UNIVERSITY OF SOUTHAMPTON

Tuning Dynamic Power Management for Mobile Devices

by

James R.B. Bantock

Thesis for the degree of Doctor of Philosophy

in the
Faculty of Engineering and Physical Sciences
School of Electronics and Computer Science

January 2021

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF ENGINEERING AND PHYSICAL SCIENCES
SCHOOL OF ELECTRONICS AND COMPUTER SCIENCE

Doctor of Philosophy

by James R.B. Bantock

Mobile devices have rapidly reached almost ubiquitous adoption amongst the global population. Smartphones have been the catalyst for introduction of high-performance System-on-Chips to mobile devices bringing with them the capability to execute ever more demanding applications but also widespread power management challenges. Traditionally, the foremost power management challenge was extension of battery lifetime. The emergence of sustained performance applications including mobile gaming, Virtual and Augmented Reality has presented a new challenge in constraining performance to within a sustainable thermal envelope. Cooling techniques, limited to passive technologies in mobile devices, have proved insufficient to maintain device skin temperatures below thresholds the human skin can tolerate. Dynamic Power Management policies have been developed to reduce mobile device power consumption to meet both energy and thermal constraints. This thesis proposes and then explores a new area of research in systematic tuning of Dynamic Power Management policies for mobile devices.

Static and dynamic configuration of Dynamic Power Management policy parameters are compared to quantify the potential energy and performance improvements. Experimental results from a modern mobile device across four applications suggest up to 10% reduction in dropped frames and a 25% reduction in CPU energy consumption.

Interactive performance degradation from Frequency Capping - the Dynamic Power Management lever used in device skin temperature throttling - was shown to induce up to a 43% increase in dropped frames. A new Dynamic Power Management lever - Task Utilisation Scaling - is proposed and validated to mitigate the performance degradation by reducing the number of dropped frames by up to 8.5% compared to Frequency Capping.

Evaluation of interactive performance metrics such as dropped frames was shown to be up to 750x slower than CPU energy consumption. An investigation into the root cause of this problem lead to a new metric for estimating dropped frames - PPPx - which can accelerate evaluation by up to 43.3x. PPPx was validated in a tuning experiment that lasted 8 days which would otherwise have required 8 months.

Contents

Abbreviations

API	Application Programming Interface
AR	Augmented Reality
CNN	Convolutional Neural Network
CPU	Central Processing Unit
DNN	Deep Neural Network
DSE	Design Space Exploration
DSP	Digital Signal Processor
DPM	Dynamic Power Management
DVFS	Dynamic Voltage and Frequency Scaling
EAS	Energy Aware Scheduling
FPS	Frames-Per-Second
GPS	Global Positioning System
GPU	Graphics Processing Unit
HCI	Human Computer Interaction
HCS	Hierarchical Control System
HD	High Definition
UHD	Ultra High Definition
IC	Integrated Circuit
IP	Intellectual Property
IPS	Instructions-Per-Second
IR	Infrared Radiation
LSTM	Long Short-Term Memory
LT	Load Tracking
MEMS	Micro-Electro-Mechanical Systems
MIMO	Multiple Inputs Multiple Outputs
ML	Machine Learning
NDK	Native Development Kit
NoC	Network-on-Chip
NPU	Neural Processing Unit
NVM	Non-Volatile Memory
OS	Operating System
PCB	Printed Circuit Board

PDF	Probability Density Function
PELT	Per-Entity Load Tracking
PPP	Peak Probability Percentile
QoS	Quality of Service
QoE	Quality of Experience
RL	Reinforcement Learning
RNN	Recurrent Neural Network
RTT	Round-Trip Time
SDK	Software Development Kit
SISO	Single Input Single Output
SoC	System-on-Chip
SRAM	Static Random-Access-Memory
SSV	Structured Singular Value
TLP	Thread-Level Parallelism
UI	User Interface
VR	Virtual Reality
WALT	Window-Assisted Load Tracking

Declaration of Authorship

Print name:

Title of thesis:

I declare that this thesis and the work presented in it is my own and has been generated by me as the result of my own original research.

I confirm that:

1. This work was done wholly or mainly while in candidature for a research degree at this University;
2. Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated;
3. Where I have consulted the published work of others, this is always clearly attributed;
4. Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work;
5. I have acknowledged all main sources of help;
6. Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;
7. Parts of this work have been published as: [1]

Signature: Date: / /

Acknowledgements

I would like to acknowledge the following people for their support and advice throughout my PhD:

My supervisors Geoff and Bashir.

My friends and colleagues at the University of Southampton, ARM and Samsung.

My family.

Chapter 1

Introduction

The optimisation of mobile device operation during the design process is influenced by a range of factors reflecting their diverse capability and use-cases. In this chapter, these factors and trends that affect them are introduced including mobile device market trends in Section 1.1 and wider trends in computer architecture in Section 1.2. Existential challenges in tuning Dynamic Power Management for mobile devices are presented in Section 1.3 and traced from these challenges, are the research aims and contributions achieved by this thesis in Sections 1.4 and 1.5 respectively.

1.1 Trends in the mobile device market

Mobile devices have become a ubiquitous part of human life, smartphone ownership now extends to over 88% of the population of the United Kingdom [2]. The classification of a mobile device has extended beyond just mobile phones to include smartphones, tablets, laptops, smart watches, Virtual (VR) and Augmented Reality (AR) headsets [2, 3]. This fragmentation is in part due to increasing device specialisation to support demanding applications under tight constraints. Mobile devices are distinct from traditional computing devices by their increased portability as a result of reduced mass and volume. This reduction in form-factor presents unique design challenges including reliance on battery-power and device thermal constraints [4, 5, 6, 7].

Growth in mobile device ownership in the United Kingdom from 2013 to 2019 is shown in Figure 1.1. Data for smartphones, tablets and other wearables in the United States reflects a similar slowdown in growth [3]. Smartphone replacement cycles are increasing and with the slowdown in ownership growth this has led to a fall in smartphone unit shipments [8, 9]. Although unit sales are falling, this has been offset by an increase in revenue per unit [2, 10].

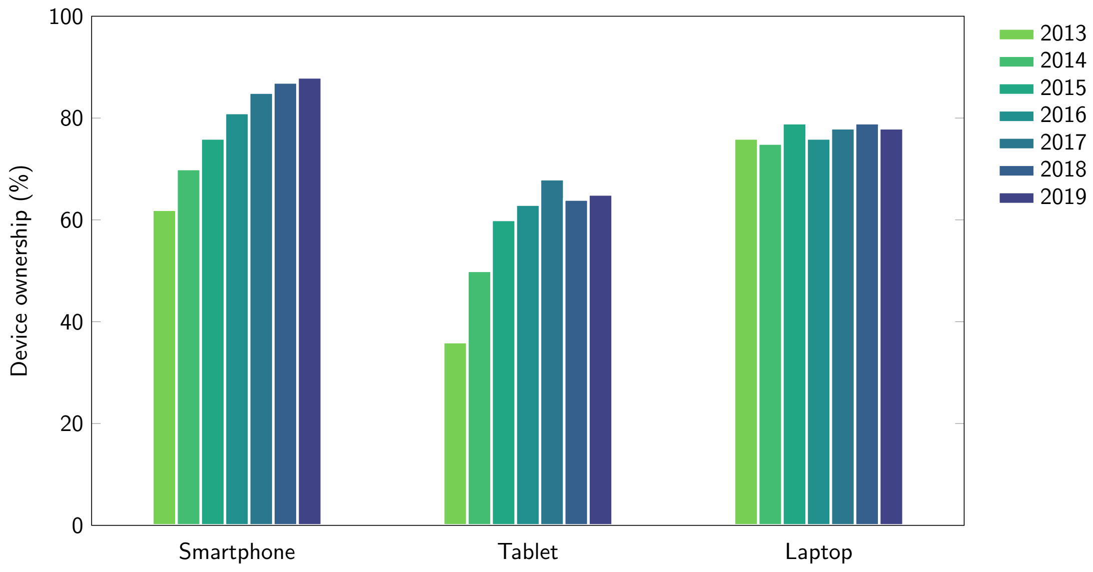


FIGURE 1.1: Mobile device ownership in the United Kingdom by year (2013-2019) - data from [2].

Global revenue estimated to be attributable to the mobile device value chain was over \$3tn in 2014 and represents far more than the estimated \$500bn revenue from mobile device retail [11]. In developing countries, mobile devices represent access to digital infrastructure that is forecast to drive \$3.9tn of purchases by 2022 [12]. The traditional business model of profit from device sales faces disruption as device manufacturers focus on monetising software and services [13, 14].

Competition between mobile device manufacturers has increased as the market has matured [8, 9]. Not all market dynamics can be explained by device performance and cost, eco-system factors such as the dominance of the mobile device operating systems Android and iOS influence sales [15, 16]. However, consumers are increasingly considering the capabilities of devices for differentiation including performance, battery-life and the camera [2, 3, 17].

Mobile device manufacturers not only compete against each other but must also drive sales from their existing customers to upgrade from their older devices [2]. Previously, visual differentiation between devices was sufficient for this purpose, in recent years a convergence in external design has reduced the effectiveness of this strategy [2]. Instead, manufacturers have focused on broadening the capability of mobile devices to encompass emerging applications that are too demanding for older devices.

Emerging mobile applications include mobile gaming, AR, VR and Machine Learning (ML) acceleration [2, 19, 20]. Mobile video gaming is the most established application with a forecast global consumer spend of \$90bn in 2019, more than all other gaming platforms combined [19]. The market opportunity for AR/VR has been forecast to be \$80bn in global revenue by 2025 with \$48bn driven by consumers and \$32bn enterprise

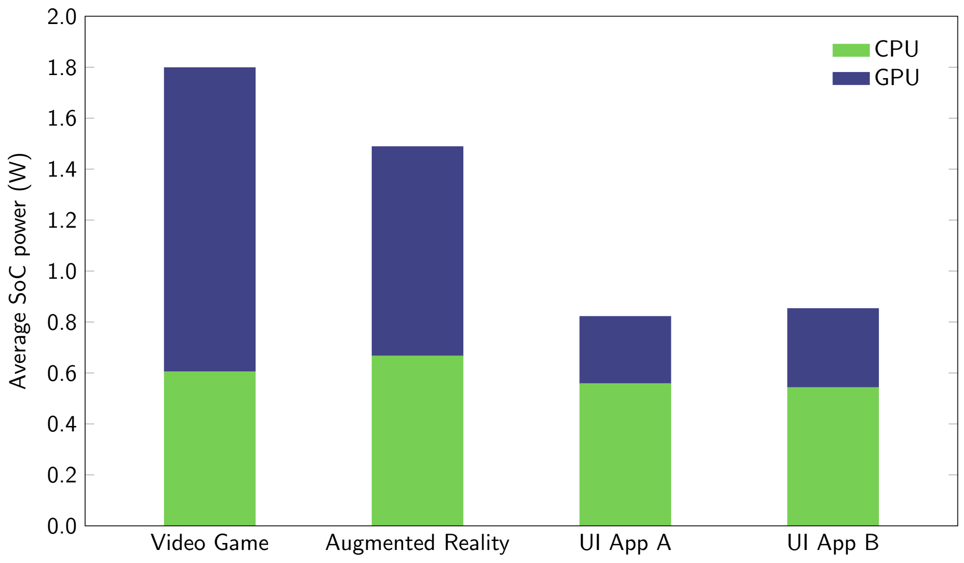


FIGURE 1.2: Average power consumed by CPU and GPU subsystems when executing smartphone applications - data from [18].

[21]. On-device ML capabilities are forecast to be present on 80% of smartphones by 2022, compared to 10% in 2017 [20].

Performance requirements from emerging mobile applications have been most demanding on the Graphics Processing Unit (GPU) of the System-on-Chip (SoC), shown in Figure 1.2. UI applications are mostly CPU dominant compared to emerging applications (mobile gaming, AR and VR) that are performance-limited by graphics processing power [18, 22, 23]. ML workloads are more suited to execution using dedicated acceleration hardware, however, as this technology is still in its infancy, existing hardware can be used [24, 25, 26, 27].

Modern mobile SoCs have been designed to target these performance requirements, improvements have been made to mobile GPUs in particular [28]. A portion of these improvements have increased the performance at the same power consumption, however, the overall power consumption of mobile SoCs has also increased [28]. This trend is highlighted in Figure 1.3, SoC power efficiency is improving but even the most efficient modern mobile System-on-Chips are consuming over 5W under demanding graphics workloads, outside the 3.5W sustainable thermal envelope of a smartphone [29, 30, 31].

Power consumption challenges have been addressed in computer architecture research previously, Dynamic Power Management (DPM) is the collective term for the use of an array of hardware and software levers available to scale power consumption at runtime [33]. Hardware DPM levers limit performance of the SoC using techniques such as Dynamic Voltage and Frequency Scaling (DVFS) and Central Processing Unit (CPU) idle states [33]. Software DPM levers include scheduling and mapping of software application tasks to available resources [34]. Modern mobile SoCs extensively make use of DPM

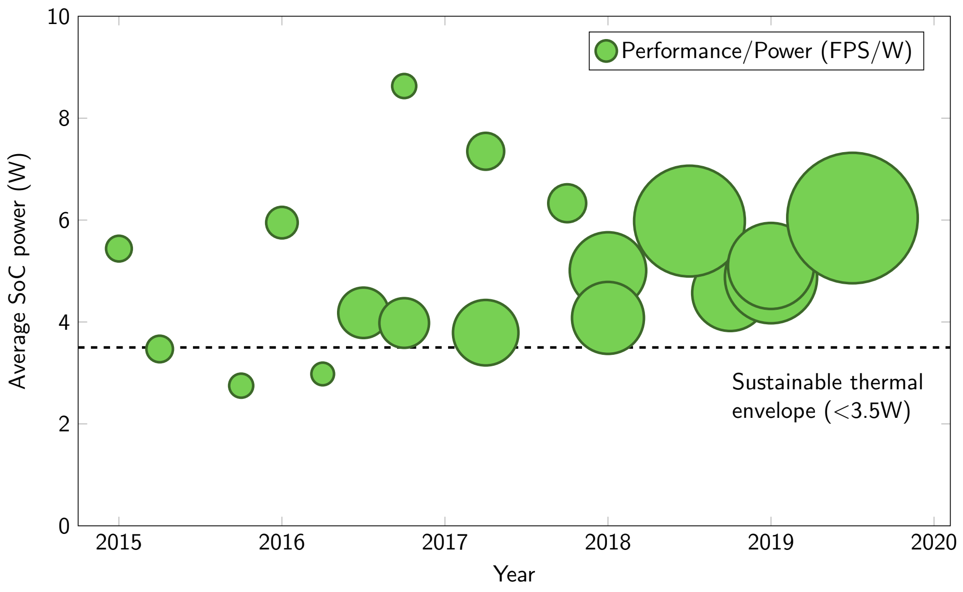


Figure 1.3: Mobile System-on-Chip average power consumption and performance per Watt during graphics benchmark - data from [28, 32].

techniques to maintain energy and thermal budgets [31], this leads to a regression in application and system performance [31, 35].

Operating Systems (OSs) typically have assumed responsibility for maintaining energy and thermal budgets in mobile devices [36]. Implementation of this functionality is not always aligned with the OS's traditional goals of throughput performance and fairness between application and users [37]. Control of DPM levers is typically implemented by the OS using individual policies [38, 39, 40], these policies may incorporate heuristics for user Quality of Experience in decision making. The separation between DPM lever policies has led to conflicts in between separate policies and the emergence of system-level techniques that aim to unify decision making [41].

Effective usage of DPM techniques is dependent on the workload being executed, this is not an issue for a single-workload system, however, modern mobile devices execute many diverse workloads [42, 43]. Variations in workload cannot be characterised by only observing the applications executing at that instant, environmental conditions such as ambient temperature are also a factor [44, 45]. DPM is tuned according to modelled or measured workloads during three processes. Design Space Exploration (DSE) provides the opportunity for changes to DPM levers directly but system performance and workloads are only models at this stage [46]. Static and dynamic configuration occurs with measured system performance data and includes the capability to tune DPM policy parameters such as thresholds and timers [47, 48], see Table 1.1. Design Space Exploration occurs during the System-on-Chip (SoC) design stage [49], static configuration as part of system integration and dynamic configuration during application execution [47, 48, 50]. Tuning using modelled performance and workload data can typically take

DPM policy tuning process	Design Space Exploration	Static Configuration	Dynamic Configuration
Design stage	SoC Design	Integration	Runtime
SoC performance	Model	Measured	Measured
System workload	Model	Model	Measured

Table 1.1: Dynamic Power Management (DPM) policy parameters may be tuned during three different processes: Design Space Exploration, static configuration and dynamic configuration.

place earlier in the design process than measured data but will be less representative of the final system.

Determination of an optimal configuration of DPM in a mobile device is not a feasible problem, there are factors such as software workload, environmental conditions and user input variation that are either unknown or can only be loosely modelled during the design process [42, 43, 44, 45, 46]. Traditionally, DPM has been reliant on relatively simple policies employing heuristics with static configuration such as the Linux power governors [38, 39, 40]. As hardware and software has moved on it has become evident that such solutions are no longer fit for purpose, a fact recognised by the Linux community through development of a new Energy Aware Scheduling (EAS) solution [51]. The use of a desktop and server kernel such as Linux to underpin the Android mobile Operating System has added design choice constraints compared to a clean slate mobile-first design. Future DPM policies will leverage ML techniques, research has thus far focused on Reinforcement Learning (RL) for DPM control [52] and Recurrent Neural Networks (RNNs) for workload prediction [53].

1.2 Trends in computer architecture

An increase in complexity of computer architecture presents the challenge of additional policy parameters that must be configured correctly for effective workload execution. These parameters are typically thresholds or timers that are statically or dynamically configured dependent on the architecture [48].

There are three major trends in computer architecture which affect system complexity, these are set to further increase the challenge of tuning DPM beyond current levels. The increase in computer architecture design metrics (described in Section 1.2.1), the increase in integrated chip design cost at leading nodes (Section 1.2.2) and the increasing

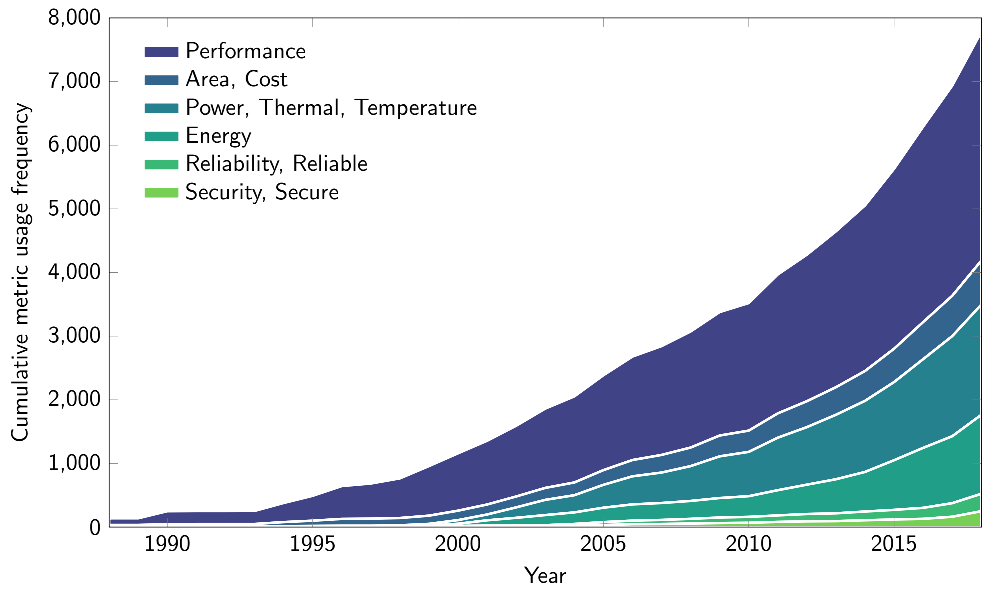


FIGURE 1.4: Usage of computer architecture design metrics in abstracts from leading academic conferences (HPCA, ISCA, MICRO) [61, 62, 63].

complexity of adaptive hardware and software architectures (Section 1.2.3). The combination of these trends in system complexity will result in an explosion of the state-space of parameters, including those relating to DPM policies, that must be tuned requiring new tuning techniques beyond manual and heuristic-based methods to be developed.

1.2.1 Design metrics

Computers have developed from calculation tools into general-purpose systems with a diverse range of applications. In correspondence with this, computer architecture research has developed to include a wider range of design metrics for optimisation. Initially, the predominant design metrics were performance and reducing silicon area to increase yield and lower costs [54]. As the consumer market shifted towards mobile devices, energy and thermal concerns were elevated to a higher level as the mobile form-factor presented these challenges [54, 55, 56]. Most recently, reliability and security have emerged as growing concerns as high-profile hardware vulnerabilities have raised questions about the use of computers for financial and confidential workloads [57, 58, 59, 60]. In Figure 1.4, the abstracts from leading computer architecture conferences have been datamined to highlight this trend.

An increase from two design metrics to six has placed a significant burden on the operating system to balance individual budgets in accordance with requirements. In addition, requirements such as reliability and security can be at an application level as well at the system level to add further complication [64]. It is typical that to improve in accordance with one goal, for it to negatively affect another. Examples of this include the well

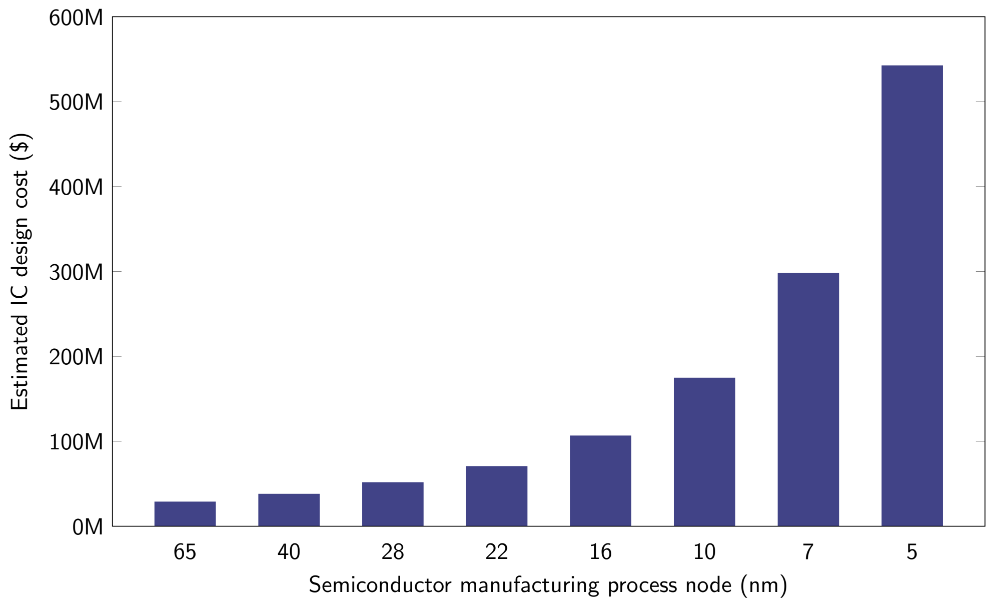


FIGURE 1.5: Estimated design costs for an Integrated Circuit (IC) as the semiconductor manufacturing process node shrinks - data from [73].

understood performance versus power trade-off but also less studied trade-offs including security versus area, energy and performance [65].

In the future, it is uncertain whether more design metrics may also be elevated to higher importance, these could include obfuscation to reduce Intellectual Property (IP) theft [66, 67] and environmental concerns such as product longevity or ability to be recycled [68, 69, 70, 71]. What is certain is that optimisation of computer architecture including the DPM subsystem must consider more design metrics than only performance and power which was its original remit.

1.2.2 Integrated Circuit design cost

At leading semiconductor nodes, the design cost for an Integrated Circuit (IC) is increasing [72, 73]. In Figure 1.5, estimates of the total design cost of an advanced IC at current and future nodes are presented. The IC design process is increasing in cost to a price point where it is prohibitive for many new designs to use the leading node. Looking ahead to 5nm and 3nm nodes, the design cost is forecast to exceed $1bn [73], this cost will become a challenge for even the largest semiconductor companies and will require innovative solutions.

One such solution to the increasing design cost that has been long proposed is to design ICs as a collection of smaller chiplets that may be packaged together [74, 75, 76]. Semiconductor manufacturers have commercialised 2.5D/3D integration processes for this purpose [77, 78, 79, 80].

