, nov.-dec. 2005.

[28] K. A. Bowman, A. R. Alameldeen, S. T. Srinivasan, and C. B. Wilkerson. Impact of die-to-die and within-die parameter variations on the clock frequency and throughput of multi-core processors. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 17(12):1679–1690, Dec 2009.

[29] K.A. Bowman, A.R. Alameldeen, S.T. Srinivasan, and C.B. Wilkerson. Impact of die-to-die and within-die parameter variations on the throughput distribution of multi-core processors. In *Low Power Electronics and Design (ISLPED), 2007 ACM/IEEE International Symposium on*, pages 50 –55, aug. 2007.

[30] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, New York, NY, USA, 2004.

[31] Benton H. Calhoun and Kyle Craig. Flexible on-chip power delivery for energy efficient heterogeneous systems. In *Proceedings of the 50th Annual Design Automation Conference*, DAC '13, pages 160:1–160:6, New York, NY, USA, 2013. ACM.

[32] B.H. Calhoun and K. Craig. Flexible on-chip power delivery for energy efficient heterogeneous systems. In *Design Automation Conference (DAC), 2013 50th ACM / EDAC / IEEE*, pages 1–6, May 2013.

[33] Aaron Carroll and Gernot Heiser. An analysis of power consumption in a smartphone. In *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference*, USENIXATC'10, pages 21–21, Berkeley, CA, USA, 2010. USENIX Association.

[34] Aaron Carroll and Gernot Heiser. An analysis of power consumption in a smartphone. In *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference*, USENIXATC'10, pages 21–21, Berkeley, CA, USA, 2010. USENIX Association.

[35] Tuck-Boon Chan, P. Gupta, A.B. Kahng, and Liangzhen Lai. Ddro: A novel performance monitoring methodology based on design-dependent ring oscillators. In *Quality Electronic Design (ISQED), 2012 13th International Symposium on*, pages 633 –640, march 2012.

[36] Hung-Ching Chang, A.R. Agrawal, and K.W. Cameron. Energy-aware computing for android platforms. In *Energy Aware Computing (ICEAC), 2011 International Conference on*, pages 1–4, Nov 2011.

[37] Y. Chen, X. Chen, M. Zhao, and C. J. Xue. Mobile devices user - the subscriber and also the publisher of real-time oled display power management plan. In *2012 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 687–690, Nov 2012.

[38] Bachir Chihani, Khalil ur Rehman Laghari, Emmanuel Bertin, Denis Collange, Noel Crespi, and TiagoH. Falk. User-centric quality of experience measurement. In Gerard Memmi and Ulf Blanke, editors, *Mobile Computing, Applications, and Services*, volume 130 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 33–46. Springer International Publishing, 2014.

[39] M. Choudhury, V. Chandra, K. Mohanram, and R. Aitken. Analytical model for tddb-based performance degradation in combinational logic. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, pages 423 –428, march 2010.

[40] A. Ciancio, J. F. L. de Oliveira, F. M. L. Ribeiro, E. A. B. da Silva, and A. Said. Quality perception in 3d interactive environments. In *2013 IEEE International Symposium on Circuits and Systems (ISCAS2013)*, pages 9–12, May 2013.

[41] Ayse Kivilcim Coskun, Tajana Simunic Rosing, and Kenny C. Gross. Proactive temperature management in mpsocs. In *Proceedings of the 2008 International Symposium on Low Power Electronics & Design*, ISLPED '08, pages 165–170, New York, NY, USA, 2008. ACM.

[42] A. Das, A. Kumar, and B. Veeravalli. Reliability and energy-aware mapping and scheduling of multimedia applications on multiprocessor systems. *IEEE Transactions on Parallel and Distributed Systems*, 27(3):869–884, March 2016.

[43] Anup Das, Bashir M. Al-Hashimi, and Geoff V. Merrett. Adaptive and hierarchical runtime manager for energy-aware thermal management of embedded systems. *ACM Trans. Embed. Comput. Syst.*, 15(2):24:1–24:25, January 2016.

[44] R. Degraeve, N. Pangon, B. Kaczer, T. Nigam, G. Groeseneken, and A. Naem. Temperature acceleration of oxide breakdown and its impact on ultra-thin gate oxide reliability. In *VLSI Technology, 1999. Digest of Technical Papers. 1999 Symposium on*, pages 59 –60, 1999.

[45] Chenwei Deng, Lin Ma, Weisi Lin, and King Ngi Ngan. *Visual Signal Quality Assessment: Quality of Experience (QoE)*. Springer Publishing Company, Incorporated, 2014.

[46] B. Dietrich and S. Chakraborty. Managing power for closed-source android os games by lightweight graphics instrumentation. In *Network and Systems Support for Games (NetGames), 2012 11th Annual Workshop on*, pages 1–3, Nov 2012.