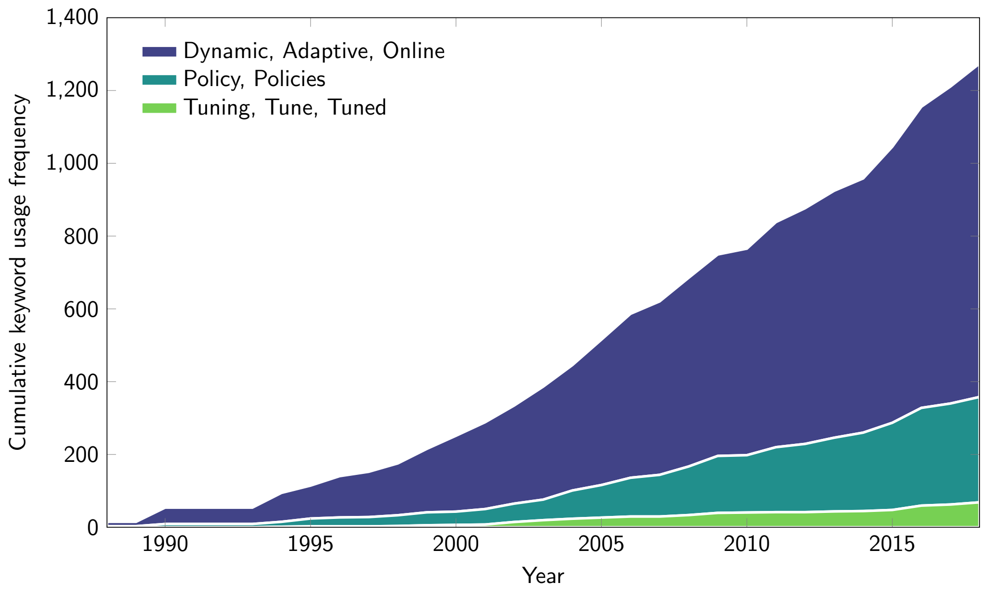


Figure 1.6: Usage of computer architecture terminology in abstracts from leading academic conferences (HPCA, ISCA, MICRO) [61, 62, 63].

SoC developers have begun to adopt 2.5D/3D integration of chiplets along with other advanced packaging techniques [81, 82, 83]. A major reason for this adoption has been the increased design cost such that it is not cost effective to design all parts of the chip at the leading node. Another is the increase in use of design modularity and repackaging chiplets into multiple different ICs [81, 84]. Design modularity and re-use has long been a key part of developing ICs, but the industry consolidation of the IP used is increasing in correlation with the increasing design costs.

Modular and reconfigurable systems by design have more parameters to allow effective reuse [56, 85, 86, 87], however, this also means that the total number of parameters in the system that must be tuned statically and dynamically will increase, further increasing the state-space for tuning DPM. As design costs continue to increase and companies are forced to license more external IP, this challenge will become more problematic.

1.2.3 Adaptive hardware and system software

Frequency of operation of high-end SoCs has plateaued due to physical constraints [88], in mobile devices the situation is similar but limited more by the sustainable thermal envelope [31]. However, reduction in transistor geometry has allowed more transistors on a single SoC at new design nodes [88]. Chip developers have made use of these extra transistors to develop more complex architectures to increase chip performance without changes in the frequency of operation [88]. As shown in Figure 1.6, researchers have placed more emphasis in their paper abstracts on creation of adaptive policies than techniques used to tune them. Modern architectural innovations range from advanced memory systems and branch predictors to Networks-on-Chip (NoCs). [54, 89], irrespective of their purpose, they implement policies in hardware and software that require tuning [90, 91].

Tuning of policies in hardware and software is dependent on both the rest of the system and the workloads being executed, it can rarely be configured once and reused for all systems and workloads. There is no evidence that computer architectures will become simpler in the future nor that frequency of operation of SoCs will increase beyond small increments, as such this trend is likely to continue. The number of policy parameters that must be tuned will therefore increase and improved tuning techniques must be developed accordingly.

1.3 Challenges in tuning Dynamic Power Management

The underlying trends described in Sections 1.2.1, 1.2.2 and 1.2.3 all suggest that tuning computer architecture will become increasingly complex in the future. The primary challenges faced by mobile devices relate to energy consumption and thermal management, both controlled by DPM. This implies that the already complex task of tuning DPM for mobile devices is likely to become both more important and increasingly challenging in the future and beyond the current limited techniques available.

Processes for tuning DPM may be implemented across the system layers of a mobile device from fundamental hardware design to pure software solutions as highlighted in Table 1.1. There are promising areas for investigation in each of DSE, static and dynamic configuration. In Section 1.3.1, the challenges raised by current commercial mobile devices are discussed, in particular, device thermal restrictions and DSE of new DPM systems with these challenges as a first-class target. In Section 1.3.2, the advances in configuration of system parameters in datacentre applications are highlighted which present an opportunity for static configuration of parameters in DPM policies.

1.3.1 Real devices and real workloads

Research into mobile devices must consider the challenges posed by the current and future generations of devices and applications in order to be relevant. Challenges that were highlighted when mobile devices first began to proliferate the market are not necessarily relevant in the current generation of devices. Applications which present the greatest challenge to modern mobile device DPM systems are of a sustained performance nature, such as AR, VR and mobile gaming [18, 22, 23, 92, 93]. These applications require performance that exceeds the sustainable thermal envelope of a smartphone which in turn leads to violations of the safe skin temperature threshold of 45°C [18, 31, 93, 94].

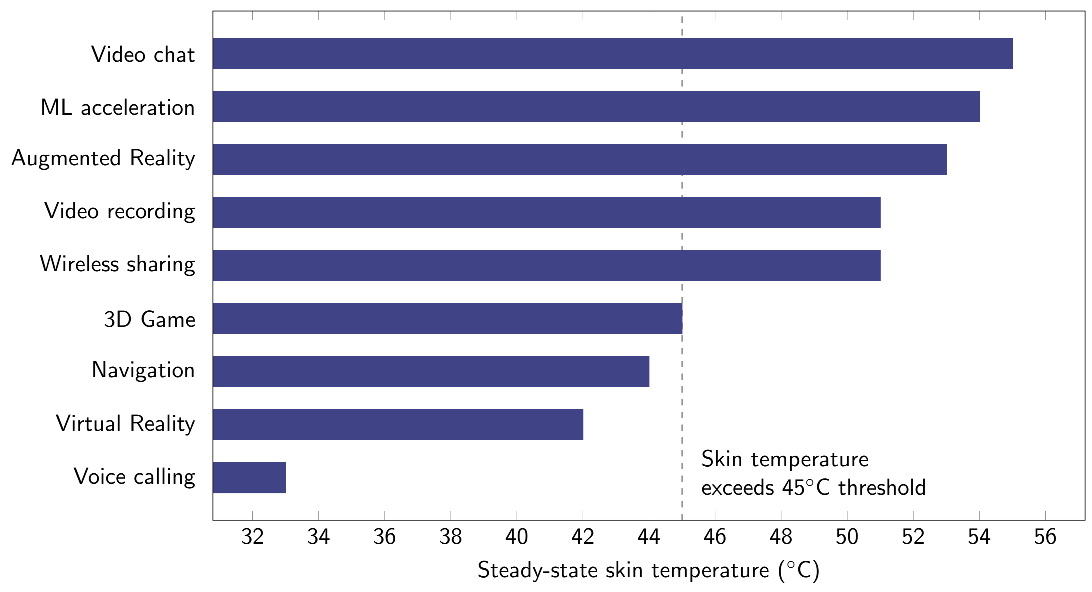


Figure 1.7: Steady-state device skin temperature during Android application execution on a smartphone - data from [93].

In Figure 1.7, results are presented from a study which measured the steady-state skin temperature when executing a variety of applications on a modern smartphone [93]. Thermal throttling was enabled, and conservative environmental conditions were chosen including physical suspension of the device. Despite thermal throttling being enabled, multiple applications were observed to exceed the safe skin temperature threshold of 45°C. The high average power consumed by modern mobile SoCs is an existential challenge for sustained usage [29, 30, 31].

Furthermore, unfortunately maintaining device skin temperature at or below 45°C cannot be considered a complete success. User satisfaction research has shown that user experience is degraded from 40°C. In Figure 1.8, results are presented from a survey of 20 users who played a 3D game on a smartphone for 20 minutes each [93]. The device skin temperature threshold was maintained at or below 45°C by the thermal throttling systems yet 75% of users reported they felt discomfort from the skin temperature. Reasons provided for discomfort include heat sensation, skin burn concerns, sweating and device becoming slippery as a result of sweat. Mobile device skin temperature violations by modern smartphones degrade user experience.

A common approach in DPM research is to make use of easily available hardware and software for experimentation. For example, development boards with external power sensors monitoring SoC power consumption reduce the barrier to entry for measurement [23, 95, 96, 97, 98]. Experimentation using this hardware doesn't necessarily represent the issues faced by real mobile devices. In addition, automation of mobile applications is a well-researched field of its own right [99] but is not necessarily trivial to implement for commercial devices and applications, instead it is easier to fall back on generic architecture benchmarks such as [100, 101]. These benchmarks do not represent real

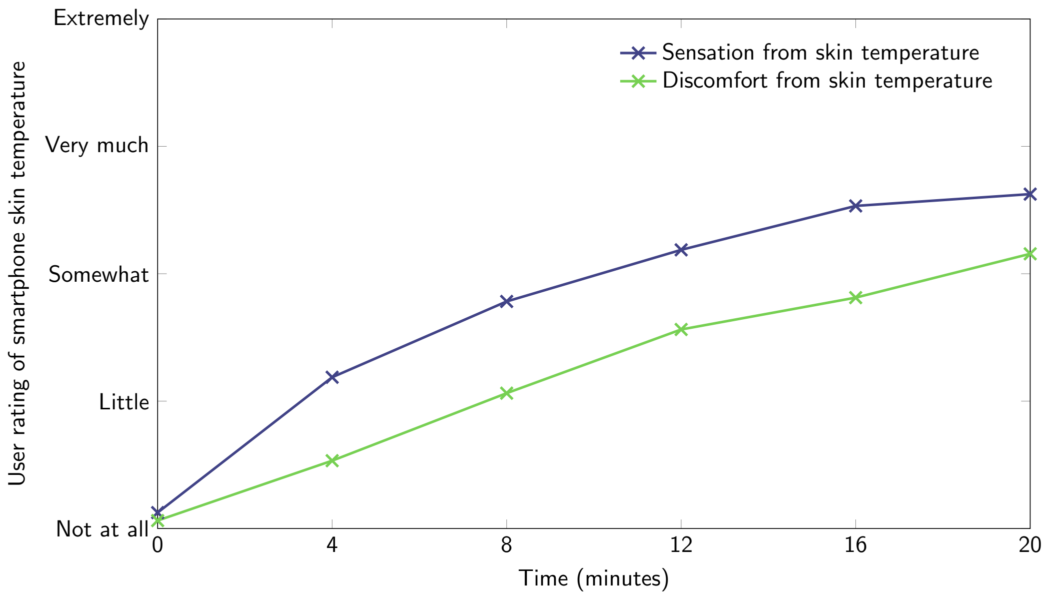


FIGURE 1.8: Average user ratings for heat sensation and discomfort while playing a 3D game on a smartphone - data from [93].

workloads for mobile devices and therefore the research conducted with them is likely to not be applicable mobile DPM [42, 102, 103].

1.3.2 Static configuration in datacentre applications

Datacentre applications are deployed and executed at immense scale by technology companies with capital expenditure exceeding \$50 billion per year by the largest five deployers [104]. Many of these applications are based on open-source software such as Apache Spark and Hadoop, designed to be sufficiently configurable to a wide range of use-cases. Inherently, these applications expose many parameters to the system administrator, this can potentially extend to over 100 parameters [105]. The focused execution of a small number of applications with an input workload that can be modelled has led to advances in configuration tuning for many datacentre applications. These can include webservers [105], databases [105, 106] and stream processing of data [107].

Previously, researchers have questioned from a Human Computer Interaction (HCI) perspective whether too many configuration parameters is a good thing [108], however, the approaches of partial configuration automation [109] or full configuration automation have received traction [105, 106] as automatic configuration tuning techniques have been demonstrated. In Figure 1.9, throughput improvement gained from parameter configuration is shown compared to the default parameter configuration for five datacentre applications [105]. The improvements of up to 330% are somewhat misleading as they imply the default configuration is particularly ineffective. However, in a case study, one of the application configurations was shown to exhibit 4% higher throughput than an expert human configuration. The same study showed that setting parameters outside

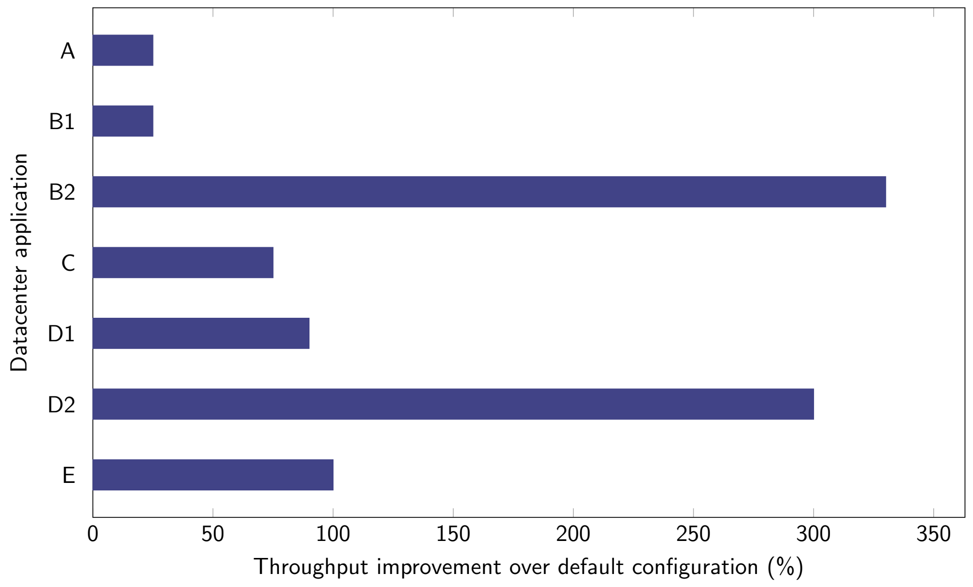


FIGURE 1.9: Throughput improvement after parameter configuration for datacentre applications compared to the default configuration - data from [105].

of the recommended value could be effective in increasing performance reflecting that a data-driven approach can exceed the results possible with human intuition [105].

Mobile DPM policies expose parameters such as threshold and timers to the system administrator [38, 39, 40, 48, 51]. Although not in a mobile use-case, some of these parameters have been tuned through static configuration demonstrating improvements in power efficiency [47]. The search techniques highlighted in the datacentre configuration tuning research are applicable to reducing the state-space of parameters in the mobile space. For example, grid search is highlighted as too sample inefficient and a recursive bounded random search is used instead [105].

There open research questions about the scale of benchmarking required to tune mobile workloads as there is not the single application and use-case model that reduces the scope of datacentre application configuration. The largest publicly announced mobile device lab is run by a mobile application developer [110, 111]. This device lab has reported issues with repeatability of benchmarks [112], citing implementation of deterministic DPM frequency selection which would not be an option when evaluating the DPM subsystem performance itself. Furthermore, mobile applications are interactive and subject to external resources such as network performance and user input [113].

1.4 Research aims

This thesis aims to advance techniques for tuning Dynamic Power Management for mobile devices in the face of increasing computer architecture complexity and policy parameter spaces. The research aims can be summarised as follows:

1. Quantify the potential performance and power improvements achievable through systematic tuning of mobile DPM policies.
2. Investigate the performance effects of the emergent mobile device skin temperature throttling algorithms and evaluate potential mitigations for this challenge.
3. Leverage research in datacentre application configuration tuning to improve static configuration of DPM power management policy parameters.

1.5 Research contributions

The contributions of this work towards addressing the aforementioned research aims are as follows:

1. A mobile DPM policy was tuned for web browsing applications under varying network conditions and user inputs demonstrating up to 13% CPU energy savings or QoE improvement of 27% addressing Research aim 1. A paper summarising contribution was presented at ISLPED 2017 [1].
2. A new DPM lever, Task Utilisation Scaling, was proposed to replace Frequency Capping in mobile device skin temperature throttling algorithms resulting in a decrease of dropped frames by up to 43% addressing Research aim 2. A paper reflecting this contribution was published in IEEE Embedded System Letters [114].
3. A new metric was proposed, PPPx, to accelerate evaluation of dropped frames in mobile applications by up to 43x. This metric was used with datacentre configuration tuning techniques to tune DPM policy parameters across four mobile applications demonstrating a reduction in dropped frames of up to 10% and a CPU energy reduction of up to 25% addressing Research aims 1 and 3.

1.6 Thesis structure

The remainder of this thesis is structured as follows:

In Chapter 2, a review of the relevant literature is presented including: the classification of mobile device hardware, software applications and challenges (Section 2.1), Dynamic Power Management levers and control policies (Section 2.2) and performance evaluation of applications, power consumption and tuning (Section 2.3).

In Chapter 3, empirical data is collected from experiments on a mobile System-on-Chip to support the case of tuning mobile Dynamic Power Management policies. This includes a comparison between static and dynamic configuration under user input and resource access-time variability (Section 3.1) and an architecture to dynamically configure policy parameters for unseen dynamic workloads (Section 3.2).

In Chapter 4, the use of Frequency Capping as a lever for mobile device skin temperature throttling is demonstrated along with the side-effects on application interactivity and user Quality of Experience (Section 4.1.2). A new lever is proposed, Task Utilisation Scaling, to mitigate these side-effects and its efficacy evaluated on both User Interface and sustained performance applications (Section 4.2.1).

In Chapter 5, the problem of multimodal distributions when measuring frame rate janks is highlighted along with effect it has on performance evaluation. A new metric is proposed, PPPx, to accelerate the evaluation of frame rate janks (Section 5.1). PPPx is validated through tuning of three Dynamic Power Management policies and their parameters on a mobile device 5.2.

In Chapter 6, conclusions are drawn (Section 6.1) and opportunities for future research are presented (Section 6.2).

Chapter 2

Background: Dynamic Power Management for Mobile Devices

In Chapter 1, emergent trends affecting the mobile device market and computer architecture as a whole were presented unveiling the growing challenge of tuning mobile devices. Through a review of background literature, this chapter sheds further light on the design constraints at the root of this challenge and the current techniques available to meet them. An overview of the classification of mobile devices is laid out in Section 2.1 including both hardware classification (Section 2.1.1) and classification of current and future software applications (Section 2.1.2). Following this, is an in-depth discussion of the energy and thermal challenges posed by the mobile device form-factor in Section 2.1.3. Dynamic Power Management (DPM), addressed in Section 2.2, describes a collection of levers (Section 2.2.1) and control policies (Section 2.2.2) that are effective in reducing power consumption to mitigate these energy and thermal challenges. The final piece of the puzzle is the capability to evaluate the performance of DPM against its goals which is presented in Section 2.3. Included in this evaluation is application performance (Section 2.3.1) and power (Section 2.3.2) measurements, either directly or indirectly through use of a model. The DPM evaluation results may be used to improve the DPM control policies (Section 2.3.3) through tuning parameters which acts to close the loop on the DPM design process.

2.1 Mobile devices

Market trends presented in Chapter 1 highlight a burgeoning ecosystem that has emerged around mobile devices. The limits of this ecosystem are defined in terms of both hardware (Section 2.1.1) and software specifications (Section 2.1.2), the combination of which pose significant challenges to device manufacturers from an energy and thermal perspective (Section 2.1.3).

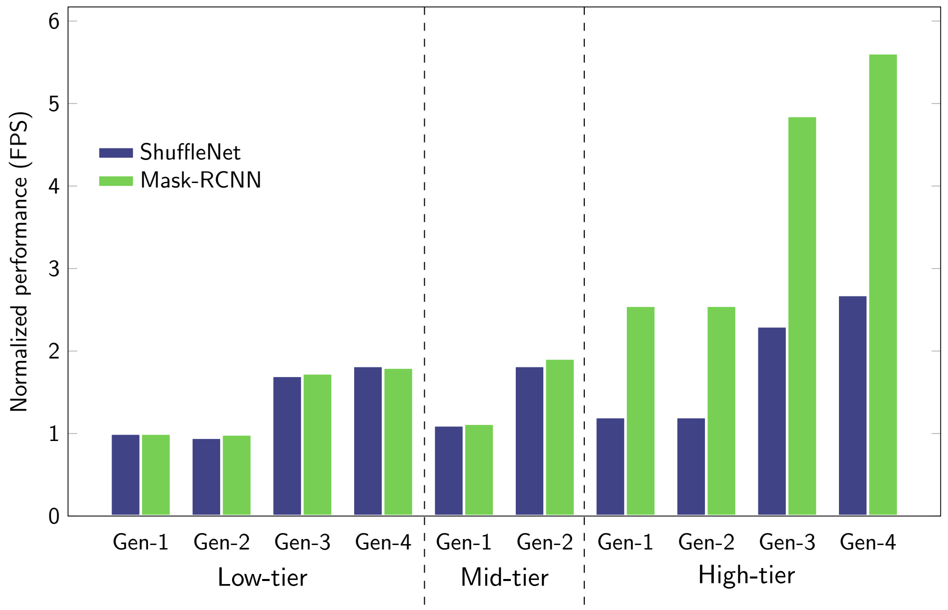


Figure 2.1: Segmentation in performance between tiers and generation of mobile device for DNN inference, specifically ShuffleNet and Mask-RCNN - data from [24]. The speed-up is not uniform across all applications particularly for emerging use-cases such as ML acceleration.

2.1.1 Hardware classification

Portability, inherent to mobile devices, defines much of the hardware requirements imposed on mobile device design [4], Broadly, mobile devices may be segmented by how they are intended to be carried or worn. For instance, the size and weight permitted by the laptop form-factor [115] allows integration of a greater hardware specification than that of a smartwatch [116]. This heterogeneity in hardware specification is visible in benchmarks executed across device classes and defines the possible software applications that may be used [117, 118]. Some of the more common mobile device form-factors are detailed in this section along with their particular requirements.

It would be disingenuous to review the state-of-the-art in mobile devices without starting with the smartphone, in Chapter 1 it was revealed that ownership has now extended to over 87% of the UK population. It should not be a surprise given the size of the market that segmentation has emerged within its bounds. In Figure 2.1, two benchmarks from Machine Learning (ML) are presented from a range of smartphones. When new generations of smartphone are compared to older generations, the benchmark performance is higher, but this is not necessarily uniform across tiers of the market [24]. In particular, high-tier smartphones are increasing in computational capability. Previously, much of the power budget of a mobile phone was allocated to the display and data connection [119, 120, 121], in today's flagship smartphones this is trending towards larger allocations for the CPU and GPU [18, 22, 23, 31]. Power consumption data for accelerators such as image and video processing can be up to 28% of SoC power usage for specialised tasks such as video recording [102]. Closely related to the smartphone is the tablet form-factor, which aside from a larger display and consequently battery often uses the same System-on-Chip (SoC) or a variant of the same System-on-Chip with additional graphics processing power[122].

Seated at the base of the pack in performance are low-power wearables, the smartwatch being one of most widespread [116]. With approximately 70% of their energy consumed in dozing or sleeping states, smartwatches are designed for lighter usage than other mobile devices [116]. The power budget allocation is not dissimilar to in smartphones, however, there are additional sensing functions for health and fitness applications that can dominate this allocation when enabled [123]. Due to their relative infancy in adoption it is not yet known if smartwatches and other wearables will mirror the ubiquitous usage of smartphones or if they will remain an outlet for push notifications from other mobile devices [116].

At the other end of the performance scale are Augmented Reality (AR) and Virtual Reality (VR) wearables. These wearables were initially scoped as standalone glasses or headsets, but the computational demands have so far proved too high for a feasible implementation as untethered devices [18, 22]. This has demonstrated that the software application requirements cannot be passed over during the hardware design process.

2.1.2 Software applications: current and future

Given the vast array of emergent software applications available to execute on mobile devices such as smartphones [16], it is forgivable to gloss over that the current generation of applications are still widely used and their performance important [102]. Studies have shown that the provisioning of powerful multi-core CPUs in smartphones doesn't correspond favourably with the largely single-threaded workloads that they execute [31, 43, 124].

Key functions that have been implemented by smartphones since their inception such as video decoding have been isolated into application-specific Intellectual Property (IP) cores on the SoC [102, 125]. These IP cores can be optimized for individual use-cases and often represent a large power and execution time improvement over CPU execution. However, the specialisation comes at a cost, if applications map poorly to these IP accelerators then these improvements are not always realisable [125].

Characterisation of current mainstream smartphone applications pinpoints power consumption allocated to three main subsystems, the CPU, memory and interconnect, and IP cores [102, 125] as showcased in Figure 2.2. When a web-browser is scrolled, much of the required computation is carried out by the CPU as there is no specialised IP core for this purpose, although one such core has been proposed [126]. However, during 4K video recording the Image Signal Processor (ISP) is exploited represented by the large

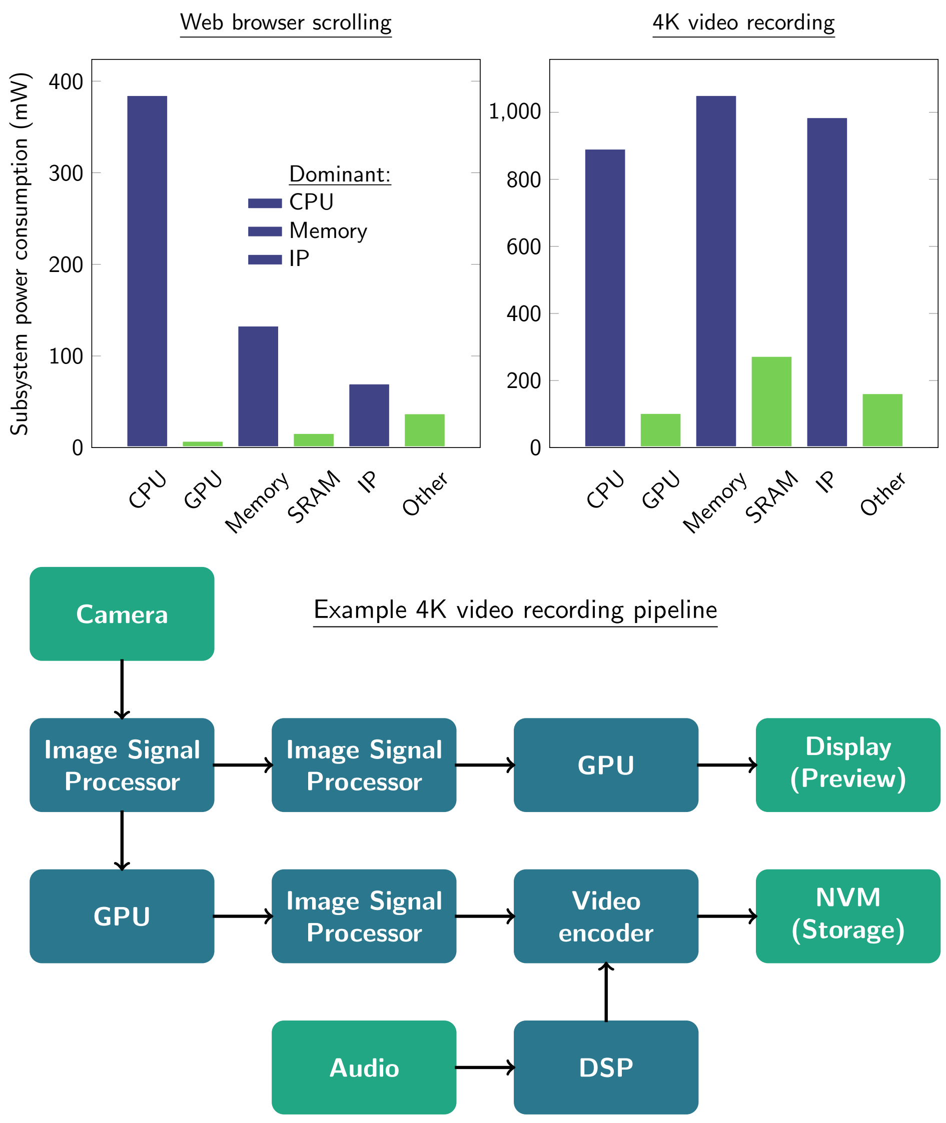
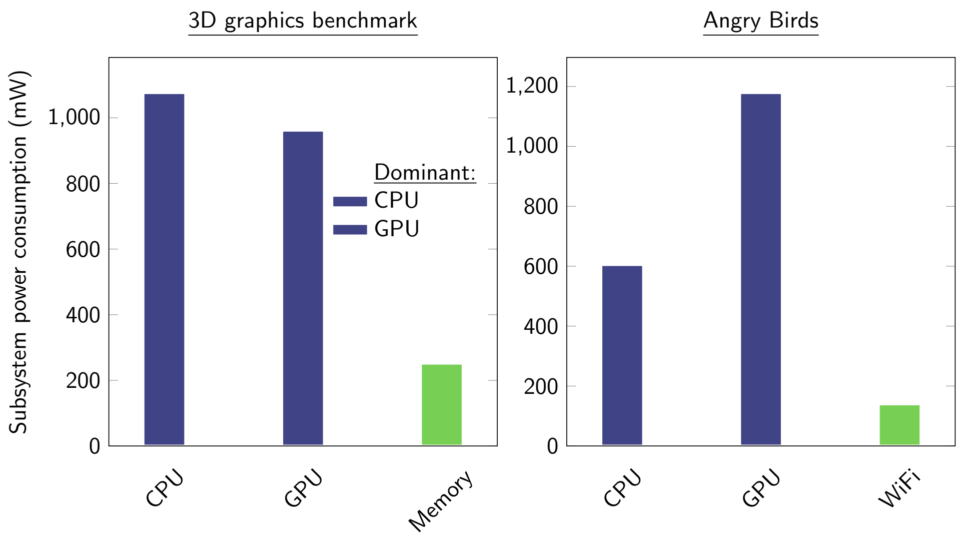


FIGURE 2.2: Power consumption broken down by subsystem for mainstream mobile device applications - data from [102].

IP core power consumption of up to 28%. It is important to note how minimal the GPU usage is for these applications when compared to emerging and future applications, as specialised accelerators are developed for emerging applications such as ML this may reduce the differential in power consumption breakdown.

Additionally, in Figure 2.2, a representative pipeline is laid out for a 4K video recording application. CPU, memory and IP subsystems are dominant as applications are executed by a complex chain of IP modules orchestrated by the CPU as shown in the example 4K video recording pipeline. There are two inputs, camera and audio, which are processed through a chain of IP cores to two outputs. These outputs are a preview version of the video on the smartphone display and the complete encoded video stored in Non-Volatile Memory (NVM). This (simplified for the purposes of illustration) complex chain



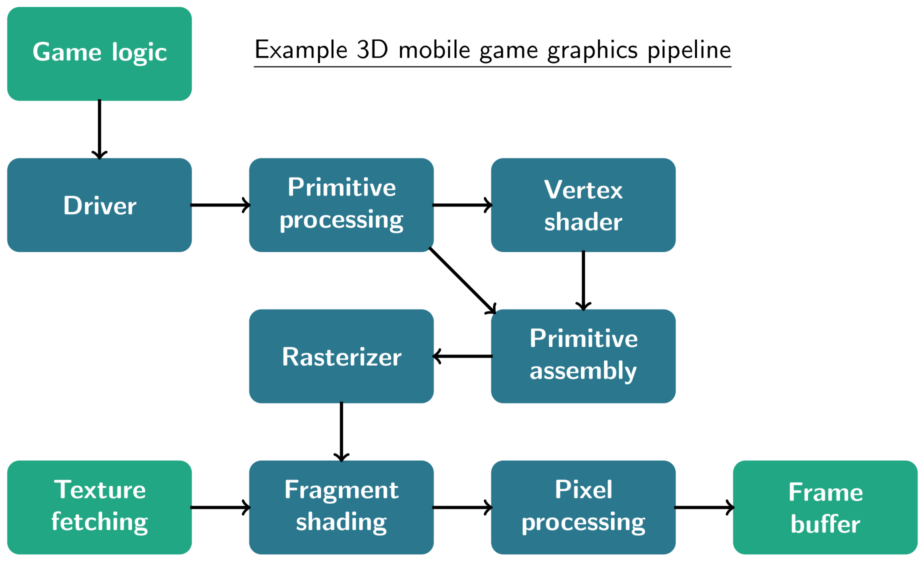


FIGURE 2.3: Power consumption broken down by subsystem for 2D and 3D mobile games - data from [18, 96].

of IP cores gives an indication of the orchestration required by the CPU and memory subsystems that consume similar power allocations to the IP cores themselves.

As highlighted in Chapter 1, there has been an emergence of applications with higher computational demands, the first of which are from mobile gaming. Mobile games, both 2D and 3D, involve frame-based computation with a target frame rate of up to 60 Frames-Per-Second (FPS) [23, 102]. Rendering of each frame individually computed using a combination of the CPU and GPU, either subsystem may throttle the game below 60 FPS if the performance or power required is beyond constraints. Figure 2.3 shows subsystem power consumption breakdowns for a 2D game and 3D graphics benchmark representative of a 3D game. In both cases the dominance of CPU and GPU power allocations is clear, as is the GPU power consumption of over 10x the mainstream applications in Figure 2.2.

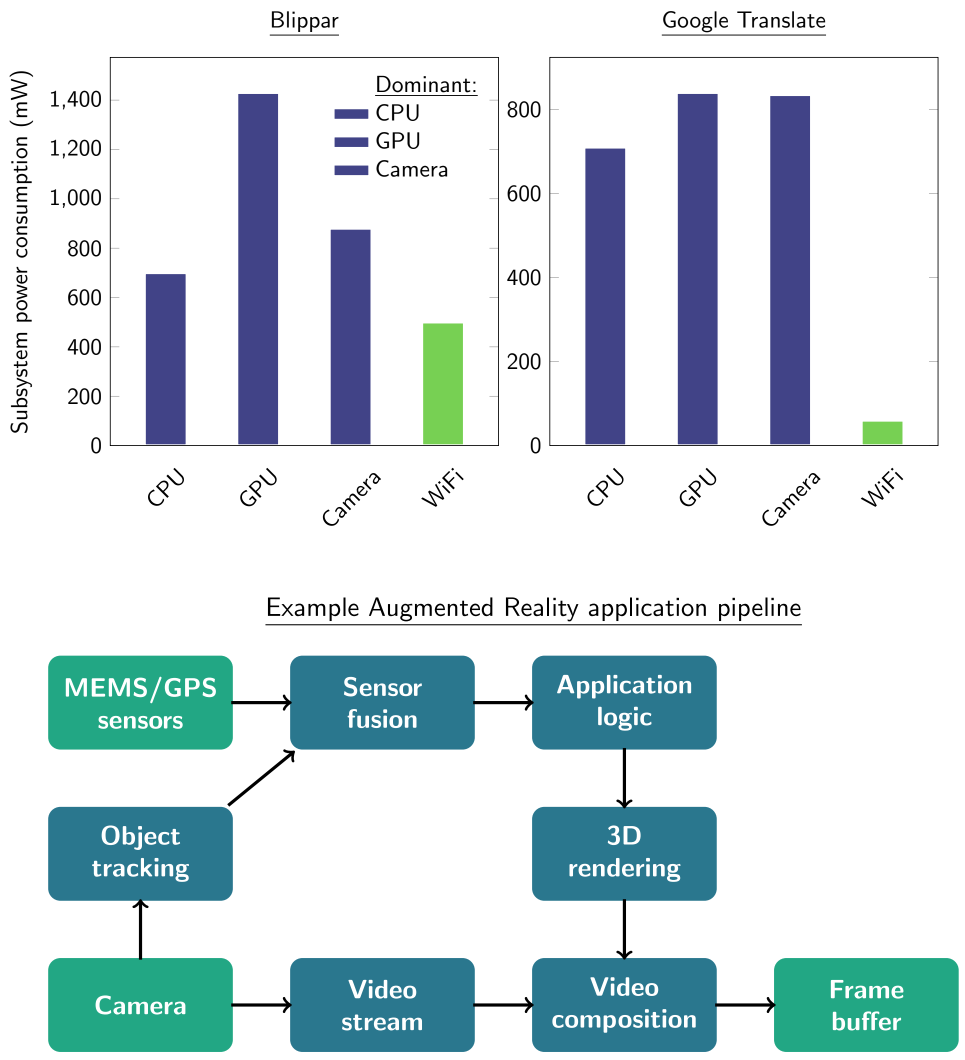


Figure 2.4: Power consumption broken down by subsystem for mobile Augmented Reality applications and example AR pipeline - data from [18].

An example mobile graphics pipeline for a 3D game is also shown in Figure 2.3 exhibiting minimal differences to a graphics pipeline for a desktop computer game. CPU and GPU subsystems are dominant when demanding workloads are processed with reduced usage of application-specific IP modules. The example graphics pipeline is executed on the GPU after the game logic and memory orchestration to fetch and load required textures is computed [127].

Augmented Reality (AR) applications are in their infancy with regards to mainstream adoption and this is reflected in the limited characterisation data available [18, 24]. The tasks that must be run to execute an AR application are more diverse than in other applications and this is reflected in a higher Thread-Level Parallelism (TLP) metric [18]. Figure 2.4 provides a high-level overview of the tasks involved in the pipeline. These include fusion sensor data from objects tracked by the camera with location data from Micro-Electro-Mechanical Sensors (MEMS) and the Global Positioning System (GPS) chipset, streaming video to the display and augmenting this video with 3D-rendered

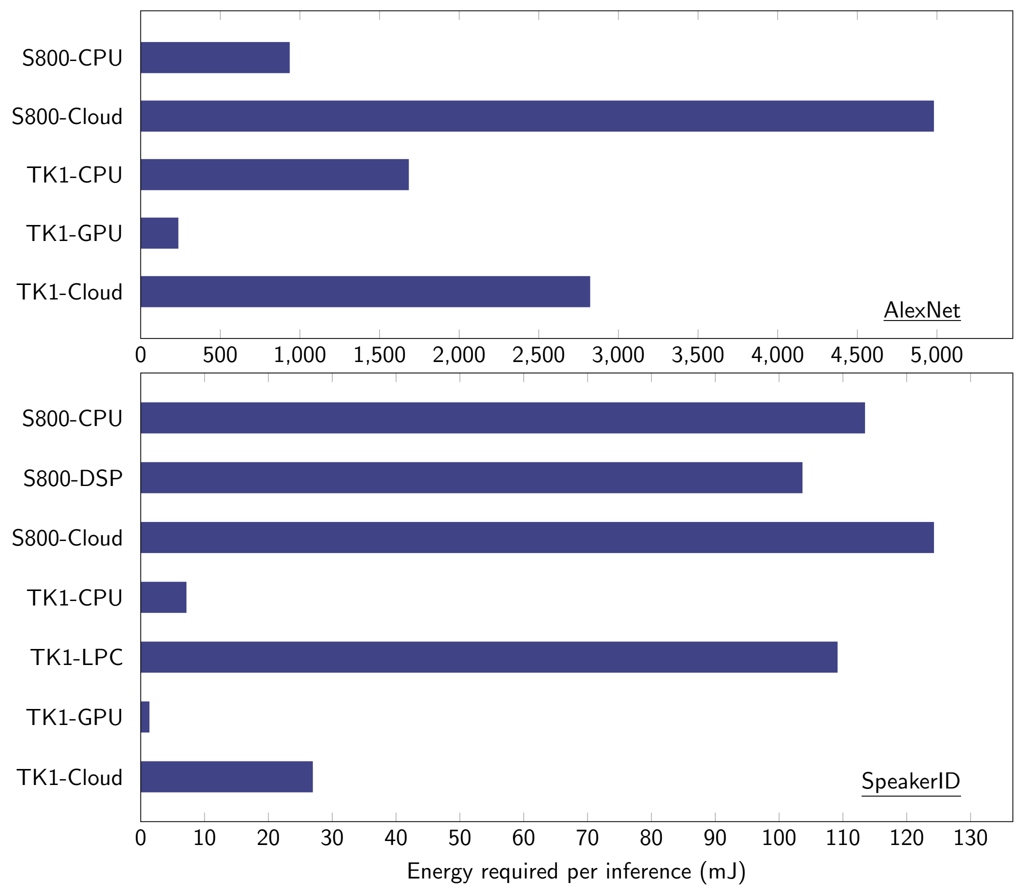


Figure 2.5: Energy required to execute Machine Learning models (AlexNet and SpeakerID) on-device and offloaded to the cloud - data from [128].

images from the application. Unsurprisingly, this execution increases power consumption across the SoC with particular dominance from the CPU, GPU and camera subsystems. One of the applications in Figure 2.4, Blippar, also conducts real-time web-search over WiFi during execution to exacerbate the power challenge.

Finally, the list of emerging applications would not be complete without Machine Learning (ML) acceleration. The rapid adoption estimates of 80% of smartphones by 2022 discussed in Chapter 1 may yet prove unfounded but it is already integrated in some popular applications [24, 115]. Whilst it is predicted that ML acceleration will benefit from specialist IP cores or Neural Processing Units (NPUs) in the future, for now a range of existing subsystems are used [24, 25, 26, 27].

An example of the heterogeneity possible when accelerating ML inference is visible in Figure 2.5; a comparison is made for inference of two ML models across the candidate accelerators of two SoCs. Accelerators are compared across the Snapdragon 800 and Tegra K1 SoCs. When compared with offloading to the cloud, most of the on-device accelerators compare favourably in energy consumption, although it must be noted there

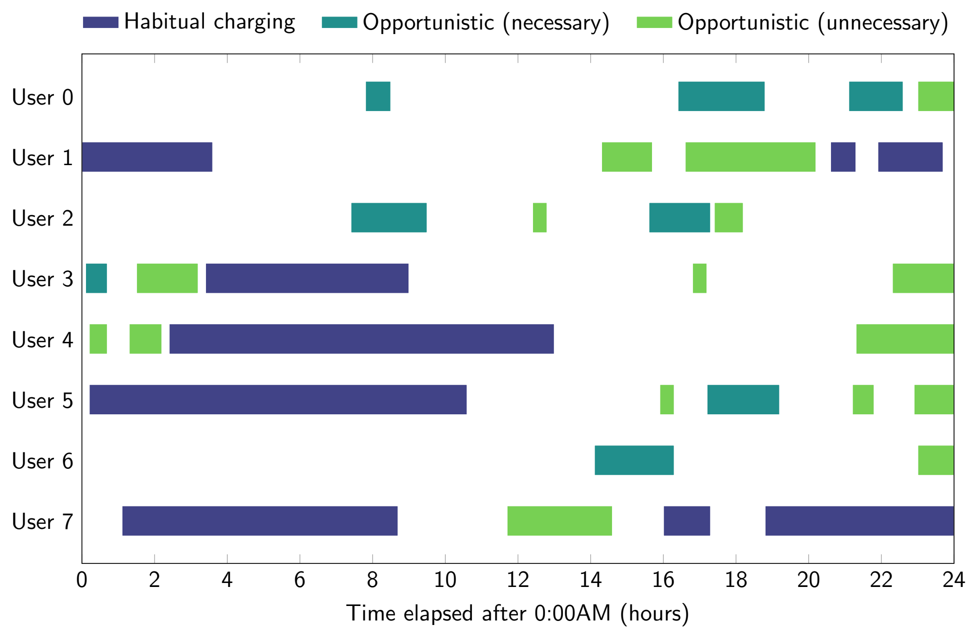


FIGURE 2.6: Mobile phone charging cycle profiles recorded from eight users during a mobile device usage study - data from [121].

are other trade-offs in latency and throughput. Out of the available on-device accelerators the GPU is most energy-efficient and is most commonly used in current devices [24, 27].

2.1.3 Device form factor challenges

As a precursor to analysis of the challenges imposed by the mobile device form-factor, it is important to reiterate the diversity in usage of smartphones and other devices amongst their users [119, 121, 129, 130, 131, 132]. Given the widespread adoption of smartphones in a large population of varying usage patterns, there will undoubtedly be variation in the challenges experienced in mobile device lifetimes. Irrespective of this variation, there are two major power management challenges, battery life and skin temperature throttling that will be discussed in this section.

Battery life preservation has always been at the forefront of mobile power management [4], if battery charge falls below the minimum level allowed for the battery technology then the mobile device will shut off and be unusable. Whilst battery capacity technologies have developed over time, they fall short of keeping pace with the increase in computational power of modern devices [31, 102].

There is an assumed requirement that mobile device battery life will extend for at least a day of use, however, in reality this requirement is less clear-cut. In Figure 2.6, the charge cycles of eight users from a mobile device usage study are shown. The charge cycles are annotated as one of three categories, habitual charging, necessary and unnecessary

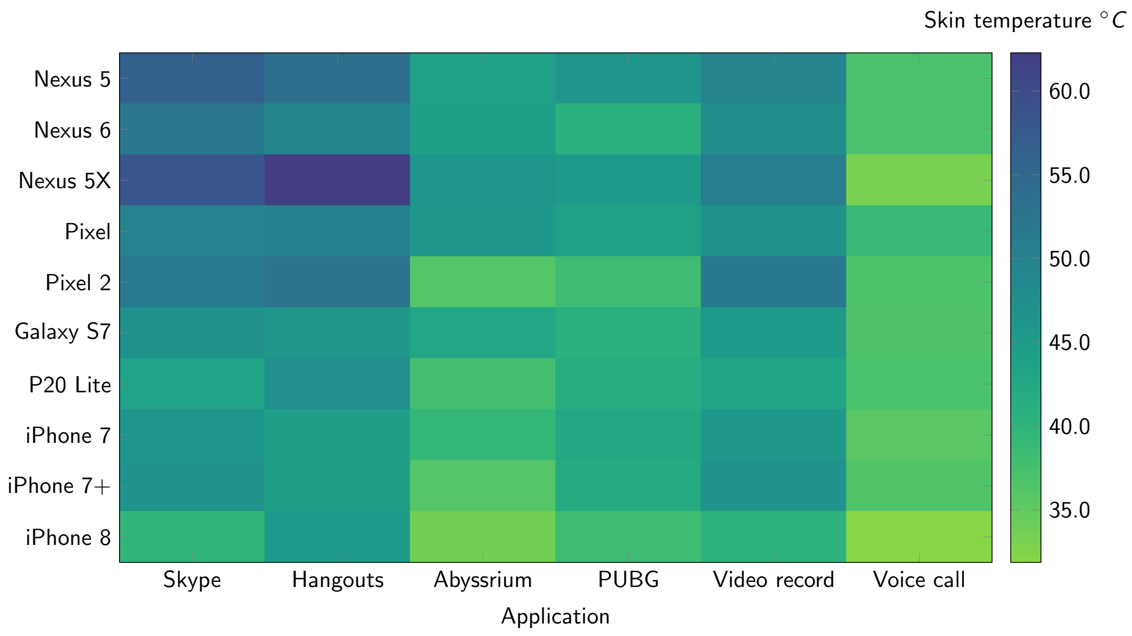


FIGURE 2.7: Steady-state skin temperature recorded across ten mobile phones during execution of everyday applications - data from [93].

opportunistic charging. Habitual charging refers to a recognisable pattern in charging over a number of days, for the users shown in Figure 2.6 this is normally overnight. Opportunistic charging refers to charging without a pattern, if the battery level is low then this is defined as necessary and if not then it is defined as unnecessary [121]. The takeaway is that whilst charging patterns can roughly be divided into patterns, habitual and demand-based, the mobile device battery life requirement is a moving target.

The upwards trend in SoC power in recent smartphones was highlighted in Figure 1.3 in Chapter 1. The emerging software applications discussed in Section 2.1.2 (3D games, AR and VR) have increased the computational requirements on the GPU without let-up in CPU requirements as was shown in Figure 1.2. When these applications are executed for moderate lengths of time this is known as sustained performance [92]. Sustained performance has created a new challenge for smartphones, high device skin temperatures. Device skin temperatures greater than $40°C$ start to cause heat sensation and then later discomfort for users holding the device [93]. The exact temperature threshold and therefore sustainable thermal envelope depends on both the material of the device and the user tolerance [93, 94]. Steady-state device skin temperatures reached across several mobile devices and applications are showcased in Figure 2.7. Thermal throttling issues are present even in less demanding applications and older devices, in this study they are most prevalent in applications with high camera usage.

High device skin temperatures may also manifest themselves as hotspots caused by individual components of the device. These components can include the SoC, WiFi chipset and camera [18, 93]. As these components are relatively small in area the heat is not dissipated across the whole area when it reaches the skin of the device. Other larger heat dissipating components such as the battery and display are less likely to cause hotspots [93].