[47] Benedikt Dietrich and Samarjit Chakraborty. Power management using game state detection on android smartphones. In *Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '13, pages 493–494, New York, NY, USA, 2013. ACM.

[48] James Donald and Margaret Martonosi. Techniques for multicore thermal management: Classification and new exploration. In *Proceedings of the 33rd Annual International Symposium on Computer Architecture*, ISCA '06, pages 78–88, Washington, DC, USA, 2006. IEEE Computer Society.

[49] Hossein Falaki, Ratul Mahajan, Srikanth Kandula, Dimitrios Lymberopoulos, Ramesh Govindan, and Deborah Estrin. Diversity in smartphone usage. In *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, MobiSys '10, pages 179–194, New York, NY, USA, 2010. ACM.

[50] Gene F. Franklin, Michael L. Workman, and Dave Powell. *Digital Control of Dynamic Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 3rd edition, 1997.

[51] Siddharth Garg and Diana Marculescu. System-level throughput analysis for process variation aware multiple voltage-frequency island designs. *ACM Trans. Des. Autom. Electron. Syst.*, 13(4):59:1–59:25, October 2008.

[52] J. George, B. Marr, B. E. S. Akgul, and K. V. Palem. Probabilistic arithmetic and energy efficient embedded signal processing. In *Proceedings of the 2006 International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, CASES '06, pages 158–168, New York, NY, USA, 2006. ACM.

[53] D. Glasson. Development and applications of multirate digital control. *IEEE Control Systems Magazine*, 3(4):2–8, November 1983.

[54] Lorenzo Gomez, Iulian Neamtiu, Tanzirul Azim, and Todd Millstein. Reran: Timing- and touch-sensitive record and replay for android. In *Proceedings of the*

2013 International Conference on Software Engineering, ICSE '13, pages 72–81, Piscataway, NJ, USA, 2013. IEEE Press.

[55] Michael Grant and Stephen Boyd. Graph implementations for nonsmooth convex programs. In V. Blondel, S. Boyd, and H. Kimura, editors, *Recent Advances in Learning and Control*, Lecture Notes in Control and Information Sciences, pages 95–110. Springer-Verlag Limited, 2008.

[56] Michael Grant and Stephen Boyd. CVX: Matlab software for disciplined convex programming, version 2.1, March 2014.

[57] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of things (iot): A vision, architectural elements, and future directions. *Future Gener. Comput. Syst.*, 29(7):1645–1660, September 2013.

[58] P. Gupta, Y. Agarwal, L. Dolecek, N. Dutt, R. K. Gupta, R. Kumar, S. Mitra, A. Nicolau, T. S. Rosing, M. B. Srivastava, S. Swanson, and D. Sylvester. Underdesigned and opportunistic computing in presence of hardware variability. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 32(1):8 –23, jan. 2013.

[59] Pearl S. Guterman, Kazuho Fukuda, Laurie M. Wilcox, and Robert S. Allison. 75.3: Is brighter always better? the effects of display and ambient luminance on preferences for digital signage. *SID Symposium Digest of Technical Papers*, 41(1):1116–1119, 2010.

[60] A. Haggag, M. Moosa, N. Liu, D. Burnett, G. Abeln, M. Kuffler, K. Forbes, P. Schani, M. Shroff, M. Hall, C. Paquette, G. Anderson, D. Pan, K. Cox, J. Higman, M. Mendicino, and S. Venkatesan. Realistic projections of product fails from nbti and tddb. In *Reliability Physics Symposium Proceedings, 2006. 44th Annual., IEEE International*, pages 541–544, March 2006.

[61] V. Hanumaiah and S. Vrudhula. Reliability-aware thermal management for hard real-time applications on multi-core processors. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2011*, pages 1 –6, march 2011.

[62] Vinay Hanumaiah, Digant Desai, Benjamin Gaudette, Carole-Jean Wu, and Sarma Vrudhula. STEAM: A smart temperature and energy aware multicore controller. *ACM Trans. Embed. Comput. Syst.*, 13(5s):151:1–151:25, October 2014.

[63] Yuxiong He, Sameh Elnikety, and Hongyang Sun. Tians scheduling: Using partial processing in best-effort applications. In *Proceedings of the 2011 31st International Conference on Distributed Computing Systems*, ICDCS '11, pages 434–445, Washington, DC, USA, 2011. IEEE Computer Society.

[64] Jörg Henkel, Lars Bauer, Nikil Dutt, Puneet Gupta, Sani Nassif, Muhammad Shafique, Mehdi Tahoori, and Norbert Wehn. Reliable on-chip systems in the nano-era: Lessons learnt and future trends. In *Proceedings of the 50th Annual Design Automation Conference*, DAC '13, pages 99:1–99:10, New York, NY, USA, 2013. ACM.