Cooling solutions available in smaller mobile devices such as smartphones have historically used passive techniques [133]. These include heat-pipes to channel heat across the device [134] and heat-spreaders which act to reduce hotspots on the skin of the device [6]. Recently, some device manufacturers have created devices designed specifically for sustained performance applications such as mobile gaming and these have incorporated active cooling fans [135, 136].

In response to the power management challenges introduced by sustained performance applications, device manufacturers have refined the SoC architecture used in mobile devices. The first modification has been the integration of more specialised accelerators as the workload demands of emerging applications has become clearer. These specialised accelerators include further advancements in image processing acceleration to support up to 8K video recording as well as photo post-processing [137] as well as Neural Processing Units (NPUs) for Machine Learning applications [32, 137, 138]. Accelerators are beneficial for both battery lifetime improvement and device thermal management but also can improve performance sufficiently to support applications that would not otherwise be viable on a mobile device. The second modification has been to further segment CPU performance operating points to support sustained performance applications where power and performance must be delicately traded-off. Typically the switch has been from four big cores and four LITTLE cores to one or two big cores, two or three medium cores and four LITTLE cores [137, 138, 139]. Appropriately selected operating points for the medium cores allows performance to be sustained longer as the big core operating points are heavily weighted towards performance for executing short tasks with low latency. This modification has not been universal across all manufacturers, in fact the best performing SoC on the market has continued to use a big.LITTLE architecture without medium cores [32].

2.2 Dynamic Power Management

Operating systems shoulder the task of meeting the power challenges discussed in Section 2.1.3, battery life management and device skin temperature constraints. Choices early in the design process can be made to reduce overall power consumption; however, after these variables are fixed runtime management is implemented using a collection of techniques referred to as Dynamic Power Management (DPM). DPM existed before the widespread proliferation of mobile devices [33], but its use has expanded rapidly due to their adoption rates. There is a wide range of levers that may be used to reduce power

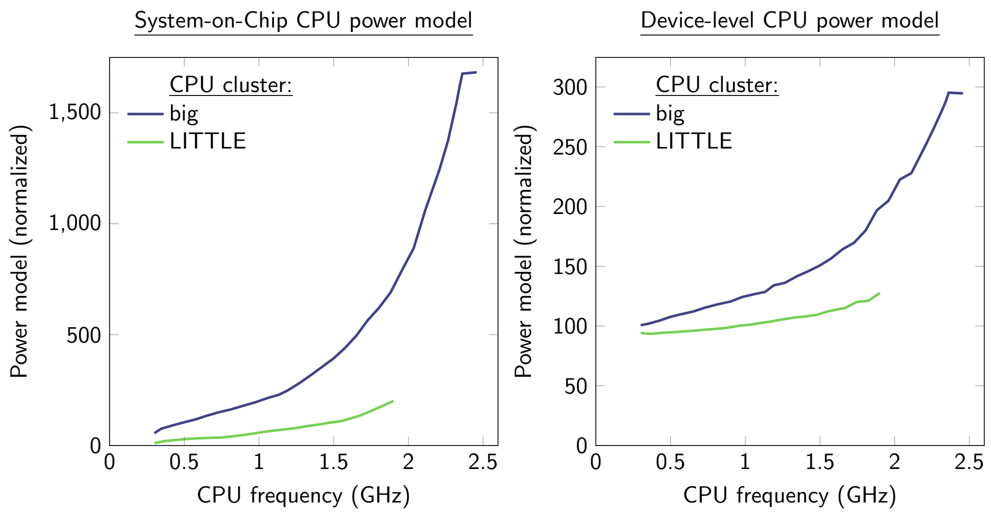


Figure 2.8: Pixel 2 power profiles driving thread scheduling decisions and the device-level power model driving battery management decisions - data extracted from the System-on-Chip power model using Android Debug Bridge.

consumption at runtime (Section 2.2.1) and various metrics that can be monitored as part of a control-loop to configure these levers (Section 2.2.2).

2.2.1 Levers for reducing power consumption

Ubiquitous in modern mobile devices, is Dynamic Voltage and Frequency Scaling (DVFS), the premise of which is by reducing SoC frequency of operation when not required provides the opportunity to reduce SoC Voltage and therefore power consumption. It is intuitive, upon observation of the SoC power model in Figure 2.8, to believe that DVFS can solve most of the power challenges in mobile devices alone. However, when compared to the device-level power model in Figure 2.8, it is clear that it is only part of the picture for device-level power management. Widespread diversity in device hardware, software applications and usage covered in Section 2.1 guarantees that optimal DPM levers and control will vary between use-cases. Scaling CPU frequency using DVFS can significantly reduce CPU power consumption but is only one of a range of levers available to reduce device power consumption.

DVFS has been applied to CPUs, GPUs, memory, interconnect, IP cores and other subsystems using the universal principle of slowdown when performance is not required. Workloads change rapidly as do their performance requirements; in some scenarios it can be challenging to update the DVFS operating point in synchronisation with these rapidly changing workloads. To overcome this, a higher-level DPM lever has been developed [140, 141] which sets minimum and maximum bounds for the DVFS operating point to offer performance flexibility with a slower update-rate.

Slowdown in performance using DVFS is a good match for some workload performance patterns, however, many mobile workloads are interactive which means they often contain periods of time with no work to be done [113]. In the extreme case parts of the SoC may be powered down entirely which also reduces static power consumption [142]. In mobile devices this is normally restricted to IP cores and other intermittently used hardware [102], this approach is commonly referred to as dark silicon. There are challenges including overheads in energy and time associated with fully powering down SoC subsystems, so it doesn't provide a one-size-fits-all approach to exploiting idle periods in workloads [143]. Instead CPU cores are designed with a multitude of low-power states that fall short of fully powered down the CPU. These include idle and sleep states which progressively wind down operations of the CPU between fully operational and fully powered down. The lowest-power states represent a significant power saving over operational modes which has led to a technique called "race-to-idle" or "race-to-sleep" where it is preferred to reduce reliance on DVFS and instead seek to execute a task quickly such that a low-power state may be entered [144, 145]. The efficacy of "race-to-idle" techniques depends on the cost in power and time to transition between these low-power states.

The rise of heterogeneity in both asymmetric CPU design [146, 147, 148] and across other subsystems of the SoC [117, 118, 149, 149] presents choices in where and when tasks may be executed. Where a task is executed is referred to as mapping and it is not limited to the boundaries of the SoC, other ICs or even offloading to the cloud may be selected for execution [128, 150]. When a task is executed is referred to as scheduling, scheduling initially existed as a method to share CPU time between competing applications on the same CPU core to increase fairness. It has now evolved into a DPM technique that can be used to optimise for energy and thermal targets [34, 151].

DVFS, idle states, mapping and scheduling are all commonly applied in general-purpose computing systems, there are also DPM levers that are more specific to mobile devices. Data movement between disk, memory and SoC is costly in both time and power [115], if data movements are known ahead of time they can be executed in a more efficient manner through prefetching. This can be internal to an application such as loading a file into memory [164] or can involve pre-loading entire applications into memory if their use is predicted in the near future [165, 166]. In the same way application use can be predicted, the absence of use can also be predicted. Background applications can still consume computation and memory of a mobile device, both throttling and removal of background applications have been proposed as DPM levers [96, 131]. If the performance requirements of foreground or background applications cannot be met due to power constraints, then one option is to reduce the performance requirements which can be softer than the power constraints. This can involve reducing the quality of media such as images, video or audio or in the case of approximate calculations such as ML acceleration go further to reduce the accuracy of computation [128, 167]. Linked to

DPM lever	Description	Reference
Dynamic Voltage and Frequency Scaling (DVFS)	CPU (GPU, memory, and others) frequency of operation reduced allowing a reduction in Voltage.	[31, 35, 97, 140, 141, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162]
Dark Silicon	Subsystems of the SoC, often accelerators, are powered down to reduce static power consumption.	[31, 142, 143]
Idle/sleep states	Low-power states are used in between computation in a process named "race-to-idle" or "race-to-sleep".	[33, 144, 145]
Mapping and scheduling	Tasks are mapped and scheduled on asymmetric processors on-device or offloaded to the cloud.	[34, 117, 118, 128, 146, 147, 148, 149, 150, 151, 163]
Prefetching	Data or applications are pre-loaded from disk into memory using prediction models of future usage.	[164, 165, 166]
Background applications	Background applications have their computation throttled or are removed from memory.	[96, 131]
Application quality	Degradation of application quality requirements is used to reduce the demand for computation.	[128, 167]
Display brightness and colours	Mobile device displays consume a high proportion of the power budget which could be reallocated.	[95, 119, 120, 168]

Table 2.1: Taxonomy of Dynamic Power Management levers commonly used in mobile devices to reduce power consumption.

application quality is display quality, the display has been identified in multiple studies as consuming a significant proportion of the power budget. The most common DPM lever applied is to reduce the brightness of the display [119, 120, 168]; however, it has also been proposed to approximate the colours of pixels on the display to save power [95].

2.2.2 Control of levers

Use of a DPM lever is the end-product of a DPM control loop, in order to reach this point, a decision must be made determining the correct setting for the lever. The process carried out to achieve this typically takes the form of a policy that monitors one or more metrics of the SoC or mobile device [33, 36]. In addition to the diverse array of DPM levers discussed in Section 2.2.1, there are equally a diverse range of metrics and policies for setting them that will be discussed in this Section. Levers, metrics and policies may be combined in a custom fashion to meet DPM requirements for a particular device, workload and user.

Setting of DVFS levers is a natural fit with monitoring utilisation of resources in the System-on-Chip as the metric. For example, the required CPU core frequency is correlated to the utilisation percentage recorded for that CPU core at that instant [40, 51]. The same relationship extends beyond CPUs to GPUs, IP cores and other resources with measurable utilisation [169]. In some cases, the overall utilisation of a resource is not sufficiently granular to determine the exact performance requirement. When compute resources are stalled waiting for the completion of accesses to memory this can manifest as high utilisation when in fact little computation is required at that instant. More granular information can be obtained from hardware performance counters including counters for cache and memory accesses [170].

DPM does not always seek to reduce performance when it isn't required, if power constraints are overwhelming then performance must be reduced regardless as is the case in thermal throttling [35, 93]. Thermal throttling may monitor metrics for the device skin temperature and the temperature of internal components [94, 97, 154, 155, 168, 171]. Information gleaned from these metrics may be used before the temperature exceeds constraints to mitigate sudden losses in performance. Equally, for the battery lifetime management the battery charge level may be monitored closely even before low-charge levels are reached to mitigate the risk of running out of charge [98, 167].

Real-time measurement of metrics for both thermal and energy management is important as device usage behaviour is so distinct between users [119, 121]. This usage behaviour can also be observed directly by monitoring user inputs that is used by prediction models to predict the probability of future usage. These probabilities can be as granular as predicting the next touch input [172, 173] or at the other end of the spectrum high-level decisions such as which applications will be opened next [166].

Execution time of applications is also monitored as a metric to inform future predictions [174]. In situations where the execution time can be modelled accurately it can be used as an oracle for DPM to track required performance [156], however, the complexity of modern SoCs prove a challenge to this. This complexity also applies to predicting user input and leads to the more accurate metric of detecting events as they occur

DPM control inputs	Description	Reference
SoC resource usage	Utilization percentages are sampled from SoC subsystems such as CPUs and GPUs.	[36, 38, 40, 51, 165, 169, 170]
Device and SoC temperature	Device skin and internal SoC temperatures are monitored to avoid thermal constraint violations.	[94, 97, 154, 155, 168, 171]
Battery charge and lifetime estimate	Battery lifetime estimates are calculated to predict shortfalls requiring additional power savings.	[98, 167]
Predicted user input	Dynamic Power Management varies between workloads which may be predicted along with user inputs.	[119, 121, 166, 172, 173]
Predicted execution time	Tasks can be scheduled as in a real-time system if execution times are known or are predicted.	[156, 174, 175, 176]
Detected events	Interrupt-based detection is more accurate than prediction models but is not in advance of the event.	[177, 178]
Application performance	Performance metrics are measured or queried including execution time, throughput and latency.	[140, 160, 179, 180]

Table 2.2: Taxonomy of Dynamic Power Management control inputs for use in determining runtime performance and power requirements for mobile devices.

[157, 177, 178, 181]. Unfortunately interrupt-based detection of events does not provide the advance notice offered by predicting events before they occur.

Reverse-engineering application performance information through prediction or detection of events is a challenge for implementation. Alternatively, this information can be sought directly from the application or Operating System (OS) through use of an Application Programming Interface (API) [160]. Application execution-time, throughput, latency and custom metrics may be collected through querying the API in real-time. In practice for commercial mobile devices this has been implemented primarily for interfacing with the OS [140, 179] rather than in applications [160, 180]. The role of system integrators in providing and using APIs for DPM can result in advantages for vertically integrated device manufacturers with control over hardware, system software and applications that can be used to improve user Quality of Experience over non-vertically integrated device manufacturers.

Initial iterations of DPM control policies relied on a heuristic-driven approach [38, 39, 182, 183]. The policies were designed to set DPM levers based on pre-configured resource utilisation thresholds and timers to track the performance required by tasks. Recent evolutions of these control policies remain in use in many current mobile devices such as the *interactive* governor [40] and Energy Aware Scheduling [51]. The *interactive* governor extends the heuristic-driven approach to wake-up the CPU when user inputs are triggered [157, 181, 184, 185]. This extension incorporates user satisfaction into the decision-making process but doesn't go as far as to monitor any user metrics directly [95, 186]. The Energy Aware Scheduling policy enhances the accuracy of measurement of resource utilisation and incorporates an energy-efficient mapping process to exploit heterogeneity in asymmetric CPU cores [51].

DPM control techniques proposed in literature include mobile device domain-specific and application specific enhancements. The graphics pipeline rendering frames at 60 Frames-Per-Second (FPS) in mobile devices is well documented, this can be exploited to segment DPM control into a frame-by-frame decision process [140, 161]. Segmentation is also possible by application [158], in particular for mobile devices, sustained and non-sustained performance applications can be selectively throttled [187] to mitigate performance losses under thermal throttling. Furthermore, workload phases from a single application may be segmented [169, 188] to overcome the variety in workloads exhibited within applications.

Heuristic-driven approaches have been substituted for formal control theory approaches including Single Input Single Output (SISO), Multiple Inputs Multiple Outputs (MIMO) and Structured Singular Value (SSV) control [189, 190, 191]. The co-operation between heterogeneous processors within an SoC has led to techniques coordinating DPM between multiple subsystems from balancing CPU and GPU power allocations [97, 141] to a leasing model for resources run by the OS [192]. An emerging theme is Hierarchical Control Systems (HCSs) which seek to maintain coordination across the SoC while reducing latency for low-level DPM lever control [193, 194, 195].

Machine Learning (ML) control is also increasingly implemented for DPM control, in particular model-free Reinforcement Learning (RL) algorithms such as Q-Learning are most prevalent [52, 194, 196, 197, 198]. Model-free RL is a natural fit for DPM control as it doesn't require a model of the environment to be constructed, which would be a challenge given the complex nature of a modern SoC. Training of RL DPM control policies has been accelerated with imitation learning, where an oracle or heuristic-driven policy is mimicked by the RL policy to bootstrap the learning process [169, 199]. Recurrent Neural Networks (RNNs) have been demonstrated for use in both predicting future workloads and making decisions based on the result [53, 200].

Commercial sensitivity has meant that some ML techniques used by device manufacturers to optimise power management are not documented publicly. Among the publicised ML power management techniques that have been implemented in modern mobile devices is a DVFS controller that traces render calls between an application and GPU as the input to a neural network and the output is a DVFS operating point for the GPU [201]. The training of the neural network is specific to an individual SoC and application so has not been widely applied due to this constraint. The use of individual training per application allows over-fitting of the performance characteristics of that application rather than attempting to cover the diverse range of mobile applications with a single controller. Another publicised ML power management technique is at the system level for managing battery usage of background applications in Android [202]. Suppression of wake-up and background processing privileges is used to reduce the battery consumption whilst ensuring these privileges are available to applications most likely to be used by each individual user. A Convolutional Neural Network (CNN) is used to predict which applications the user is most likely to use within the next few hours.

2.3 Performance evaluation

Performance evaluation in general-purpose computer systems is challenging, there are widespread problems in producing repeatable and representative benchmarks [203]. Mobile devices are no exception to this rule, the metrics they are evaluated against are however, different to those of a general-purpose system [102]. From an application perspective these are focused on the performance exposed to the user in foreground applications (Section 2.3.1). For power, a combination of measurement and models are used to estimate energy and thermal results (Section 2.3.2). Tying the application and power metrics together is a suitable framework for evaluating the performance of and therefore tuning of DPM policies for mobile devices (Section 2.3.3).

2.3.1 Application performance measurement

Software applications executed on mobile devices are interactive, they are subject to variation in external resource access-times and user inputs [113]. If the external resource is access to networks such as the internet then access-times can vary geographically due to Received Signal Strength Indicator (RSSI - signal strength) [204, 205] and temporally due to network congestion at peak periods throughout the day [206]. The diversity in user inputs and the importance of modelling this has previously been covered in Section

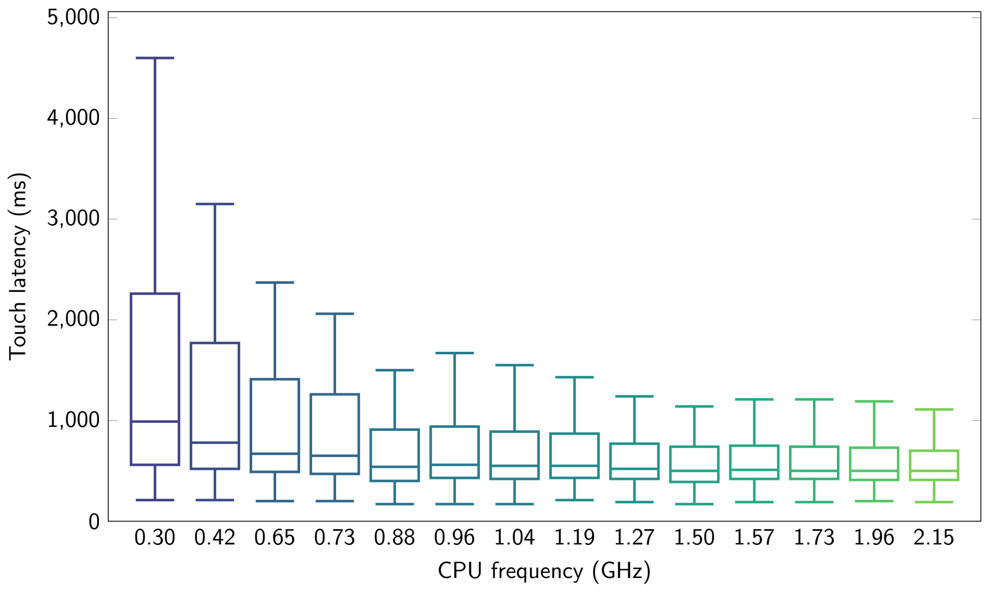


FIGURE 2.9: Latencies measured with a camera for a UI application page load triggered by user input - data from [211].

2.1. Application performance metrics are referred to as Quality of Service (QoS) metrics and their mapping to user experience results in Quality of Experience (QoE) metrics which will be detailed later in this section.

Interactive applications must be evaluated against different QoS metrics than throughput which is the default in application performance measurement in general-purpose systems [102]. The first of these performance metrics is latency [102, 147, 148, 185, 207], users notice high latency but there is a floor at approximately 20ms below which users are less affected [208]. High latency can also increase the time users take to perform a task [209] which aside from reducing QoE metrics will mean that the task will likely consume more energy. Latency can be aggravated by background tasks throttling the performance available to foreground applications when executing concurrently [210].

Latency may be measured internally by applications or externally using video recording either on the mobile device or using a camera. In Figure 2.9, a video camera was used to detect changes on the display of a mobile device which indicated that a User Interface (UI) application action was completed [211]. The time between the automated action and the recorded timestamp of the completed action in the video was registered as the latency of the action. The CPU frequency of the mobile device was varied from a minimum of 0.3GHz to a maximum of 2.15GHz and the distribution of latencies measured. Below 0.88GHz there is a noticeable degradation in the tail (worst-case) latency observed, far more pronounced than the degradation in median latency. This long-tailed distribution is typical in latency measurements and requires that a distribution not average value is measured [102].

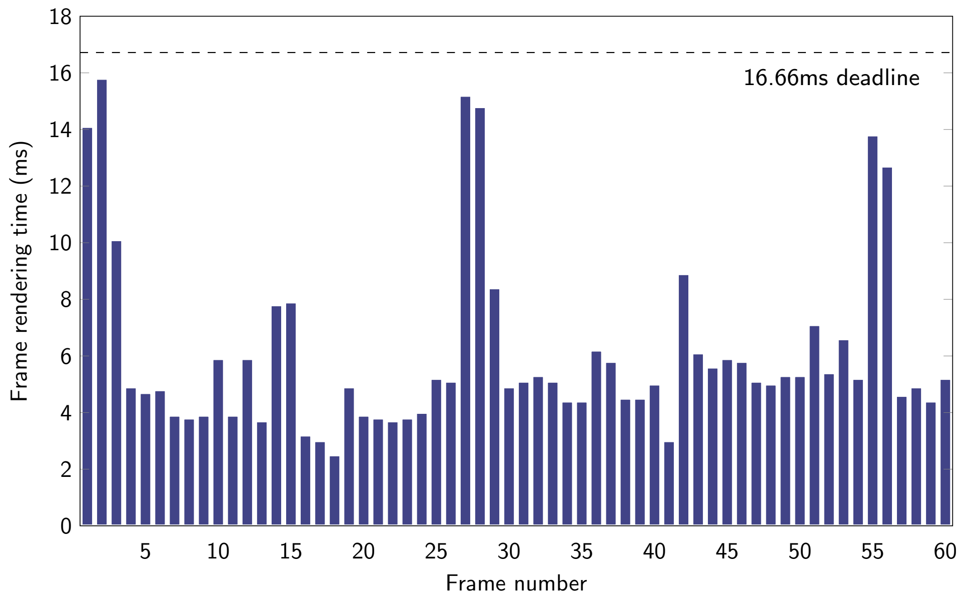


Figure 2.10: Frame rendering times measured for the Google Now Launcher for the duration of three swipe interactions - data from [140].

Webpage loading is a special case for latency measurement, on one hand it is often triggered by the same type of user touch input to the device, on the other, the user anticipates that it may take a period of time longer than 20ms to load [212, 213]. In one study the target load time is quoted at 2s [214], in another 3s [152]; there is also complexity from varying network conditions to be considered [204, 206]. Webpage loading also suffers from interference from concurrent execution of background tasks due to contention on the memory subsystem [215].

A common misstep in evaluating frame-based applications in mobile devices is to measure average frame rate [23, 97, 141, 147, 148, 154, 155, 176]. Users are in fact far more susceptible to the stability of the frame rate and in particular, frames that miss their deadlines which are referred to as frame rate janks [102, 140, 162, 177, 216]. Based on the assumption a mobile device is executing at 60 FPS, the deadline for each frame to be rendered is 16.67ms [102].

An example is provided in Figure 2.10 where the individual frame rendering times of a UI application are recorded [140]. In this instance, each frame is inside the 16.67ms deadline, however, it is observable in the three time periods where the user is inputting a swipe action to the application, the slack before the deadline is reduced. If the average frame rate was recorded, then this effect would not be visible. Frame rate janks like latency are linked to a long-tailed distribution, in this case it is the distribution of frame-rendering times. The 90th, 95th or 99th percentile frame-rendering time is often measured as proxy for frame rate janks [102].

Latencies and frame rate janks are the key application performance and therefore QoS metrics for interactive mobile applications. Users are not exposed to the raw QoS metrics

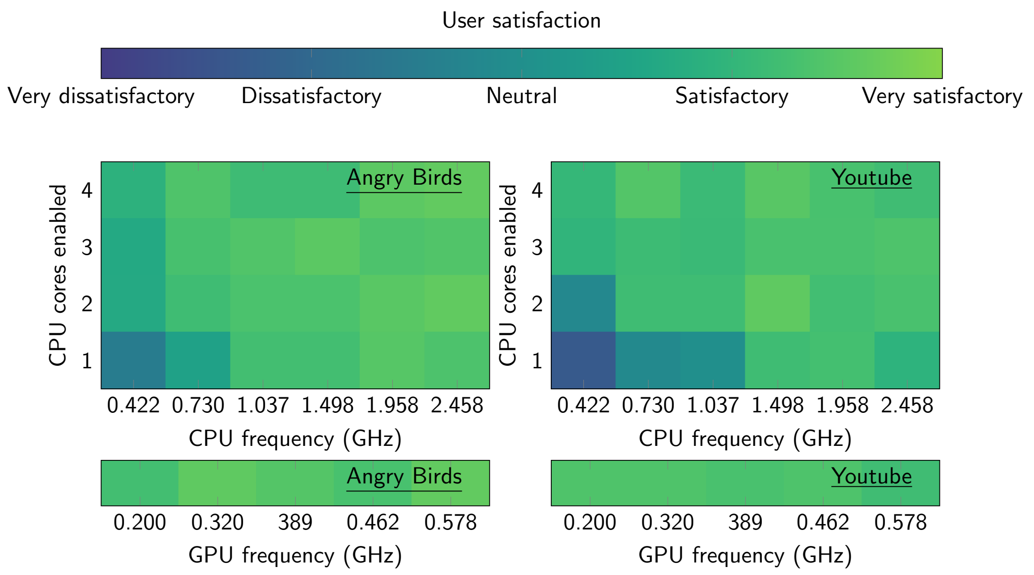


FIGURE 2.11: User satisfaction ratings for two mobile applications as number of CPU cores, CPU frequency and GPU frequency are varied - data from [31].

on their devices so if these metrics are below a floor as with 20ms in the case of latencies [208], their satisfaction is not affected. Equally if a Quality of Service metric is woefully bad then small changes in this metric are unlikely to better or worsen user's satisfaction. The mapping from QoS to user Quality of Experience (QoE) reflects these phenomena [217, 218].

Optimisation of system focused on a QoE metric will both improve user experience but also create opportunities to exploit the non-linearity in the QoS to QoE mapping [218]. QoE metrics can either be measured directly using user surveys which query their satisfaction when using a mobile device [31, 219] or estimated based on character traits or other information recorded about the user [95]. It has been proposed that some users are more tolerant of poor QoS metrics than others linked to their personality [95]. Results from a user QoE survey for mobile application performance is presented in Figure 2.11. Users were asked for a satisfaction rating on a scale from very dissatisfactory to very satisfactory when presented with mobile applications with different performance settings and therefore QoS metrics [31]. The study showed that for incumbent mobile devices user QoE varies more with CPU rather than GPU performance. Both the number of CPU cores enabled and the CPU frequency affected user satisfaction and therefore QoE.

2.3.2 Power measurement and modelling

Irrespective if the optimisation target is energy, temperature or both as described in Section 2.1.3, in a mobile device this target will be linked to power consumption. Power can either be measured or be estimated using a model, in this section the various methods of achieving this are detailed along with their characteristics.

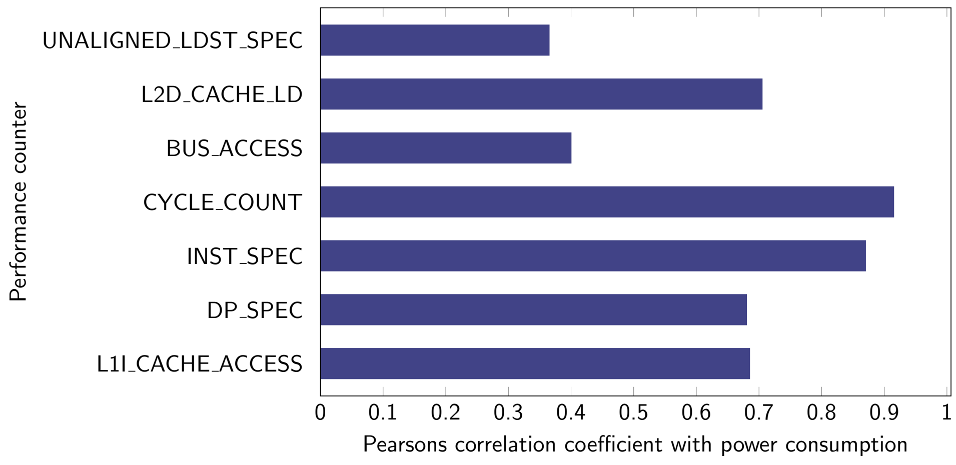


FIGURE 2.12: Seven performance counters selected for an A15 power model, along with the Pearson's correlation coefficient with power consumption - data from [221].

There are two types of measurement used to determine energy consumption of a mobile device. The first of these, sensors on the power rails used by the SoC, is most common on development boards and can be used to breakdown the SoC power consumption as far as the SoC subsystems are segmented onto separate power rails [23, 95, 96, 97, 98, 190, 191]. For the purposes of convenience, the power sensors typically interface with the SoC directly to share the data collected. Unfortunately, this can limit the sampling rate possible without adding overhead, this is a problem for evaluating energy for interactive applications which ramp up and down rapidly.

Energy can also be measured at the device-level, this is representative of the energy that would be drained from the battery during usage. This measurement incorporates energy consumption from components such as the display that are not included in SoC power rail measurements [23, 95, 96, 97, 98, 190, 191]. Device-level energy measurements are most commonly used for real mobile devices; however, the construction of modern smartphones has made this process more challenging.

In some situations, SoC power sensors and device-level measurements are not available or are insufficiently granular. Alternatively, an energy model can be developed using the most representative real-world data available, this can also be during simulation before device development [46]. Previously, such energy models have been derided as insufficiently accurate outside of controlled conditions [220], whilst this hasn't been refuted, newer models have demonstrated accuracy to within 3% [221]. A model has also been developed to separate dynamic and static power consumption under varying ambient temperature [222].

CPU energy consumption can be modelled using a variety of techniques including CPU performance counters [221], tracing of CPU frequency and idle states [223], tracing of system calls [224] and application traces [213]. Performance counters selected for an A15 CPU power model are shown in Figure 2.12 along with their correlation with

Component	Variable(s)
big CPU cluster	CPU frequency and activity
LITTLE CPU cluster	CPU frequency and activity
GPU	GPU frequency and activity
Video decoder IP	Codec resolution (4K, 1080p, ...)
Wi-Fi chipset	Throughput - upload and download
Battery charging	Normal charging or quick charging
Camera	Recording resolution (UHD, HD, ...)
OLED display	Brightness

Table 2.3: Device thermal model components and their respective variables for the Galaxy S8 smartphone - data from [232].

power consumption measurement. Selection was optimised by reducing multicollinearity between performance counters as the counters with highest correlation are not necessarily independent [221]. Furthermore, high correlation doesn't necessarily guarantee an accurate and stable model under workload and device temperature variation. Device-level energy consumption requires individual models for each of the components in a mobile device which may be summed to achieve overall energy consumption [119, 132, 223, 225, 226, 227].

Thermal measurements for mobile devices may also be conducted using two techniques, the first of these being temperature sensors such as thermistors. Temperature sensors are installed internally on Printed Circuit Boards (PCBs) and by components throughout mobile devices [124, 153, 171, 228, 229, 230]. Commonly these internal sensors are used to estimate external skin temperature, however, a study has also used an external temperature sensor for this purpose [230]. The second technique, effective for measuring external skin temperature only, is to use an Infrared Radiation (IR) camera to record thermal radiation emitted by the skin of the mobile device [18, 93, 133, 229, 231]. Thermal measurement by external camera may only be used in static configuration of DPM, a model based on internal sensors may be used for dynamic configuration.

Internal temperature sensors are more effective than internal energy sensors as changes in temperature happen far slower than for energy, the only benefits in models for these sensors are for further granularity such as a SoC temperature breakdown [231] or to predict future temperatures [153]. The logistics of recording IR video of a mobile device for sustained periods of time has meant that device skin temperature models are more prevalent for use outside of laboratories [93, 230, 232]. The list of components and their variables used in a device skin temperature model for a modern mobile device is showcased in Table 2.3. These models are subject to effects caused by material properties specific to each mobile device [229, 233, 234] and environmental effects such as ambient temperature and how a mobile device is held in the hand [93, 171].

2.3.3 Tuning Dynamic Power Management

The tuning process for DPM policies reflects the confluence of all the preceding sections. Consideration must be made for the hardware and software classifications of mobile devices detailed in Section 2.1 along with the challenges they present. The nature of DPM policies and the levers they exploit described in Section 2.2 is important to understand the parameters that will be exposed for tuning them. Finally, the performance and power metrics presented earlier in Section 2.3 are inherent to comparing the performance between two policies.

There is limited previous literature with a direct focus on tuning DPM policies, particularly with regards to mobile devices, however, far more research has been conducted into tuning computer systems and software in general that is applicable. DPM policies are tuned in a similar fashion to many other control systems, they expose parameters such as thresholds, timers and feature flags [47, 48].

System software applications also expose a large quantity of these parameters to be configured by system administrators. One perspective of the presence of these parameters is that this is a negative feature [108]. A study found that over 50% of parameters for system software can be configured by less than 1% of the users in a commercial setting, for the same software less than 10% of parameters were configured by more than 50% of the users, see Figure 2.13. The study used this evidence to claim that it was unwise to assume that users would be willing to set these parameters themselves and to reduce the number. In this thesis another perspective is presented that embraces parameters of mobile DPM policies and the opportunities they provide.

A timeline of processes throughout the design roadmap during which DPM policies can be tuned was presented in Chapter 1, Figure 1.1. These included Design Space Exploration (DSE), static and dynamic configuration, the key distinction separating these processes is the availability of accurate characterisation data for both SoC performance and the workload being executed. The further along the design roadmap the process sits, the less reliance on models which are replaced with measured data. In the DSE process, both performance and workload data are models, the static configuration process can make use of SoC performance measurements and the dynamic configuration process adds workload data measured instantaneously.

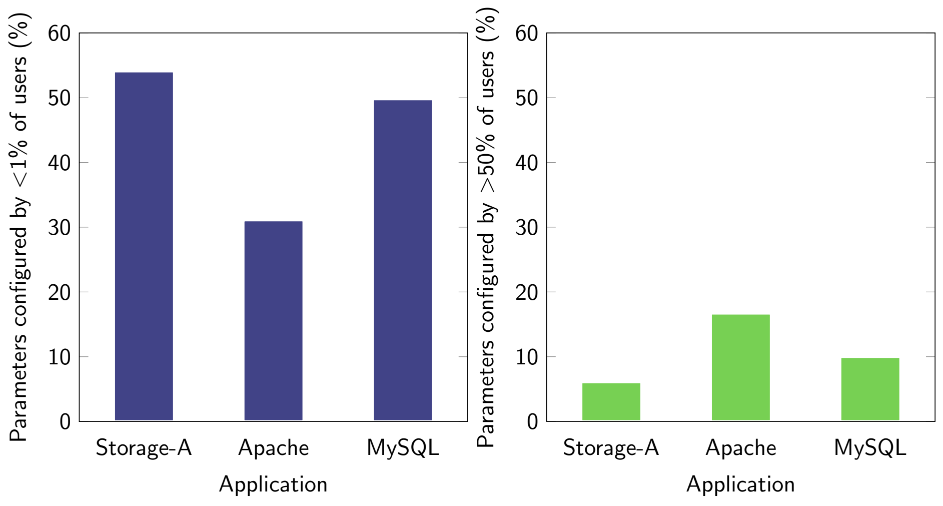


FIGURE 2.13: Software configurations analysed for parameters set by users for three system software applications - data from [108].

Tuning of mobile DPM during the DSE process is not strictly limited to parameter configuration, as elements of the wider SoC design including DPM may not be fixed. Synthesis of Register Transfer Level circuits during the SoC design roadmap has been tuned both for synthesis parameters [235, 236, 237, 238] and parameters exposed by IP cores [56, 85, 239]. These processes can be fully automated [235, 240] or make use of input from a human expert [241]. However, specific to DPM, the DSE process provides the opportunity for mathematical optimisation of potential DPM policies and the levers than underpin them [49, 50].

Static configuration of DPM policy parameters benefits from the most evidence of prior use out of the three processes. The introduction of SoC performance data characterised from real hardware allows trade-offs between performance and power to be fixed for the first time. There are industry tuning guides for DPM policy parameters during the static configuration process, they rely on guidance from the developers and expert decision making from humans [47, 48]. The remaining evidence comes from system software configuration, where systematic static configuration of software parameters has been applied successfully [105, 242, 243, 244]. In a study comparing systematic tuning to expert manual tuning by humans, a commercial webserver application was tuned. The results from the same study presented in Figure 1.9 were impressive in their magnitude, throughput improvements of over 300%, but that was compared to the default configuration. In Figure 2.14 the systematic tuning showed improvements over the manual tuned configuration in all five metrics, when the manual tuning was considered to be an optimal configuration. Systematic tuning of parameters is superior on all five metrics with the most important, throughput, showcasing a 4.1% improvement. There have been several techniques demonstrated for optimising the static configuration of software parameters. These techniques include selecting the right parameters [106, 109, 245], searching the parameter space, [105], modelling the parameter space [243, 244, 246, 247]

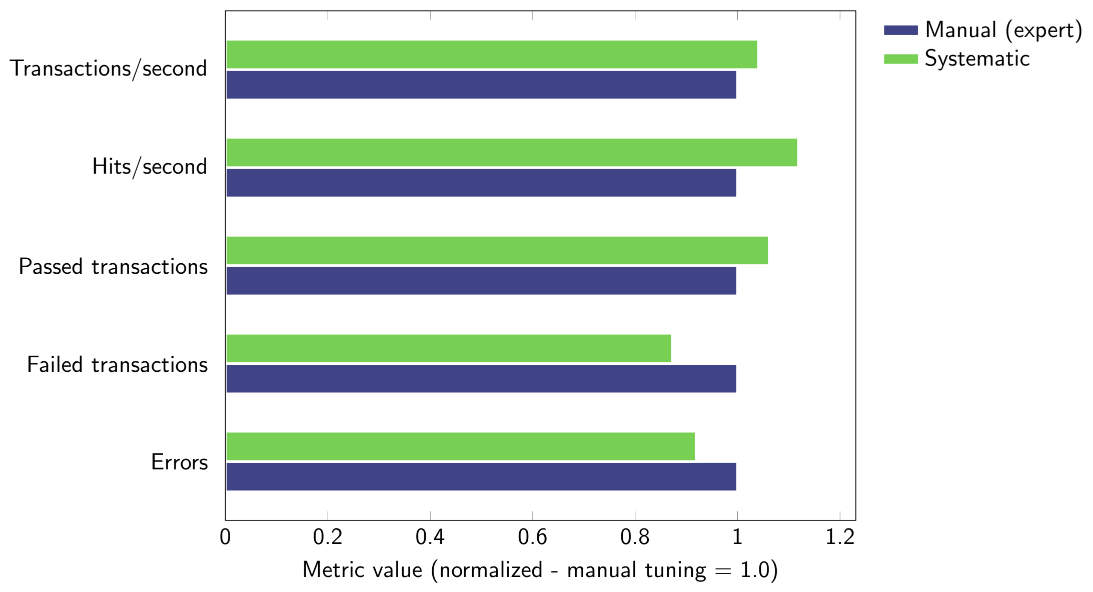


Figure 2.14: Tomcat webserver application optimised using systematic tuning of parameters compared with expert manual tuning - data from [105].

and predicting interactions between parameters [248]. Static configuration has also been applied to approximate computing applications where quality degradation is permissible [249, 250].

There remains sufficient low-hanging fruit in static configuration of computer system parameters such that dynamic configuration is rarely applied. The best examples are in distributed systems where nodes may be added or removed at runtime invalidating previous characterisation results used for static configuration [251, 252, 253]. The same logic could be applied to new applications entering and leaving the work-queue of a mobile device; but this has yet to be studied.

Regardless of which tuning process is used and whether simulation or real hardware is used, representative workloads must be executed. In computer architecture research it is widespread to use off-the-shelf benchmarks such as SPEC and PARSEC [100, 101, 103]. For mobile device research the results from these benchmarks are misleading [42, 102], showcasing minimal characteristics in common with results using real mobile applications [31, 35, 96, 97, 140, 141, 154, 155, 162, 187, 211, 254] To increase the quantity of data available, either automation may be used [31, 99, 211] or synthetic traces may be derived from representative data [255, 256].

In a similar vein to workload selection, hardware choices are equally important to achieve representative data [102]. An ideal situation permits large user studies to be conducted with real mobile phones [119, 121, 131, 132], logistically this can be both time consuming and expensive to organise but the takeaways can be significant and have laid the groundwork for much of this literature review chapter. Experimentation at scale has also been achieved through outsourcing surveys of recordings from real mobile devices

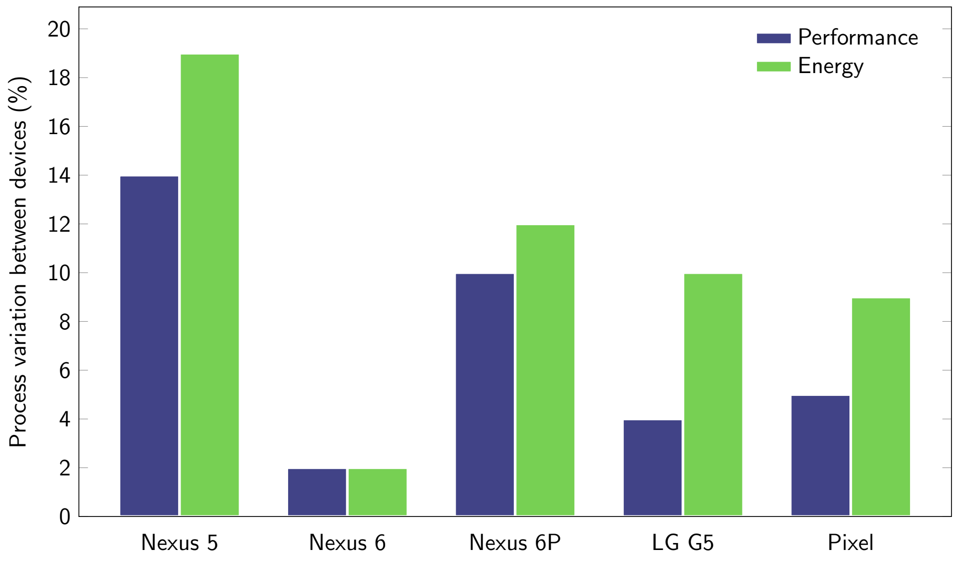


FIGURE 2.15: Mobile device System-on-Chips aren't necessarily identical, this study demonstrated performance and energy variation due to silicon binning linked to process variation of up to 19% - data from [228]

to internet workers [31, 219]. In industry some companies have invested in mobile device laboratories hosting thousands of devices to run experiments on [110, 111, 112]. Currently these mobile device labs are focused on compatibility and regression testing but could feasibly be extended to tuning experiments.

Experimental research into mobile devices is approximately divided between two types of platform, development boards and real mobile devices. Development boards can provide functionality such as power sensors that means they are easier to use for experiments [23, 95, 96, 97, 98, 190, 191], however, they do not display characteristics that are visible for studies that are run on real mobile devices [35, 140, 147, 154, 155, 160, 168, 200, 211]. The obvious omission is that device skin temperature throttling does not exist when using development boards even if the same SoC is used [35, 93]. There are others such as environmental conditions including ambient temperature on power consumption [44, 45, 222]. The most obscure effect observed by comparing real mobile devices is the silicon binning affecting mobile SoCs, in Figure 2.15, process variation is shown to lead to a performance and energy variation of up to 19% [228, 257]. Between three and five of each device were compared which implies the maximum bound may be higher if SoCs from only a fraction of the total number of bins were tested.

2.4 Discussion

This chapter has presented an overview of mobile devices from a hardware and software perspective and highlighted the energy and thermal challenges presented by the mobile device form-factor. The wide array of DPM policies and levers that underpin them was introduced, these are used to tackle the power challenges faced. Also detailed is how interactive workloads executed on mobile devices are evaluated in performance and power in a different manner to general-purpose computing systems. The combination of all the preceding knowledge is an opportunity to tune DPM for mobile devices in a systematic fashion. System-software applications have been successfully tuned using the parameters they expose and mobile device DPM policies already expose similar timers, thresholds and feature flags yet they have been tuned manually up to this point in time. In this thesis, this opportunity is explored, and new techniques are developed to enable its use for tackling the existential problem of Tuning Dynamic Power Management for Mobile Devices.

Chapter 3

Interactive and Dynamic Workloads: The Case for Tuning DPM

Previous research and overall market trends in Chapter 2 pointed towards an opportunity in tuning mobile device Dynamic Power Management (DPM) policies. It was revealed in Section 2.3.3 that there is limited prior research focussed on this area despite a swathe of related work. This chapter provides empirical evidence supporting the case for tuning DPM policy parameters, when executing interactive and dynamic workloads on mobile System-on-Chips. In Section 3.1, a framework is presented for static and dynamic configuration of DPM policy parameters under user inputs and resource access-time variability (Section 3.1.1) and a weighted function to trade-off user Quality of Experience (QoE) and energy consumption is used to implement this for efficient execution of a web-browser application (Section 3.1.2). In Section 3.2, synthetic benchmarks are developed to simulate dynamic unseen workloads (Section 3.2.1) and an architecture for dynamic configuration of DPM policy parameters is proposed with no requirement for offline profiling (Section 3.2.2).

3.1 Efficient execution of an interactive web-browser application

Software applications executed on mobile devices showcase different characteristics than those executed on general-purpose computer systems (Section 2.1.2) and are evaluated using different metrics (Section 2.3.1). An experiment into efficient execution of web browser workloads is conducted in this section inducing variability in user input and

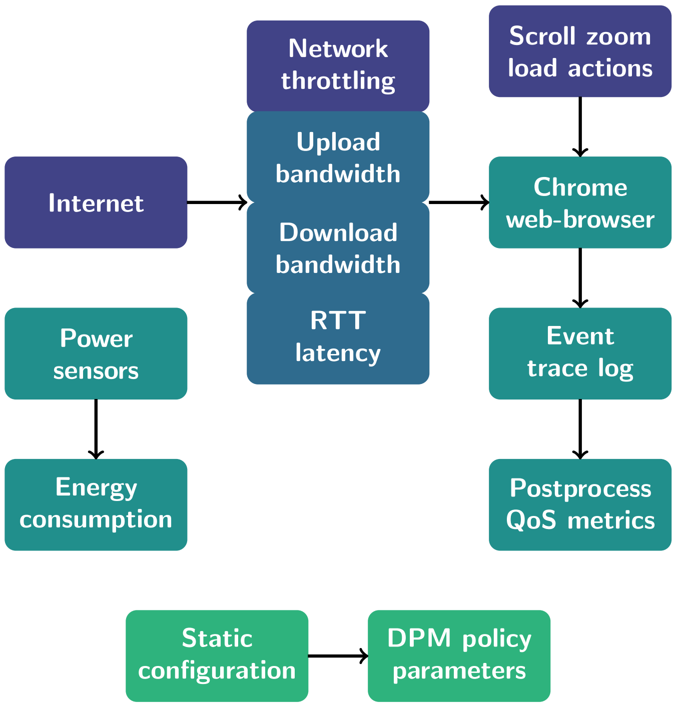


Figure 3.1: Experimental setup for evaluation of user input and resource access-time variability on Dynamic Power Management policy parameter configuration.

resource-access times native to mobile applications (Section 3.1.1). The resulting framework for static and dynamic configuration of mobile DPM policy parameters under these constraints is evaluated against user QoE and energy consumption resulting in a Pareto-optimal curve of operating points (Section 3.1.2).

3.1.1 User input and resource access-time variability

Interactive workloads are dependent on interaction between different subsystems of the System-on-Chip (SoC), peripherals, external resources and the user such as touch input during web-browsing. Inevitably, a subset of interactive workloads is affected by delays caused by data unavailability, e.g. loss or delay of data packets during voice-over-IP. At the same time, the system is required to respond quickly upon data retrieval to ensure that the user QoE metrics (frame rate, latency, etc.) are not degraded. To simulate this environment an experimental setup was created using the ODROID-XU3 development board running the Android 6.0 Marshmallow Operating System shown in Figure 3.1. The Google Chrome web-browser was used along with Chrome DevTools used to emulate user input and throttle network resources, power sensors on the development board were used to record energy consumption.