[65] C. Hu. Future cmos scaling and reliability. *Proceedings of the IEEE*, 81(5):682–689, May 1993.

[66] Chenming Hu. Gate oxide scaling limits and projection. In *Electron Devices Meeting, 1996. IEDM '96., International*, pages 319 –322, dec. 1996.

[67] Lin Huang, Feng Yuan, and Qiang Xu. On task allocation and scheduling for lifetime extension of platform-based mpsoc designs. *Parallel and Distributed Systems, IEEE Transactions on*, 22(12):2088–2099, Dec 2011.

[68] W. Huang, K. Sankaranarayanan, R. J. Ribando, M. R. Stan, and K. Skadron. An Improved Block-based Thermal Model in Hotspot 4.0 with Granularity Considerations. In *Proc. WDDD*, 2007.

[69] V. Huard, C. Parthasarathy, N. Rallet, C. Guerin, M. Mammase, D. Barge, and C. Ouvrard. New characterization and modeling approach for nbti degradation from transistor to product level. In *Electron Devices Meeting, 2007. IEDM 2007. IEEE International*, pages 797 –800, dec. 2007.

[70] Eric Humenay, David Tarjan, and Kevin Skadron. Impact of process variations on multicore performance symmetry. In *Proceedings of the conference on Design, automation and test in Europe*, DATE '07, pages 1653–1658, San Jose, CA, USA, 2007. EDA Consortium.

[71] S. Ickin, K. Wac, M. Fiedler, L. Janowski, Jin-Hyuk Hong, and A.K. Dey. Factors influencing quality of experience of commonly used mobile applications. *Communications Magazine, IEEE*, 50(4):48–56, April 2012.

[72] M. Imani, P. Mercati, and T. Rosing. Remam: Low energy resistive multi-stage associative memory for energy efficient computing. In *2016 17th International Symposium on Quality Electronic Design (ISQED)*, pages 101–106, March 2016.

[73] M. Imani, A. Rahimi, and T. S. Rosing. Resistive configurable associative memory for approximate computing. In *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1327–1332, March 2016.

[74] N. Islam and R. Want. Smartphones: Past, present, and future. *IEEE Pervasive Computing*, 13(4):89–92, Oct 2014.

[75] ITRS. Report. *Online, http://www.itrs2.net/2013-itrs.html*, 2013.

[76] JEDEC Solid State Technology Association. Failure mechanisms and models for semiconductor devices. Technical Report JEP122C, March 2006.

[77] Shubham Kamdar and Neha Kamdar. Article: big.little architecture: Heterogeneous multicore processing. *International Journal of Computer Applications*, 119(1):35–38, June 2015. Full text available.

[78] E. Karl, D. Blaauw, D. Sylvester, and T. Mudge. Reliability modeling and management in dynamic microprocessor-based systems. In *Design Automation Conference, 2006 43rd ACM/IEEE*, pages 1057 –1060, 0-0 2006.

[79] E. Karl, D. Blaauw, D. Sylvester, and T. Mudge. Multi-mechanism reliability modeling and management in dynamic systems. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 16(4):476–487, April 2008.

[80] E. Karl, P. Singh, D. Blaauw, and D. Sylvester. Compact in-situ sensors for monitoring negative-bias-temperature-instability effect and oxide degradation. In *2008 IEEE International Solid-State Circuits Conference - Digest of Technical Papers*, pages 410–623, Feb 2008.

[81] U. R. Karpuzcu, K. B. Kolluru, N. S. Kim, and J. Torrellas. Varius-ntv: A microarchitectural model to capture the increased sensitivity of manycores to process variations at near-threshold voltages. In *Dependable Systems and Networks (DSN), 2012 42nd Annual IEEE/IFIP International Conference on*, pages 1–11, June 2012.

[82] J. Keane, D. Persaud, and C. H. Kim. An all-in-one silicon odometer for separately monitoring hci, bti, and tddb. In *2009 Symposium on VLSI Circuits*, pages 108–109, June 2009.

[83] Dongwon Kim, Nohyun Jung, and Hojung Cha. Content-centric display energy management for mobile devices. In *Proceedings of the 51st Annual Design Automation Conference*, DAC '14, pages 41:1–41:6, New York, NY, USA, 2014. ACM.

[84] J.M. Kim, Y.G. Kim, and S.W. Chung. Stabilizing cpu frequency and voltage for temperature-aware dvfs in mobile devices. *Computers, IEEE Transactions on*, 64(1):286–292, Jan 2015.

[85] N. S. Kim, T. Austin, D. Baauw, T. Mudge, K. Flautner, J. S. Hu, M. J. Irwin, M. Kandemir, and V. Narayanan. Leakage current: Moore's law meets static power. *Computer*, 36(12):68–75, Dec 2003.

[86] Sangwook Kim, Hwanju Kim, Jeaho Hwang, Joonwon Lee, and Euiseong Seo. An event-driven power management scheme for mobile consumer electronics. *Consumer Electronics, IEEE Transactions on*, 59(1):259–266, February 2013.

[87] Yeseong Kim, Francesco Paterna, Sameer Tilak, and Tajana S. Rosing. Smartphone analysis and optimization based on user activity recognition. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '15, pages 605–612, Piscataway, NJ, USA, 2015. IEEE Press.

[88] V. J. Kozhikkottu, R. Venkatesan, A. Raghunathan, and S. Dey. Vespa: Variability emulation for system-on-chip performance analysis. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2011*, pages 1–6, March 2011.

[89] Young-Woo Kwon and Eli Tilevich. Reducing the energy consumption of mobile applications behind the scenes. In *Proceedings of the 2013 IEEE International Conference on Software Maintenance*, ICSM '13, pages 170–179, Washington, DC, USA, 2013. IEEE Computer Society.

[90] L. Lai, V. Chandra, R. C. Aitken, and P. Gupta. Slackprobe: A flexible and efficient in situ timing slack monitoring methodology. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 33(8):1168–1179, Aug 2014.

[91] Seung-Hi Lee. Multirate digital control system design and its application to computer disk drives. *IEEE Transactions on Control Systems Technology*, 14(1):124–133, Jan 2006.

[92] L. Leem, H. Cho, J. Bau, Q. A. Jacobson, and S. Mitra. Ersa: Error resilient system architecture for probabilistic applications. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, pages 1560–1565, March 2010.

[93] Man-Lap Li, Pradeep Ramachandran, Swarup Kumar Sahoo, Sarita V. Adve, Vikram S. Adve, and Yuanyuan Zhou. Understanding the propagation of hard errors to software and implications for resilient system design. In *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XIII, pages 265–276, New York, NY, USA, 2008. ACM.

[94] Xin Li, Mian Dong, Zhan Ma, and Felix C.A. Fernandes. Greentube: Power optimization for mobile videostreaming via dynamic cache management. In *Proceedings of the 20th ACM International Conference on Multimedia*, MM '12, pages 279–288, New York, NY, USA, 2012. ACM.

[95] Xueliang Li, Guihai Yan, Yinhe Han, and Xiaowei Li. Smartcap: User experience-oriented power adaptation for smartphone's application processor. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*, pages 57–60, March 2013.

[96] Xueliang Li, Guihai Yan, Yinhe Han, and Xiaowei Li. Smartcap: Using machine learning for power adaptation of smartphone's application processor. *ACM Trans. Des. Autom. Electron. Syst.*, 20(1):8:1–8:16, November 2014.

[97] Wen-Yew Liang, Shih-Chang Chen, Yang-Lang Chang, and Jyh-Perng Fang. Memory-aware dynamic voltage and frequency prediction for portable devices. In *Embedded and Real-Time Computing Systems and Applications, 2008. RTCSA '08. 14th IEEE International Conference on*, pages 229–236, Aug 2008.

[98] Wen-Yew Liang and Po-Ting 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*, pages 1–6, Aug 2010.

[99] Geunsik Lim, Changwoo Min, Dong Hyun Kang, and Young Ik Eom. User-aware power management for mobile devices. In *Consumer Electronics (GCCE), 2013 IEEE 2nd Global Conference on*, pages 151–152, Oct 2013.

[100] G. Littlewort, M. S. Bartlett, I. Fasel, J. Susskind, and J. Movellan. Dynamics of facial expression extracted automatically from video. In *Computer Vision and Pattern Recognition Workshop, 2004. CVPRW '04. Conference on*, pages 80–80, June 2004.

[101] G. Littlewort, J. Whitehill, T. Wu, I. Fasel, M. Frank, J. Movellan, and M. Bartlett. The computer expression recognition toolbox (cert). In *Automatic Face Gesture Recognition and Workshops (FG 2011), 2011 IEEE International Conference on*, pages 298–305, March 2011.

[102] Yepang Liu, Chang Xu, S.C. Cheung, and Jian Lu. Greendroid: Automated diagnosis of energy inefficiency for smartphone applications. *Software Engineering, IEEE Transactions on*, 40(9):911–940, Sept 2014.

[103] Marcelo Martins and Rodrigo Fonseca. Application modes: A narrow interface for end-user power management in mobile devices. In *Proceedings of the 14th Workshop on Mobile Computing Systems and Applications*, HotMobile '13, pages 5:1–5:6, New York, NY, USA, 2013. ACM.