The Chrome DevTools WebSocket protocol enables repeatable user input to be emulated, such as touch gestures to the screen display. User actions available include loading webpages, scrolling through webpages and zooming in on webpages. Additionally, resource

Parameter	Category	Lower bound	Upper bound	Default
timer_rate	DVFS	1,000	100,000	20,000
target_loads	DVFS	60	99	99
scaling_max_freq	DVFS	800,000	2,000,000	2,000,000
min_sample_time	DVFS	5,000	500,000	80,000

TABLE 3.1: Parameters selected from the *interactive* Dynamic Voltage and Frequency Scaling governor to be configured.

access-time variation for mobile network conditions is induced using the network emulation which artificially throttles content loading time by adding to the Round-Trip Time (RTT) latency and limiting upload and download bandwidths. Events collected in the web-browser trace log are post processed to extract rendering frame rate and content loading times for various web-pages requests in order to measure the QoS for each individual emulated user input. At the same time, through a device driver Android provides direct access to the on-board Exynos-5422 power sensors for energy consumption measurements.

The default Dynamic Voltage and Frequency Scaling (DVFS) governor for Android 6.0 Marshmallow is the *interactive* governor, which was selected as the target DPM policy to be tuned for the experiment. The parameters of the *interactive* governor selected for tuning are listed in Table 3.1. The default values are listed along with the minimum and maximum bounds selected for this experiment to reduce the state space. *scaling_max_freq* is an enumeration, *target_loads* is a threshold, and *timer_rate* and *min_sample* are timers. *timer_rate* is a timer which controls the time period for the governor to make decisions based on the CPU utilisation with a default value of 20,000 microseconds. *target_loads* is the target utilisation threshold for the CPU, higher values will result in a more congested CPU schedule and slower performance, the default value is 99% to reduce power consumption. *scaling_max_freq* sets the maximum CPU frequency the governor can select, in effect capping the CPU frequency, the default value is 2.0 GHz. *min_sample_time* is the minimum timer period at which the governor must spend at the current frequency before making a decision based on sampling utilisation, the default value is 80,000 microseconds.

The proposed approach incorporates both static and dynamic configuration of DPM policy parameters to compare their efficacy for managing user input and resource access-time variability. There is an offline profiling stage which uses static configuration of policy parameters to gather data about the workloads inelasticity with performance

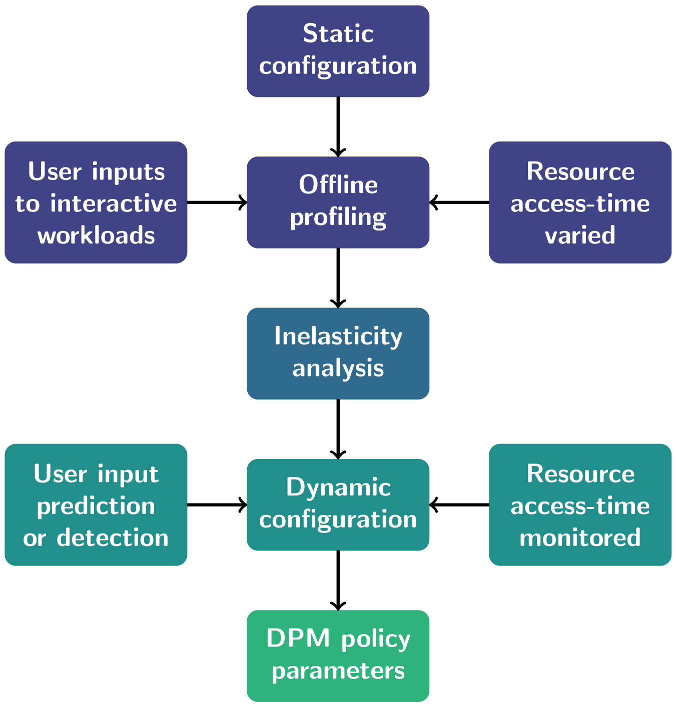

Static configuration
User inputs to interactive workloads
Offline profiling
Resource access-time varied
Inelasticity analysis
User input prediction or detection
Dynamic configuration
Resource access-time monitored
DPM policy parameters

FIGURE 3.2: Proposed relationship between static configuration and dynamic configuration of Dynamic Power Management policy parameters.

under varying user input and resource access-times. The results of this inelasticity analysis are exploited at runtime, either through static configuration again or dynamic configuration. In the case of static configuration, the best parameter configuration on average is selected before execution and used for the duration. For dynamic configuration a sliding scale of runtime variables can be monitored and used to inform parameter configuration. This runtime data can include user input data predicted using a model, user input data detected using interrupts and resource access-time variations such as network conditions as shown in Figure 3.2. As shown in the Figure, offline profiling using static configuration of parameters provides data pertaining to workload inelasticity under varying user input and resource access-times. This data is used to inform dynamic configuration of Dynamic Power Management policy parameters as user inputs and resource access-times are monitored at runtime.

3.1.2 Quality of Experience and energy trade-offs

In order to determine a score for the DPM parameter configurations evaluated in the inelasticity analysis process, the Quality of Service (QoS) metrics between the different input actions must be normalised to derive a comparable QoE. To demonstrate that the proposed approach is applicable to different interpretations of quantitative derivation of QoE, two methods of normalisation are used simultaneously; namely, an exponential decay as proposed by [217] and a linear function for the purposes of a control. A graphical representation of these functions is shown in Figure 3.3 where QoS metrics of web-page

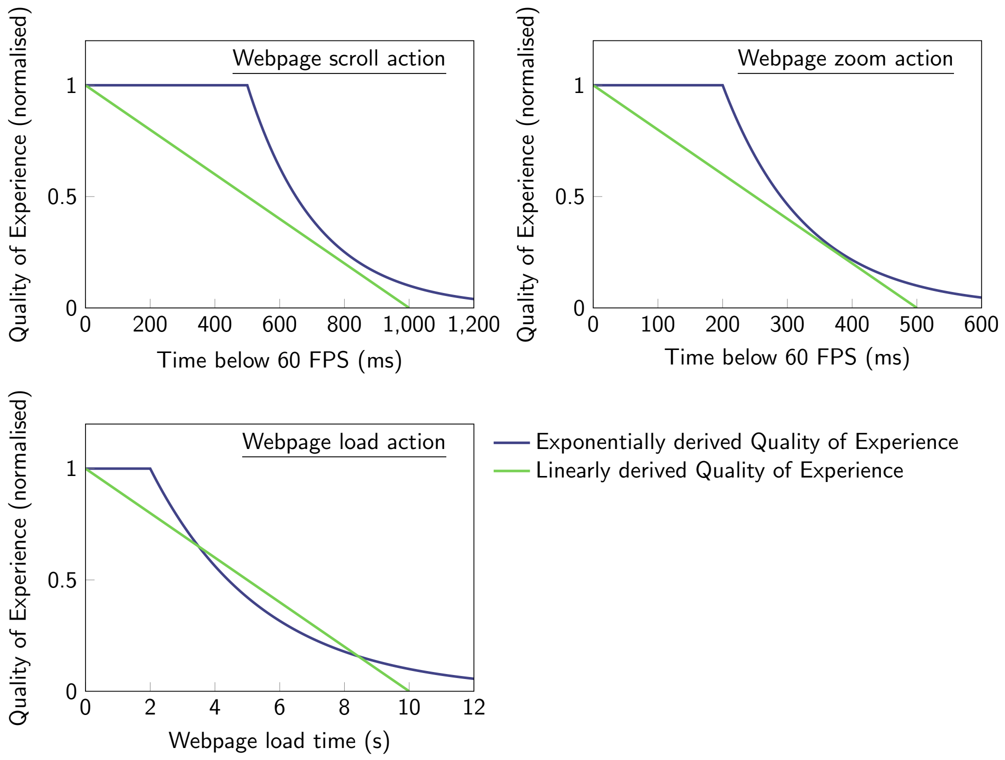


Figure 3.3: Exponential [217] and linear models to derive normalised user Quality of Experience for three user actions in a web browser application; web-page load, scroll and zoom. The exponential model uses a concept of a floor in user satisfaction [208], the linear model is a control to investigate the effects of this approach.

load-time and time spent below 60 frames-per-second are translated to normalized QoE values.

An inelasticity analysis was conducted using the aforementioned user inputs of webpage load, scroll and zoom along with variation over three network conditions of 3G, 4G and WiFi. For webpage load, the network throttling used to emulate each of the network conditions implemented the default parameters suggested by Google Chrome DevTools. 3G, 4G and WiFi connections were emulated by varying the upload bandwidth, download bandwidth and Round-Trip Time. On-device caching was disabled also using Google Chrome DevTools to force reloading of network resources, there is also the potential for caching by Internet Service Providers in the network, this is typically used by Content Delivery Networks for image and video content and cannot be disabled.

In total 50 different DPM parameter configurations for the four parameters in Table 3.1 to be used for the inelasticity analysis. These included the default parameter configuration and 49 randomly generated alternatives. Random parameter search was implemented rather than an alternative method such as block search due to sample efficiency required for the sample size of 50, in Chapter 5 a more advanced search techniques is implemented. The random generation was implemented by uniformly selecting across the available range of each parameter and mixing these uniform selections. Each DPM

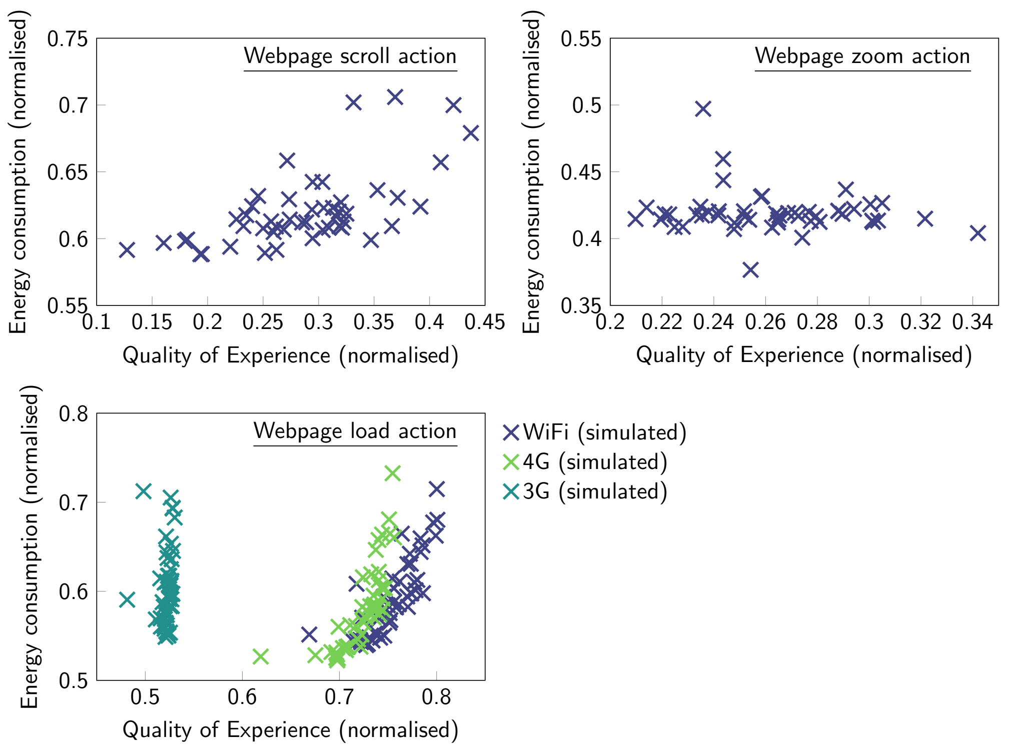


Figure 3.4: Results from the offline profiling across 50 configurations of Dynamic Power Management policy parameters. User Quality of Experience and energy consumption are compared for three web-browser actions.

tuning configuration was profiled offline for all the user inputs and network conditions over 50 repeats and averaged to obtain mean values for QoS and energy. The QoE normalisation functions were then applied, together with an energy normalisation with respect to the highest energy recorded. The resulting trade-offs between QoE and energy are shown in Figure 3.4.

Variation in QoE and energy metrics are observable for all DPM parameter configurations for all user inputs. It should be noted, however, that one parameter configuration is a distant outlier and hasn't been plotted, this is because a highly inefficient tuning for energy conservation was randomly generated. The user inputs which trigger interactive workloads lasting longer timer periods, such as webpage loading and scrolling, show larger variation than the shorter zoom user input. This result is not surprising as there is a fixed overhead for waking up and then sleeping a CPU core.

The effect of resource access-time variability on the QoE of the load input is evident, when the network is relatively unthrottled in the WiFi scenario, CPU performance is clearly driving a variation in QoE. However, when the network is throttled in the 3G case, extra CPU energy is consumed by some parameter configurations for no change in QoE, this energy is consumed by the CPU due to higher than required performance being provided when the resources required are the critical path. The 4G case is a combination of the two, there still exists a trade-off between QoE and energy but there is a steeper gradient than the WiFi case.

Once the inelasticity analysis is complete, these profiles may be exploited at runtime. For this process, the DPM parameter selection will establish a current probability $P(A)$ for the next input being each of the possible choices $A_{0..k}$ and estimate the access times for each resource $R_{0..m}$. Given a pool of n DPM parameter configurations $C_{0..n}$, each configuration will be given a score for QoE and energy using equations 3.1 and 3.2 respectively. In these equations, A_{QoE} refers to the QoE resulting from an action given the resource access-times $R_{0..m}$ and parameter configuration C. A_{energy} refers to the energy consumption resulting from an action given the same conditions.

$$\forall C : C_{QoE} = \sum_{i=0}^{k} P(A_i)(1 - A_{QoE}(R_{0..m}, C)) \tag{3.1}$$

$$\forall C : C_{energy} = \sum_{i=0}^{k} P(A_i)A_{energy}(R_{0..m}, C) \tag{3.2}$$

QoE and energy must be weighted at the system level to determine a trade-off denoted by w_{QoE} and w_{energy}. A composite score of QoE and energy is then calculated for each DPM tuning configuration using equation 3.3. Finally, the minimum score is determined to determine the parameter configuration that will be chosen for the next user input using equation 3.4.

$$\forall C : C_{score} = w_{QoE}C_{QoE} + w_{energy}C_{energy} \tag{3.3}$$

$$C_{choice} = min(C_{score}) \tag{3.4}$$

The inelasticity analysis results highlight ample opportunities to capture energy savings or QoE improvements for all the user inputs profiled. To validate how the proposed approach would exploit these opportunities online, we use a Monte Carlo simulation to generate random interactive workloads and network resource access time conditions. The simulation evaluates the optimisation equation 3.3 for six variants of the proposed approach including static configuration and five dynamic configuration variants. The results are compared considering mean QoE and energy consumption to the default DPM parameter configuration.

The simulation was executed for 100,000 iterations to observe convergence and repeated for both linear and exponential methods of calculating the normalized QoE result. The weightings for QoE and energy, w_{QoE} and w_{energy}, were swept from {0, 1} to {1, 0} to expose the full trade-off curve between QoE and energy dependent on system goals.

The mean QoE and energy consumption values for the variants of the proposed approach under the generated interactive workloads are shown in Figure 3.5. Quality of Experience and energy consumption are compared for QoE ($w_{QoE} = 1$, $w_{energy} = 0$), Energy ($w_{QoE} = 0$, $w_{energy} = 1$) and QoE & energy ($w_{QoE} = 0.5$, $w_{energy} = 0.5$). The largest improvements are demonstrated when both QoE & energy are considered during optimisation. The "QoE" results represent when $w_{QoE} = 1$, such that there is a full weighting towards QoE improvement. The "Energy" results represent when $w_{energy} = 1$, such that there is a full weighting towards energy improvement. The "QoE & Energy" results represent when $w_{QoE} = 0.5$ and $w_{energy} = 0.5$ or an equal weighting between QoE and energy.

The *Default* approach is the default configuration for the DPM parameters. The *Static* approach considers usage of the inelasticity profiles with equal probability of each user input and network condition occurring. The *Dynamic-Network* approach adds accurate detection of current network resource access time. The *Dynamic-Predict* and *Dynamic-Detect* approaches emulate accurate user input prediction and detection respectively. The *Dynamic-Predict+Network* and *Dynamic-Detect+Network* combine accurate detection of current network resource access time with user input prediction and detection respectively.

Under most operating conditions, a suitable trade-off between energy and QoE is more desirable than fully weighting one or the other as shown in Figure 3.5. Instead there is a full range of Pareto-optimal points which are shown in Figure 3.6 for both methods of normalisation of the QoE metric. There is minimal qualitative difference between the two methods of QoE normalisation, this is because the QoS floor used in the exponential derivation doesn't overlap substantially with the data points so has minimal effect. Selection of the parameters used in the QoS floor is individual to the each user's tolerance of performance and each application, setting these parameters should be done through user studies [31].

All variants of the proposed approach demonstrate energy savings and QoE improvements, when compared to the default DPM parameter configuration. The resource access-time detection used by the *Dynamic-Network* variant achieves marginal improvement over the *Static* variant that models current network conditions with equal probability. Prediction of user input is shown to be more effective by the *Dynamic-Predict* and *Dynamic-Predict+Network* variants. However, detection of user input is the clear optimal choice as shown by the *Dynamic-Detect* and *Dynamic-Detect+Network* variants which achieved QoE improvements of 27% or energy savings of 13%, in comparison to the default DPM parameter configuration. In this experiment, the overhead cost of prediction and detection at runtime could not be measured as it is based on a Monte Carlo

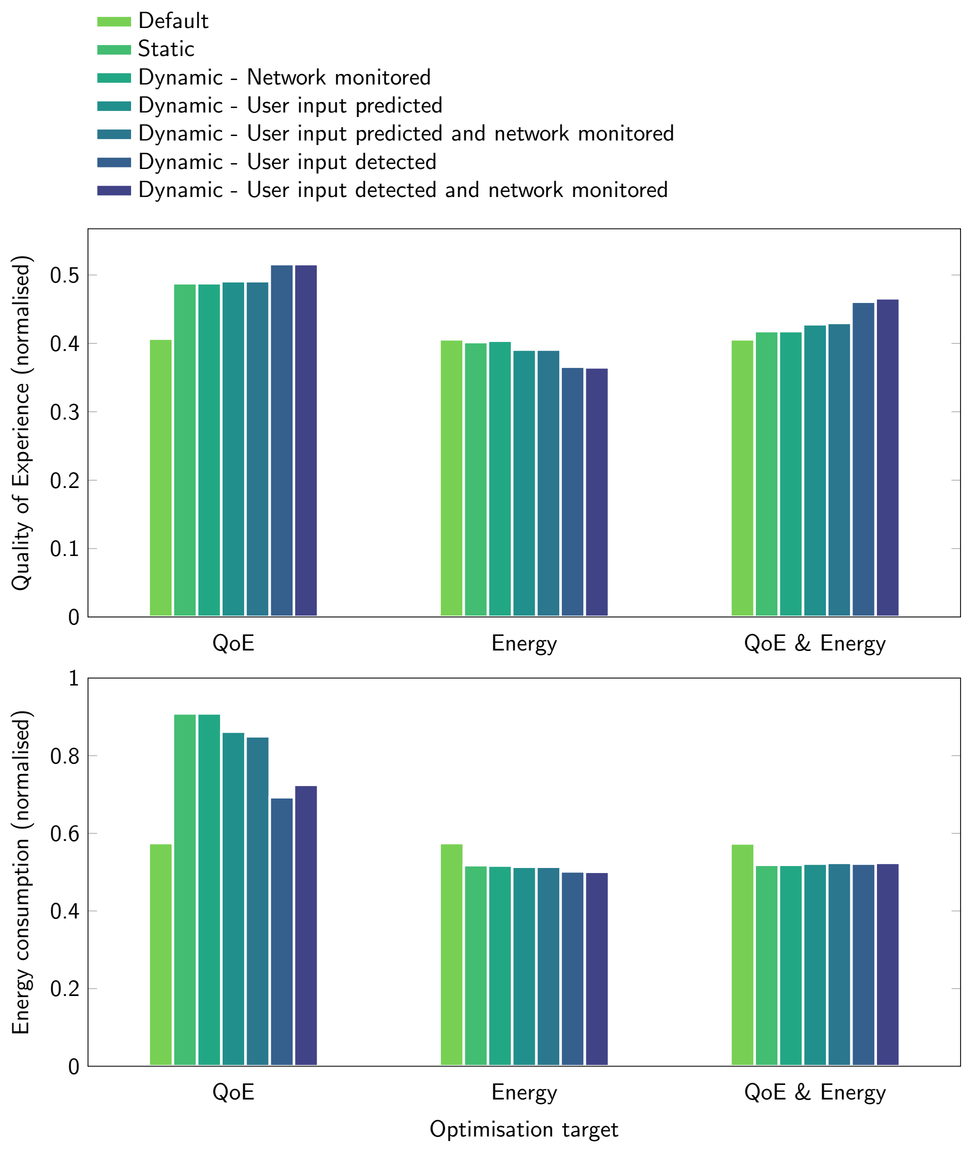


Figure 3.5: A comparison between static Dynamic Power Management policy parameter configuration, various dynamic configuration approaches, and the default parameter configuration.

simulation, the overhead in time and energy associated with these will reduce the gains possible by dynamic configuration over static configuration. In the experiments in the next section, the overheads of running a dynamic configuration approach are measured.

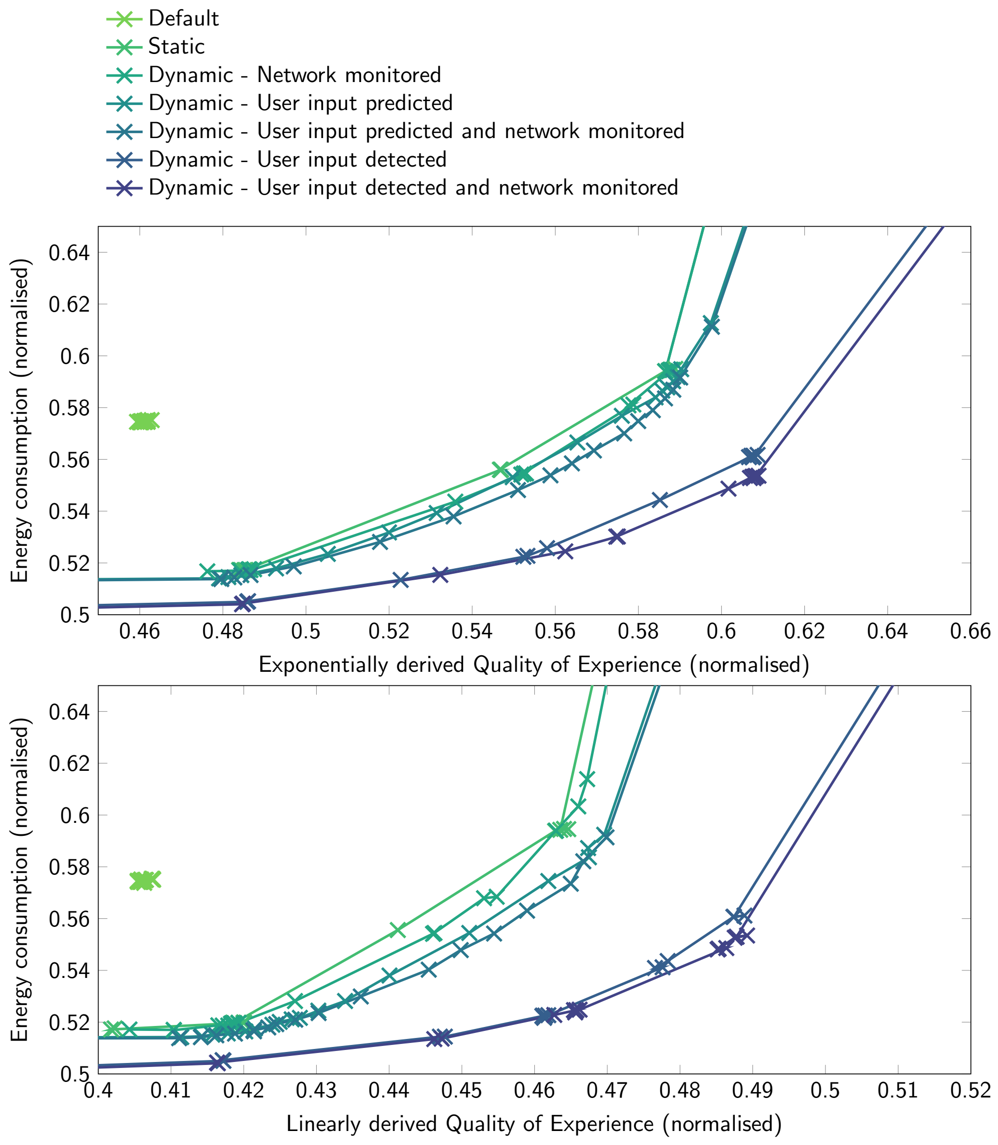


Figure 3.6: A comparison between the Pareto-optimal curves for the proposed approach using both exponentially and linearly derived Quality of Experience metrics.

3.2 Dynamic configuration for unseen dynamic workloads

There are two primary flaws in the approach described in Section 3.1 that may hinder its deployment in some scenarios. Firstly, it requires knowledge of the application, user inputs and relevant resource-access times ahead of execution for the offline profiling stage. Second, even if the offline profiling stage is logistically possible, it is extremely time consuming and the state-space explosion of adding further DPM policies and parameters would only increase this. In this section, an alternative architecture is proposed for dynamic configuration of DPM policy parameters for unseen workloads without profiling. Synthetic benchmarks are used along with the Linux Operating System rather than Android so the relative energy and performance improvements should be interpreted with scepticism as discussed in Sections 2.3.3 and 1.3.1. The synthetic benchmarks are used to obtain nominal performance results from a statically configured governor (Section 3.2.1. These results are then compared with the proposed dynamic configuration architecture and other DVFS governors (3.2.2).

3.2.1 Management of unseen dynamic workloads

The hardware platform used in the previous experiment, the ODROID-XU3 development board, was also re-purposed for this experiment. Instead of Android, a more conventional computer architecture research scenario was used, the Linux Operating System, whilst there is the negative of less representative results, the additional control over the system widens the possible scope of experimental results. The SoC on the ODROID-XU3 board is the Samsung Exynos-5422 which uses an ARM big.LITTLE architecture with 4 A15 and 4 A7 cores. For this experiment the applications were tied to the A15 cores using thread affinities as mapping policies for older big.LITTLE systems are less effective than recent editions such as Energy Aware Scheduling [51].

MP4 video-decoding applications were chosen to create the synthetic benchmarks due to their flexibility in computational demands due to variable frame rates and video codecs. In Figure 3.7 there are execution manifests for the five synthetic benchmarks used. The applications are started at fixed times during manifest providing *Step*, *Ramp* and *Dynamic* workload patterns. For the Step workload the four MP4 video-decoding applications are initiated at the same instance in time in a step response from minimal computation activity to near maximum. For DPM policies reliant on polling architectures such as the default DVFS governor in Linux this can be problematic as a response will not occur until the next polling interval. The Ramp workload represents a slower ramp up and down in computation and the Dynamic-1, Dynamic-2 and Dynamic-3 are procedurally generated pseudorandom sequences representing less predictable workloads.

For the purpose of establishing the computational intensity of each dynamic workload benchmark the clock frequency of each A15 CPU core on the ODROID-XU3 was manually pinned to the maximum 2.0GHz using the *performance* DVFS governor. Figure 3.8 shows the CPU utilisation measured whilst executing the five synthetic benchmarks from Figure 3.7. The Step workload utilisation shows an initial spike in computational demand the core utilisation then reverts to near 50% with small temporal variation that may be caused by conflicting activity between cores or difference in time to decode individual frames. The Ramp workload exhibits more gradual increases in computation with corresponding smaller spikes in core utilisation and the Dynamic workloads do not follow set patterns.

Static configuration of the Linux governors has been demonstrated to offer favourable power and throughput trade-offs for known workloads [47]. This tuning process is often ignored for general-purpose computing as the workloads are not known and as offline

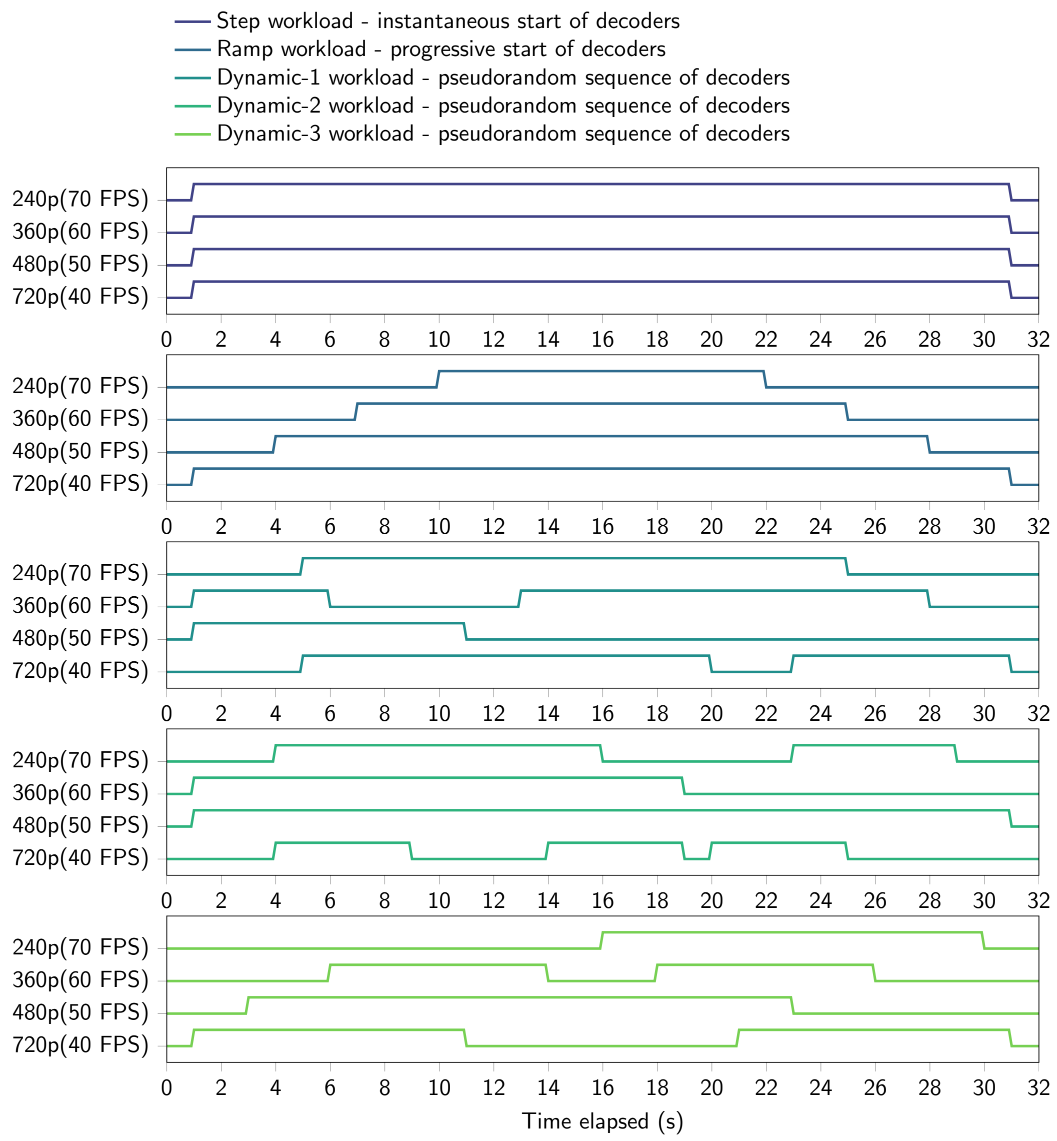


FIGURE 3.7: Execution manifests for five synthetic benchmarks consisting of four concurrent MP4 video-decoding applications.

profiling violates the objective of the system. This experiment aims to offer a unification of runtime application throughput awareness with an existing DVFS governor to dynamically configure the governor. To understand the potential throughput and power trade-offs this could unlock for the five workloads in this experiment, the *conservative* governor was profiled offline for different up- and down-utilisation threshold parameters with these five workloads and the results plotted in Figure 3.9. Two parameters of the governor were varied, the up- and down-utilization thresholds, to obtain a number of operating points trading-off performance and power. The results are normalised to the *performance* governor so that the five workloads may be plotted on the same scale. The first number in the legend is the up-utilisation threshold which refers to the CPU utilisation at which the CPU frequency will be increased, the second number is the down utilisation threshold with the opposite result.

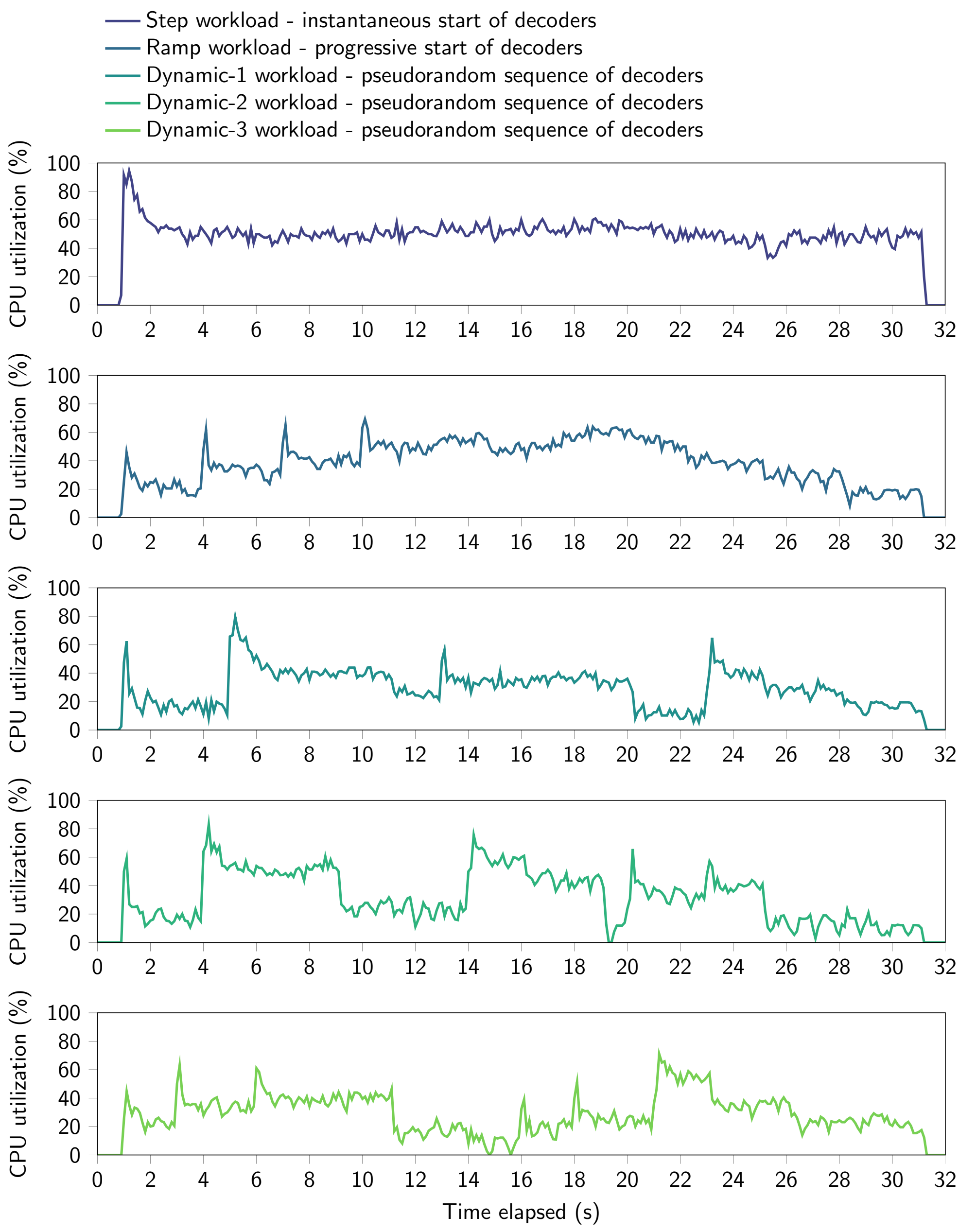


Figure 3.8: CPU utilisation traces for the five synthetic benchmarks recorded using the four ARM A15 cores that form the big CPU cluster on the ODROID-XU3 development board.

The default tuning of the *conservative* governor in the distribution of Ubuntu 14.04 used for experimental purposes has up and down utilisation thresholds set at 80/20, the results show that this naive tuning offers poor power savings compared to more optimal thresholds which offer increased power savings for little degradation in application throughput. These results replicate the findings of a previous study based on offline profiling that used benchmark suite applications [47].

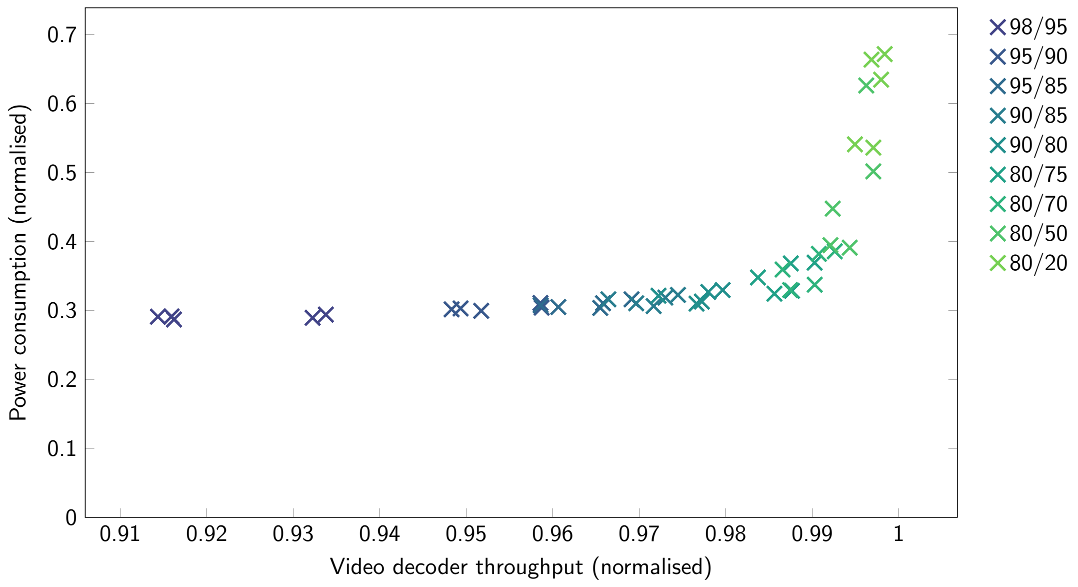


FIGURE 3.9: Power consumption and application throughput measured for the five synthetic benchmarks in Figure 3.8, using the *conservative* DVFS governor.

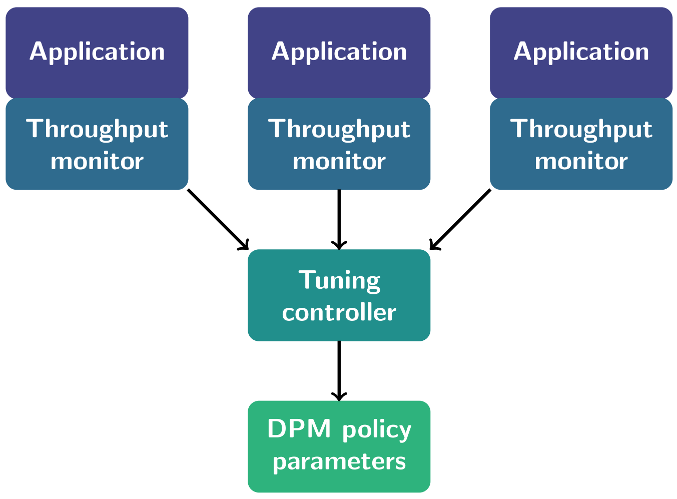


FIGURE 3.10: Simplified block diagram representing the architecture of the dynamically configured Dynamic Power Management policy.

3.2.2 Architecture for dynamic configuration

Applications report throughput violations to the tuning controller, consecutive violations are required before an escalation to reduce communication overhead.

The structure of the dynamic configuration architecture is shown in Figure 3.10. The control flow is started with the tuning controller reducing the performance margin of the system until one of the executing applications cannot meet throughput requirements. If sufficient throughput violations are raised an escalation is sent to the tuning controller to reduce the threshold parameters to increase performance until the escalated application sends a de-escalation interrupt. After de-escalation, the process resets and the tuning controller continues to squeeze the performance margin.

Conversion of a failure to meet application throughput requirements into an escalation is handled by a filter to reduce communication overhead. If an application fails to meet its throughput requirements for 75% of the preceding frames, an escalation is made. De-escalations occur when all the preceding frame history has met throughput requirements.

The tuning controller operates on the same 100,000μs sampling period as the *conservative* governor it is tuning. If the throughput controller is demanding an increase in performance, the thresholds are reduced by 5% every sampling period, whereas if the throughput controller is not demanding an increase, the thresholds are increased by 1% every sampling period. The 1% threshold increase every period in this approach allows the tuning controller to reduce the performance margin of the system until throughput violations occur, the magnitude and frequency of this increase also becomes a tunable parameter.

If the threshold reduction of 5% is used, it can be determined that if there is a change in the dynamic workload, the system may move from minimum performance to maximum performance in the same worst case 2 second period achieved by Linux. The implementation of the independent governor process acting concurrently to the tuning controller ensures this as performance is ramped up regardless of the thresholds until the system reaches a lower utilisation, at which point the thresholds are already very low such that the *conservative* governor will continue ramping up performance.

A visualisation of the auto-tuning of the governor is presented in Fig. 3.11, the first plot is the reference workload utilisation using the *performance* governor at 2.0GHz CPU frequency, the second and third plots show the frequency being changed over time in the *conservative* governor as the utilisation changes. The fourth plot show the governor thresholds that are being tuned by the tuning controller, a saw-tooth pattern is formed as the performance margin is squeezed. Importantly, performance escalation happens at a faster rate to allow interactivity, but because the system is such low overhead, the application and throughput controller can interrupt sufficiently promptly when escalation is complete, that the governor threshold does not drop to zero and forego all power savings. If performance squeezing also happened at a similar rate as escalation the system would oscillate and be less effective, it is key that the escalation gain should be higher than the de-escalation gain.

Comparison was made between the proposed dynamic configuration architecture and other approaches over the five synthetic benchmarks, results are normalised to the *performance* governor. The results in Figure 3.12 show that the dynamic configuration and two static configurations obtained from the offline profiling all are more energy sufficient than the default governors {*ondemand*, *interactive*, *conservative*} at the expense of a small degree of performance. Most importantly the dynamic configuration approach is competitive with the static configurations that used offline profiling demonstrating that it is possible to do so in scenarios where offline profiling is not feasible.

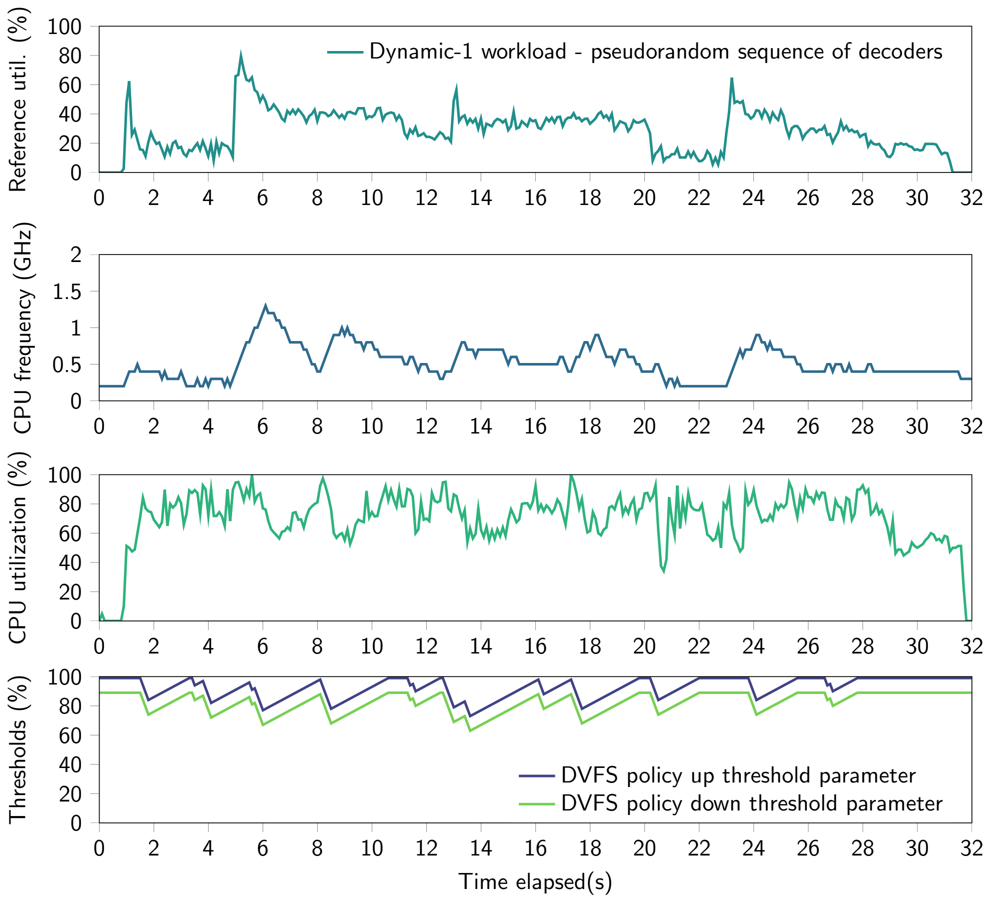


FIGURE 3.11: Visualisation of an execution trace for the *Dynamic-1* benchmark using the proposed dynamically configured governor.

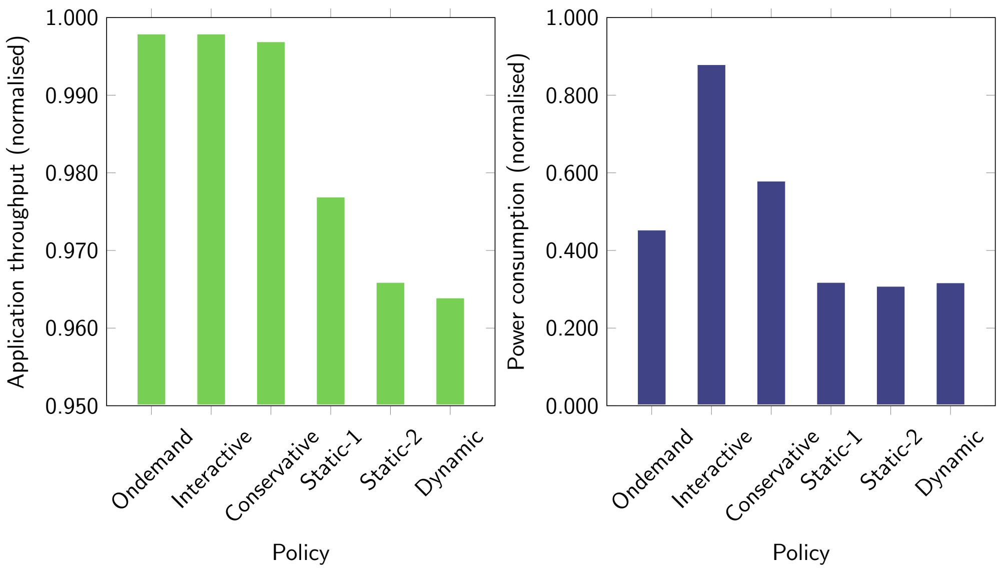


FIGURE 3.12: A comparison of application throughput and power consumption between the default governors {*ondemand*, *interactive*, *conservative*}, two static configurations of the *conservative* governor and the proposed dynamically configured *conservative* governor.

3.2.3 Summary

DPM policy parameters are configured manually on a regular basis in computer systems including mobile devices, yet until now there has been limited prior research in effective ways to choose parameter values. This chapter has detailed empirical evidence from experimentation using mobile System-on-Chips that static and dynamic configuration of DPM policy parameters offers improvements in performance, energy or a combination of the two. Variability in user inputs and resource access-times has also been exploited to profile interactive workloads inelasticity with CPU performance. There remain challenges including the requirement for data from real mobile devices scaling offline profiling to multiple DPM policies with additional parameters, both of which will be addressed in the remaining chapters of this thesis.