[104] Marcelo Martins and Rodrigo Fonseca. Application modes: A narrow interface for end-user power management in mobile devices. In *Proceedings of the 14th Workshop on Mobile Computing Systems and Applications*, HotMobile '13, pages 5:1–5:6, New York, NY, USA, 2013. ACM.

[105] P. Mercati, A. Bartolini, F. Paterna, T. S. Rosing, and L. Benini. A linux-governor based dynamic reliability manager for android mobile devices. In *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014*, pages 1–4, March 2014.

[106] P. Mercati, F. Paterna, A. Bartolini, L. Benini, and T. Rosing. Warm: Workload-aware reliability management in linux/android. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, PP(99):1–1, 2016.

[107] P. Mercati, F. Paterna, A. Bartolini, L. Benini, and T. S. Rosing. Dynamic variability management in mobile multicore processors under lifetime constraints. In *Computer Design (ICCD), 2014 32nd IEEE International Conference on*, pages 448–455, Oct 2014.

[108] P. Mercati, T. S. Rosing, V. Hanumaiah, J. Kulkarni, and S. Bloch. User-centric joint power and thermal management for smartphones. In *Mobile Computing, Applications and Services (MobiCASE), 2014 6th International Conference on*, pages 98–105, Nov 2014.

[109] Pietro Mercati, Andrea Bartolini, Francesco Paterna, Luca Benini, and Tajana Simunic Rosing. An on-line reliability emulation framework. In *Proceedings of the 2014 12th IEEE International Conference on Embedded and Ubiquitous Computing*, EUC '14, pages 334–339, Washington, DC, USA, 2014. IEEE Computer Society.

[110] Pietro Mercati, Andrea Bartolini, Francesco Paterna, Tajana Simunic Rosing, and Luca Benini. Workload and user experience-aware dynamic reliability management in multicore processors. In *Proceedings of the 50th Annual Design Automation Conference*, DAC '13, pages 2:1–2:6, New York, NY, USA, 2013. ACM.

[111] Pietro Mercati, Vinay Hanumaiah, Jitendra Kulkarni, Simon Bloch, and Tajana Rosing. Blast: Battery lifetime-constrained adaptation with selected target in mobile devices. *EAI Endorsed Transactions on Energy Web*, 15(5), 8 2015.

[112] Pietro Mercati, Francesco Paterna, Andrea Bartolini, Mohsen Imani, Luca Benini, and Tajana Šimunić Rosing. Vardroid: Online variability emulation in android/linux platforms. In *Proceedings of the 26th Edition on Great Lakes Symposium on VLSI*, GLSVLSI '16, pages 269–274, New York, NY, USA, 2016. ACM.

[113] M. Miranda, B. Dierickx, P. Zuber, P. Dobrovoln, F. Kutscherauer, P. Roussel, and P. Poliakov. Variability aware modeling of socs: From device variations to manufactured system yield. In *Quality of Electronic Design, 2009. ISQED 2009. Quality Electronic Design*, pages 547 –553, march 2009.

[114] D. Mirzoyan, B. Akesson, and K. Goossens. Process-variation aware mapping of real-time streaming applications to mpsocs for improved yield. In *Quality Electronic Design (ISQED), 2012 13th International Symposium on*, pages 41 –48, march 2012.

[115] Sparsh Mittal. A survey of techniques for improving energy efficiency in embedded computing systems. *CoRR*, abs/1401.0765, 2014.

[116] Sparsh Mittal. A survey of architectural techniques for managing process variation. *ACM Comput. Surv.*, 48(4):54:1–54:29, February 2016.

[117] T. Moreau, A. Sampson, and L. Ceze. Approximate computing: Making mobile systems more efficient. *IEEE Pervasive Computing*, 14(2):9–13, Apr 2015.

[118] N. Murray, Y. Qiao, B. Lee, G. M. Muntean, and A. K. Karunakar. Age and gender influence on perceived olfactory amp; visual media synchronization. In *2013 IEEE International Conference on Multimedia and Expo (ICME)*, pages 1–6, July 2013.

[119] T. S. Muthukaruppan and T. Mitra. Lifetime reliability aware architectural adaptation. In *2013 26th International Conference on VLSI Design and 2013 12th International Conference on Embedded Systems*, pages 227–232, Jan 2013.

[120] A. A. Nacci, F. Trovò, F. Maggi, M. Ferroni, A. Cazzola, D. Sciuto, and M. D. Santambrogio. Adaptive and flexible smartphone power modeling. *Mobile Networks and Applications*, 18(5):600–609, 2013.

[121] Nachi K. Nithi and Adriaan J. de Lind van Wijngaarden. Smart power management for mobile handsets. *Bell Lab. Tech. J.*, 15(4):149–168, March 2011.

[122] Nachi K. Nithi and Adriaan J. de Lind van Wijngaarden. Smart power management for mobile handsets. *Bell Lab. Tech. J.*, 15(4):149–168, March 2011.

[123] N. Oh, P. P. Shirvani, and E. J. McCluskey. Error detection by duplicated instructions in super-scalar processors. *IEEE Transactions on Reliability*, 51(1):63–75, Mar 2002.

[124] Venkatesh Pallipadi and Alexey Starikovskiy. The ondemand governor - past, present, and future. In *Proc. Linux Symposium*, volume 2, pages 223–238, 2006.

[125] Venkatesh Pallipadi and Alexey Starikovskiy. The ondemand governor: past, present and future. In *Proceedings of Linux Symposium, vol. 2, pp. 223-238*, 2006.

[126] Jurn-Gyu Park, Chen-Ying Hsieh, N. Dutt, and Sung-Soo Lim. Quality-aware mobile graphics workload characterization for energy-efficient dvfs design. In *Embedded Systems for Real-time Multimedia (ESTIMedia), 2014 IEEE 12th Symposium on*, pages 70–79, Oct 2014.

[127] F. Paterna, A. Acquaviva, and L. Benini. Aging-aware energy-efficient workload allocation for mobile multimedia platforms. *IEEE Transactions on Parallel and Distributed Systems*, 24(8):1489–1499, Aug 2013.

[128] F. Paterna, A. Acquaviva, A. Caprara, F. Papariello, G. Desoli, and L. Benini. Variability-aware task allocation for energy-efficient quality of service provisioning in embedded streaming multimedia applications. *IEEE Transactions on Computers*, 61(7):939 –953, july 2012.

[129] F. Paterna and T. Rosing. Modeling and mitigation of extra-soc thermal coupling effects and heat transfer variations in mobile devices. In *Computer-Aided Design (ICCAD), 2015 IEEE/ACM International Conference on*, pages 831–838, Nov 2015.

[130] F. Paterna, J. Zanotelli, and T. S. Rosing. Ambient variation-tolerant and inter components aware thermal management for mobile system on chips. In *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1–6, March 2014.

[131] A. Pathania, Qing Jiao, A. Prakash, and T. Mitra. Integrated cpu-gpu power management for 3d mobile games. In *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, June 2014.

[132] A. Pellegrini, R. Smolinski, L. Chen, X. Fu, S. K. S. Hari, J. Jiang, S. V. Adve, T. Austin, and V. Bertacco. Crashtest'ing swat: Accurate, gate-level evaluation of symptom-based resiliency solutions. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2012*, pages 1106–1109, March 2012.

[133] C. Perera, C. H. Liu, S. Jayawardena, and M. Chen. A survey on internet of things from industrial market perspective. *IEEE Access*, 2:1660–1679, 2014.

[134] Vinivius Petrucci, Orlando Loques, and Daniel Mosse. Lucky scheduling for energy-efficient heterogeneous multi-core systems. In *Presented as part of the 2012 Workshop on Power-Aware Computing and Systems*, Berkeley, CA, 2012. USENIX.

[135] Nishkam Ravi, James Scott, Lu Han, and Liviu Iftode. Context-aware battery management for mobile phones. In *Proceedings of the 2008 Sixth Annual IEEE International Conference on Pervasive Computing and Communications*, PERCOM '08, pages 224–233, Washington, DC, USA, 2008. IEEE Computer Society.

[136] George A. Reis, Jonathan Chang, and David I. August. Automatic instruction-level software-only recovery. *IEEE Micro*, 27(1):36–47, January 2007.

[137] A. Rimell and M. Hollier. The significance of cross-modal interaction in audio-visual quality perception. In *Multimedia Signal Processing, 1999 IEEE 3rd Workshop on*, pages 509–514, 1999.

[138] A. Rudi, A. Bartolini, A. Lodi, and L. Benini. Optimum: Thermal-aware task allocation for heterogeneous many-core devices. In *Proceedings of the International Conference on High Performance Computing Simulation*, HPCS '12, pages 82–87, 2014.

[139] S. Ryu and K. Sohn. No-reference quality assessment for stereoscopic images based on binocular quality perception. *IEEE Transactions on Circuits and Systems for Video Technology*, 24(4):591–602, April 2014.

[140] Adrian Sampson, Jacob Nelson, Karin Strauss, and Luis Ceze. Approximate storage in solid-state memories. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-46, pages 25–36, New York, NY, USA, 2013. ACM.

[141] S. R. Sarangi, B. Greskamp, R. Teodorescu, J. Nakano, A. Tiwari, and J. Torrellas. Varius: A model of process variation and resulting timing errors for microarchitects. *IEEE Transactions on Semiconductor Manufacturing*, 21(1):3–13, Feb 2008.

[142] Simon Scholler, Sebastian Bosse, Matthias Sebastian Treder, Benjamin Blankertz, Gabriel Curio, Klaus-Robert Muller, and Thomas Wiegand. Toward a direct measure of video quality perception using eeg. *Trans. Img. Proc.*, 21(5):2619–2629, May 2012.

[143] Krishna Sekar. Power and thermal challenges in mobile devices. In *Proceedings of the 19th Annual International Conference on Mobile Computing & Networking*, MobiCom '13, pages 363–368, New York, NY, USA, 2013. ACM.

[144] Clayton Shepard, Ahmad Rahmati, Chad Tossell, Lin Zhong, and Phillip Kortum. Livelab: Measuring wireless networks and smartphone users in the field. *SIGMETRICS Perform. Eval. Rev.*, 38(3):15–20, January 2011.

[145] Alex Shye, Benjamin Scholbrock, and Gokhan Memik. Into the wild: Studying real user activity patterns to guide power optimizations for mobile architectures. In *Proceedings of the 42Nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 42, pages 168–178, New York, NY, USA, 2009. ACM.

[146] Karan Sikka, Karmen Dykstra, Suchitra Sathyanarayana, Gwen Littlewort, and Marian Bartlett. Multiple kernel learning for emotion recognition in the wild. In *Proceedings of the 15th ACM on International Conference on Multimodal Interaction*, ICMI '13, pages 517–524, New York, NY, USA, 2013. ACM.

[147] P. Singh, E. Karl, D. Blaauw, and D. Sylvester. Compact degradation sensors for monitoring nbti and oxide degradation. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 20(9):1645 –1655, sept. 2012.

[148] P. Singh, E. Karl, D. Sylvester, and D. Blaauw. Dynamic nbti management using a 45 nm multi-degradation sensor. *Circuits and Systems I: Regular Papers, IEEE Transactions on*, 58(9):2026 –2037, sept. 2011.

[149] Gaurav Singla, Gurinderjit Kaur, Ali K. Unver, and Umit Y. Ogras. Predictive dynamic thermal and power management for heterogeneous mobile platforms. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, DATE '15, pages 960–965, San Jose, CA, USA, 2015. EDA Consortium.

[150] Kevin Skadron, Mircea R. Stan, Karthik Sankaranarayanan, Wei Huang, Sivakumar Velusamy, and David Tarjan. Temperature-aware microarchitecture: Modeling and implementation. *ACM Trans. Archit. Code Optim.*, 1(1):94–125, March 2004.

[151] Clinton W. Smullen, Vidyabhushan Mohan, Anurag Nigam, Sudhanva Gurumurthi, and Mircea R. Stan. Relaxing non-volatility for fast and energy-efficient stt-ram caches. In *Proceedings of the 2011 IEEE 17th International Symposium on High Performance Computer Architecture*, HPCA '11, pages 50–61, Washington, DC, USA, 2011. IEEE Computer Society.

[152] W. J. Song, S. Mukhopadhyay, and S. Yalamanchili. Managing performance-reliability tradeoffs in multicore processors. In *2015 IEEE International Reliability Physics Symposium*, pages 3C.1.1–3C.1.7, April 2015.

[153] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers. The case for lifetime reliability-aware microprocessors. In *Computer Architecture, 2004. Proceedings. 31st Annual International Symposium on*, pages 276–287, June 2004.

[154] J.H. Stathis. Physical and predictive models of ultra thin oxide reliability in cmos devices and circuits. In *Reliability Physics Symposium, 2001. Proceedings. 39th Annual. 2001 IEEE International*, pages 132 –149, 2001.

[155] M. Suk and B. Prabhakaran. Real-time mobile facial expression recognition system – a case study. In *2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 132–137, June 2014.

[156] Michael Taylor. A landscape of the new dark silicon design regime. *Micro, IEEE*, Sept-Oct. 2013.

[157] R. Teodorescu and J. Torrellas. Variation-aware application scheduling and power management for chip multiprocessors. In *Computer Architecture, 2008. ISCA '08. 35th International Symposium on*, pages 363 –374, june 2008.

[158] N. Vallina-Rodriguez and J. Crowcroft. Energy management techniques in modern mobile handsets. *IEEE Communications Surveys Tutorials*, 15(1):179–198, First 2013.

[159] C. H. (Kees) van Berkel. Multi-core for mobile phones. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '09, pages 1260–1265, 3001 Leuven, Belgium, Belgium, 2009. European Design and Automation Association.

[160] Lionel Vincent, Philippe Maurine, Suzanne Lesecq, and Edith Beigné. Embedding statistical tests for on-chip dynamic voltage and temperature monitoring. In *Proceedings of the 49th Annual Design Automation Conference*, DAC '12, pages 994–999, New York, NY, USA, 2012. ACM.

[161] John Von Neumann. Probabilistic logics and the synthesis of reliable organisms from unreliable components. *Automata studies*, 34:43–98, 1956.

[162] C. Wang, F. Yan, Y. Guo, and X. Chen. Power estimation for mobile applications with profile-driven battery traces. In *Low Power Electronics and Design (ISLPED), 2013 IEEE International Symposium on*, pages 120–125, Sept 2013.

[163] Shengquan Wang and Jian-Jia Chen. Thermal-aware lifetime reliability in multicore systems. In *Quality Electronic Design (ISQED), 2010 11th International Symposium on*, pages 399 –405, march 2010.

[164] X. Wang, L. Winemberg, D. Su, D. Tran, S. George, N. Ahmed, S. Palosh, A. Dobin, and M. Tehranipoor. Aging adaption in integrated circuits using a novel built-in sensor. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(1):109–121, Jan 2015.

[165] L. Wanner, S. Elmalaki, Liangzhen Lai, P. Gupta, and M. Srivastava. Varemu: An emulation testbed for variability-aware software. In *Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2013 International Conference on*, pages 1–10, Sept 2013.

[166] J. Wei, E. Juarez, M.J. Garrido, and F. Pescador. Maximizing the user experience with energy-based fair sharing in battery limited mobile systems. *Consumer Electronics, IEEE Transactions on*, 59(3):690–698, August 2013.

[167] E Wu, J Su, W Lai, E Nowak, J McKenna, A Vayshenker, and D Harmon. Interplay of voltage and temperature acceleration of oxide breakdown for ultra-thin gate oxides. *Solid-State Electronics*, 46(11):1787 – 1798, 2002.

[168] Y. Xiang, T. Chantem, R. P. Dick, X. S. Hu, and L. Shang. System-level reliability modeling for mpsocs. In *Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2010 IEEE/ACM/IFIP International Conference on*, pages 297–306, Oct 2010.

[169] Qing Xie, Mohammad Javad Dousti, and Massoud Pedram. Therminator: A thermal simulator for smartphones producing accurate chip and skin temperature

maps. In *Proceedings of the 2014 International Symposium on Low Power Electronics and Design*, ISLPED '14, pages 117–122, New York, NY, USA, 2014. ACM.

[170] Qing Xie, Jaemin Kim, Yanzhi Wang, Donghwa Shin, Naehyuck Chang, and Massoud Pedram. Dynamic thermal management in mobile devices considering the thermal coupling between battery and application processor. In *Proceedings of the International Conference on Computer-Aided Design*, ICCAD '13, pages 242–247, Piscataway, NJ, USA, 2013. IEEE Press.

[171] J. Xue and C. W. Chen. A study on perception of mobile video with surrounding contextual influences. In *Quality of Multimedia Experience (QoMEX), 2012 Fourth International Workshop on*, pages 248–253, July 2012.

[172] Chanmin Yoon, Dongwon Kim, Wonwoo Jung, Chulkoo Kang, and Hojung Cha. Appscope: Application energy metering framework for android smartphones using kernel activity monitoring. In *Proceedings of the 2012 USENIX Conference on Annual Technical Conference*, USENIX ATC'12, pages 36–36, Berkeley, CA, USA, 2012. USENIX Association.

[173] D. Zhu and H. Aydin. Reliability-aware energy management for periodic real-time tasks. *IEEE Transactions on Computers*, 58(10):1382–1397, Oct 2009.

[174] Dakai Zhu, R. Melhem, and D. Mosse. The effects of energy management on reliability in real-time embedded systems. In *Computer Aided Design, 2004. ICCAD-2004. IEEE/ACM International Conference on*, pages 35–40, Nov 2004.

[175] C Zhuo, E Karl, P Singh, D Blaauw, and D Sylvester. Sensor driven reliability and wearout management. *Design Test of Computers, IEEE*, PP(99):1, 2009.

[176] C. Zhuo, D. Sylvester, and D. Blaauw. Process variation and temperature-aware reliability management. In *2010 Design, Automation Test in Europe Conference Exhibition (DATE 2010)*, pages 580–585, March 2010.

[177] Cheng Zhuo, K. Chopra, D. Sylvester, and D. Blaauw. Process variation and temperature-aware full chip oxide breakdown reliability analysis. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 30(9):1321 –1334, sept. 2011.