Chapter 4

Mitigating Performance Degradation from Thermal Throttling

Mobile devices are limited in mass and volume reducing the viability of active device cooling implementations, this requires the use of less effective passive techniques to maintain device skin temperature levels (Section 2.1.1). Application performance demands on a modern mobile device are driven by sustained performance workloads, such as mobile gaming, Virtual and Augmented Reality (Section 2.1.2). Mobile System-on-Chips have corresponding increases in performance through both architectural changes and frequency of operation increases; which has resulted in the peak power consumption exceeding the sustainable thermal envelope defined by device skin temperature requirements (Section 2.1.3). In this chapter, existing thermal throttling techniques are characterised to show that they mitigate this challenge by capping the frequency of operation of the System-on-Chip (Section 4.1). A mobile device thermal management policy using Frequency Capping is demonstrated by executing a 3D video game on a smartphone (Section 4.1.1) and the effects of Frequency Capping on frame rate janks in interactive workloads is evaluated (Section 4.1.2). A new DPM lever is proposed to swap out with Frequency Capping in existing thermal throttling systems to mitigate some of the interactive performance degradation (Section 4.2). This lever, Task Utilisation Scaling (TUS) (Section 4.2.1) allows a higher maximum CPU frequency than Frequency Capping at the same power consumption (Section 4.2.2).

4.1 Device Skin Temperature Throttling

Passive cooling techniques used in mobile devices, such as heat spreaders and pipes have so far failed to replicate the dissipation characteristics of their active counterparts

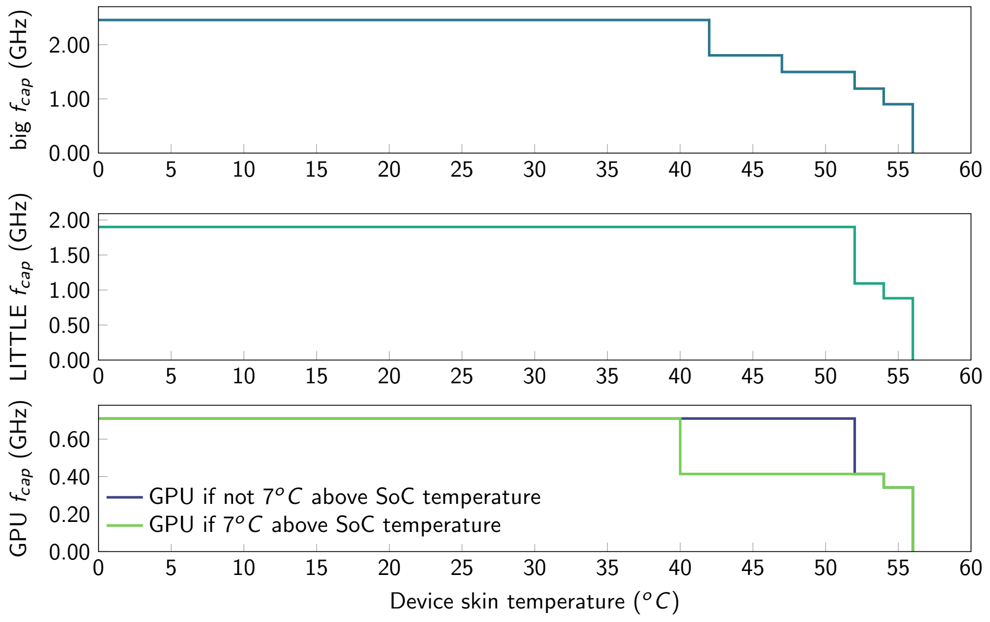


FIGURE 4.1: Thermal throttling rules implemented by the Qualcomm Thermal Engine used in the Google Pixel 2 (obtained from /vendor/etc/thermal-engine.conf).

(Section 2.1.3). The dissipation characteristics define the thermal envelope, which can be sustained without the device skin temperature exceeding a level that will cause discomfort or injury to the user; this temperature level is around 45^oC [93]. A typical sustainable thermal envelope for a smartphone is approximately 3.5W [31], although environmental conditions, including how the device is held by the user, can affect this number [93]. In this section, the thermal throttling policy of a modern mobile device, the Google Pixel 2, is demonstrated using a demanding 3D video game (Section 4.1.1). The effect of the Frequency Capping (FC) implemented by this policy is evaluated by subjecting interactive mobile workloads to the same FC implementation and measuring the performance degradation (4.1.2).

4.1.1 Thermal management policy

Thermal throttling policies typically implement limits the maximum frequency at which an SoC may operate using a DPM lever called Frequency Capping (FC). The relationship between voltage, frequency and dynamic power dictates that higher frequencies are often less power-efficient than lower frequencies (Section 2.2). Experimentation was carried out using the Google Pixel 2 smartphone, running the Android 9.0 Pie operating system along with the Energy Aware Scheduling patchset in the underlying Linux kernel. Thermal throttling in the Google Pixel 2 smartphone is implemented using the Qualcomm Thermal Engine, a programmable cross-platform solution. The input to the controller is a configuration file specifying frequency caps, f_{cap}s, that may be implemented by a PID controller according to temperature sensor measurements.

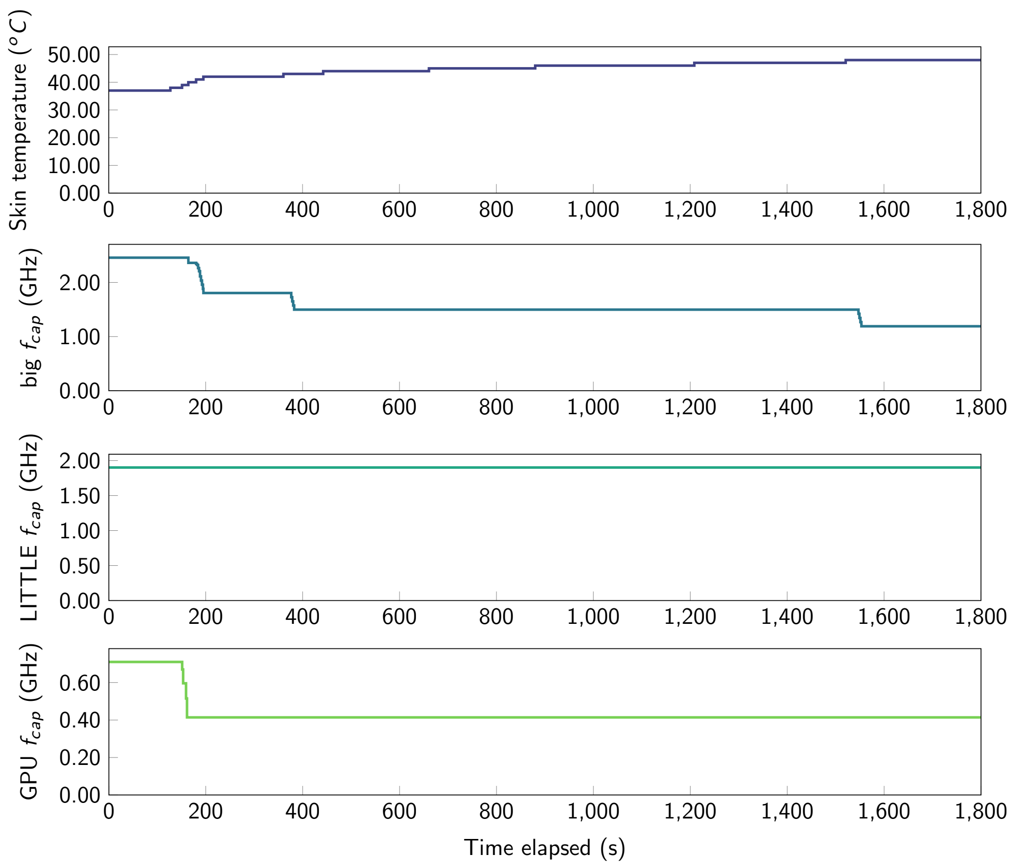


FIGURE 4.2: An execution sequence of 3D video game Player Unknown Battlegrounds demonstrating the rules from Figure 4.1 being implemented by the Qualcomm Thermal Engine.

The Google Pixel 2 implementation shown in Figure 4.1, estimates skin temperature using a designated thermistor. At 40^oC, throttling is initiated; by 47^oC, the big CPU cluster is capped below half its maximum frequency; and at 56^oC a device shutdown is enforced. In addition, the GPU frequency is capped dependent on the temperature difference measured between the GPU subsystem and the overall SoC package.

In Figure 4.2, a 3D video game (Player Unknown Battlegrounds, maximum graphics, resolution and FPS) is executed on the Google Pixel 2 to demonstrate thermal throttling in action. After 151 seconds, throttling is initiated before progressively increasing as the phone temperature rises to 48^oC, which exceeds the discomfort level of 45^oC. The big CPU cluster frequency cap, f_{cap}, is reduced to 1.19GHz from an initial 2.46GHz starting point.

4.1.2 Frequency Capping

Technical constraints limit performance analysis of individual frames for third-party interactive Android NDK applications such as 3D game engines in a production Android

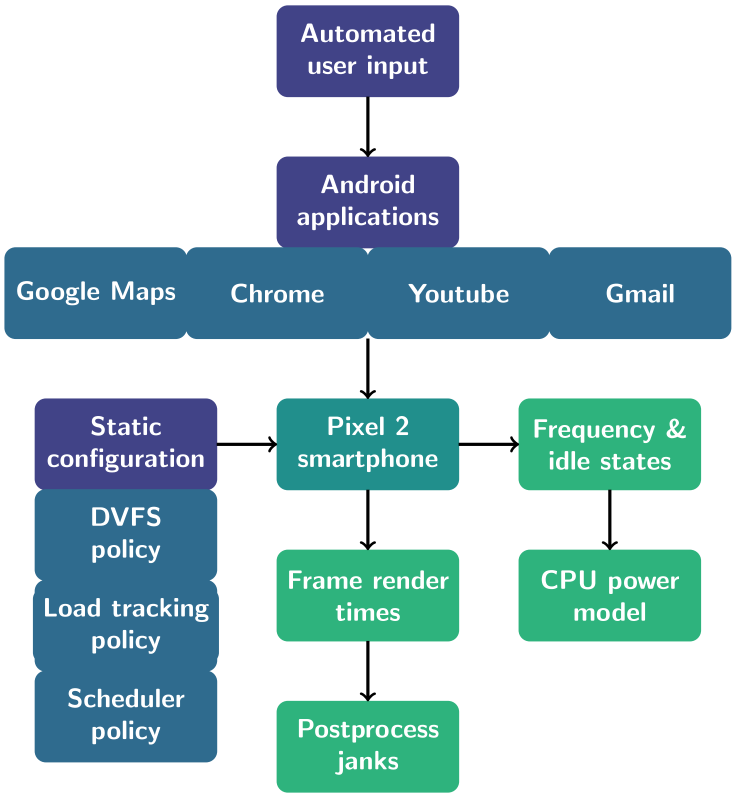


FIGURE 4.3: Experiment setup using the Google Pixel 2 smartphone to measure frame rate janks and model power consumption.

build. However, the effect of FC on interactive Android SDK applications may be analysed under enforced frequency caps. Experimental sequences were implemented for four Android SDK applications: Google Maps, Gmail, YouTube, and Chrome. Each application was executed using Android input automation using the Workload Automation framework [258]. Workload Automation uses Android UI Automator to detect Android UI elements that when combined with a script can emulate user input in a repeatable fashion for benchmarking. Compared to synthetic benchmarking techniques, using real Android applications on a real mobile device is far more representative of real-world usage (Section 2.3.3).

The execution sequences last for approximately one minute each and the frame rate janks and frequency and idle state occupancies are traced to model CPU power consumption. Frame rate janks is a metric dependent on the long-tailed distribution of frame-rendering times (covered further later in this thesis in Section 5.1, this means that the variance is high and many repeats must be carried out for a reliable sample mean or median. Configurations under test have their repeats interleaved to compensate for variance as network conditions vary during the day, a caveat of experimentation with real devices and applications.

FC effects were characterised on the experimental sequences from the four Android SDK applications, Figure 4.4 shows the results. The big CPU cluster under FC is unable to

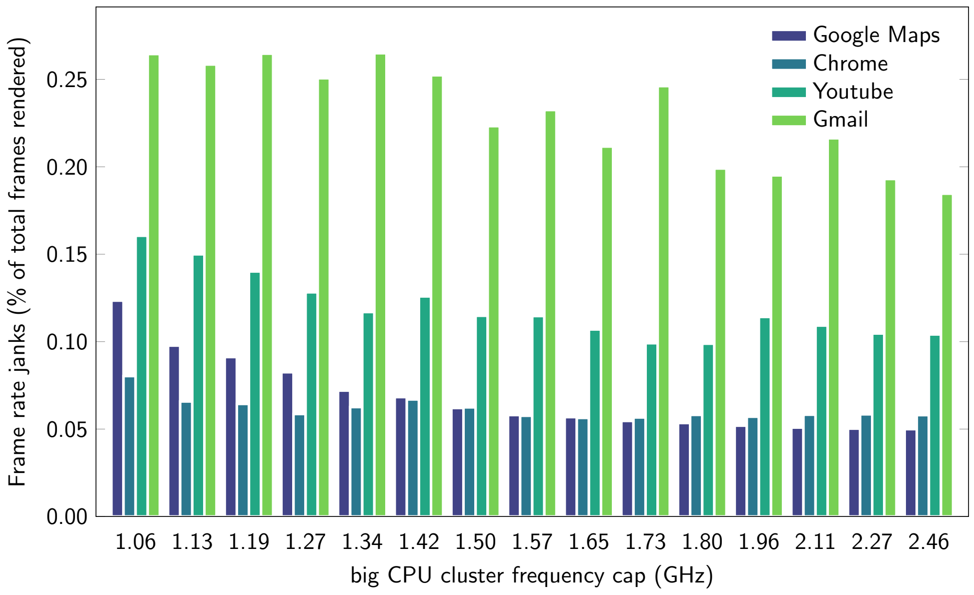


FIGURE 4.4: The effect of frequency capping on frame rate janks over four application sequences.

use frequencies beyond the defined cap, f_{cap}, even if a spike in computational demand requires it, this leads to an increase in frame rate janks. The relative and absolute degradations vary between the applications, with Google Maps exhibiting the highest relative degradation. The variation in frame rate janks for Google Maps from 5.0% of frames to 12.2% represents a 146% increase as the frequency cap, f_{cap}, is reduced.

The Google Maps sequence is highly interactive relative to the other applications, driven by the emulated utilisation involving several click-through, scrolling and zoom events. Gmail does not render many new frames as there is limited scrolling movement within the execution sequence. This means the baseline number of janks, which are largely click-through events, is higher than average but also these click-through events are more likely to miss their deadline irrespective of CPU frequency limits. Chrome and Youtube are less demanding applications as a whole and only the lower CPU frequency caps have a noticeable effect.

Median frame rate jank measurements shown in Figure 4.4 do not in fact paint the full picture. As previously mentioned, evaluation of frame rate janks will be covered further later in this thesis in Section 5.1. Figure 4.5 presents data from the same experiment in more detail, including the quartiles for each f_{cap} and application pair. The experiment would benefit from more repeats as frame rate janks is inherently a high-variance metric but trends are still visible for each application. The most surprising result is for Chrome which benefits from a reduced inter-quartile range from intermediate frequency caps before widening again from lower frequency caps. This result may be due to a reduction in CPU idle state transitions as the frequency selected matches the performance required. Furthermore, this result is also backed up by data from other iterations of the experiment

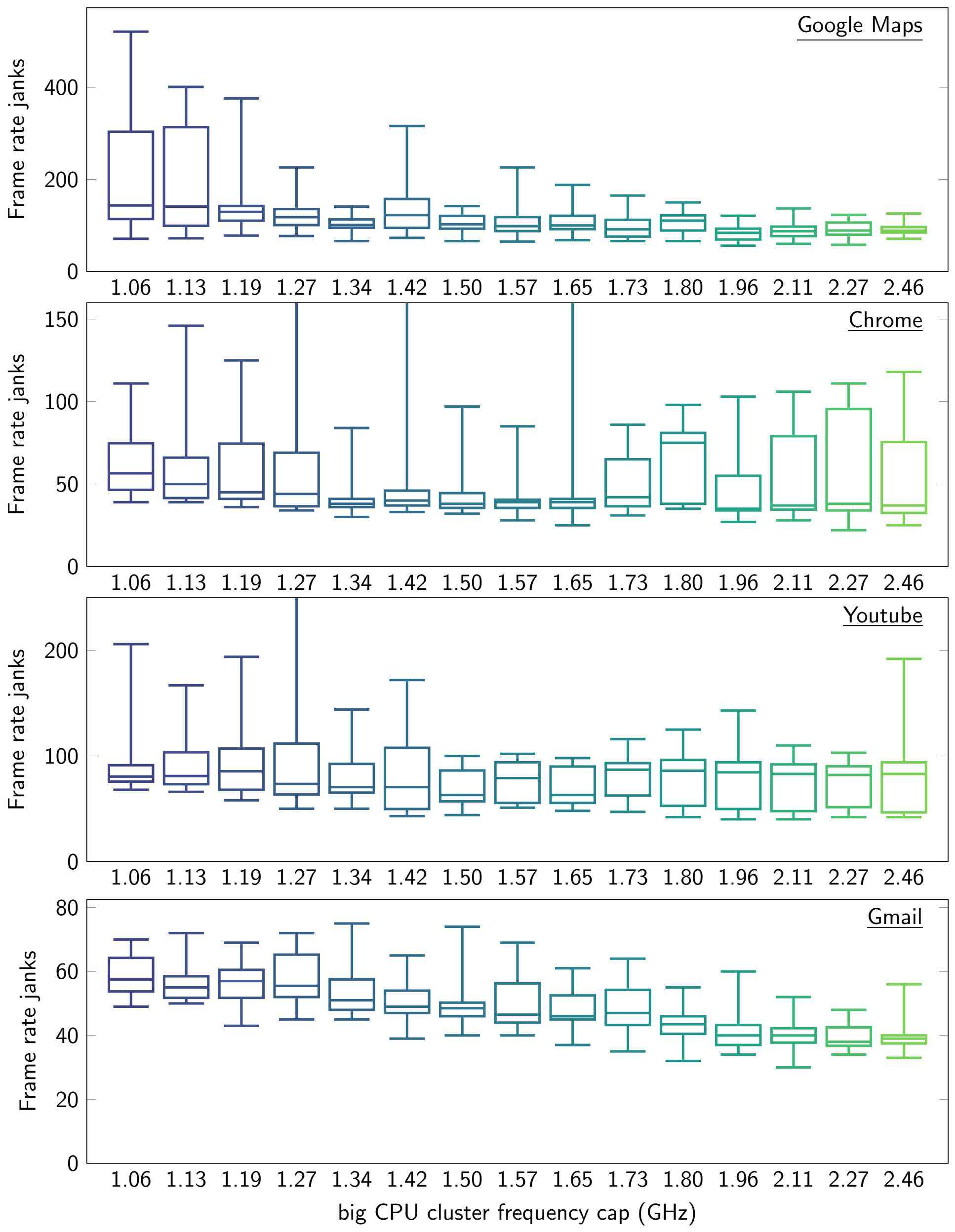


FIGURE 4.5: Comparison of frame rate jank distributions using the same data as Figure 4.4. The quartiles are plotted for each of the four applications at each frequency showing that each application is affected differently by Frequency Capping.

and demonstrates that whilst FC has an overall negative effect on frame rate janks, there are exceptions.

4.2 Swapping the lever for Dynamic Power Management

Frequency Capping (FC) effects were shown in Section 4.1.2 to increase frame rate janks across interactive mobile applications. In this section, a new DPM lever is proposed, Task Utilisation Scaling (TUS), designed to be interchangeable with FC in existing

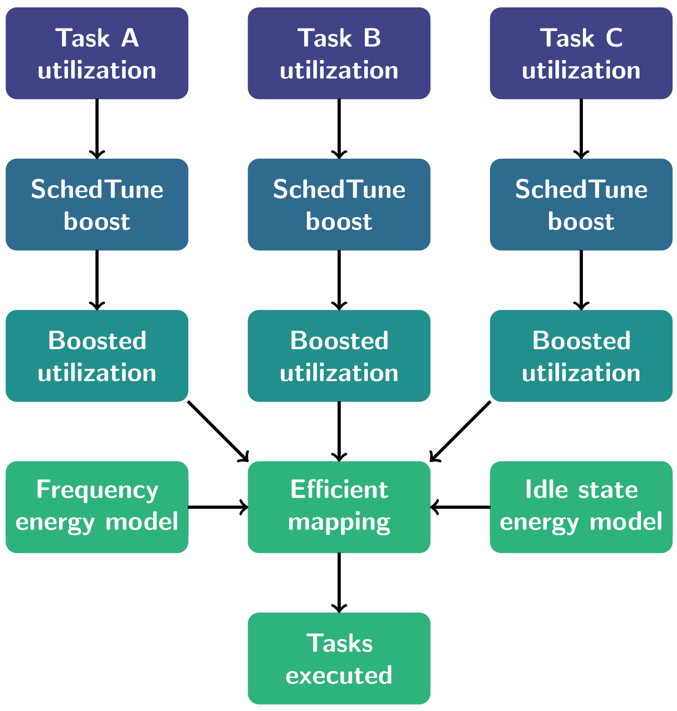


Figure 4.6: Energy Aware Scheduling high-level architecture [51] of a collection of improvements designed to improve energy-efficiency and policy coordination in the Linux Operating System.

thermal throttling systems (Section 4.2.1). TUS is compared to FC and shown to allow higher maximum frequency at the same power under thermal throttling which mitigates some of the negative effects on frame rate janks (Section 4.2.2).

4.2.1 Task Utilisation Scaling

The Google Pixel 2 smartphone uses the Energy Aware Scheduling (EAS) patchset in the underlying Linux kernel [51]. EAS is a recent enhancement to the Linux kernel which seeks to execute tasks in an energy-efficient fashion, a high-level architecture is shown in Figure 4.6. A CPU frequency and idle state energy model is profiled offline to obtain nominal performance and power trade-off values for executing workloads on each CPU core. At runtime when a new task enters the system, the available CPU cores are compared to determine the most efficient core with sufficient capacity to execute the task. Additionally, there is a periodic load balancing that takes place to re-map tasks that are sustained over a longer period. Initially, when EAS was applied to interactive workloads there were performance issues as the performance margin (Section 3.2.1) was too small to allow for variations in computational demand. SchedTune.boost was implemented to mitigate these issues; it is a multiplier applied to foreground User Interface applications to artificially inflate their reported task utilisations to increase the performance margin.

A characteristic common to the interactive workloads used in the experiment Section 4.1.2 is that high performance is required for short sporadic periods of time. This may

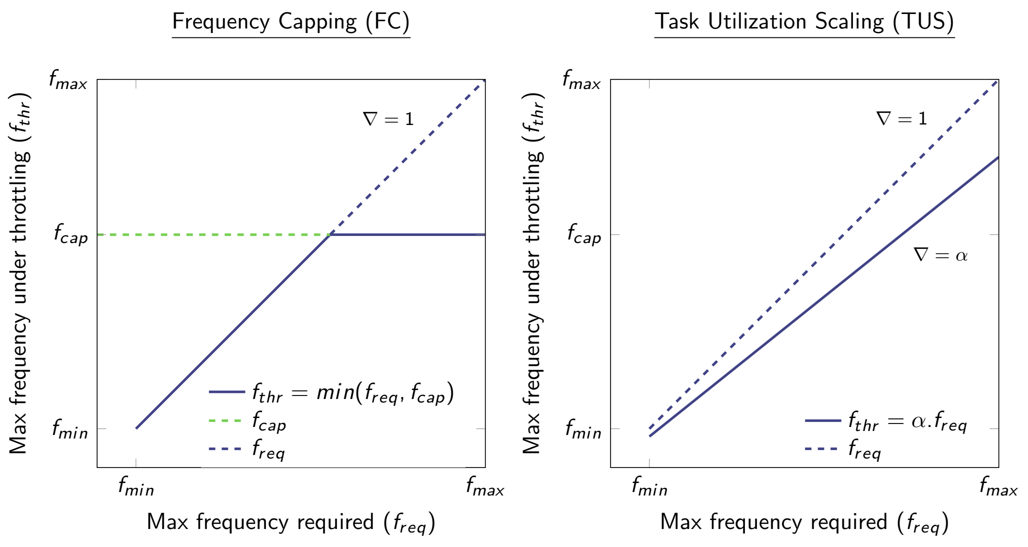


FIGURE 4.7: Effect of frequency capping and task utilisation scaling levers on the maximum single core frequency available after throttling, f_{thr}, selected by the Energy Aware Scheduling model.

be a response to user input or the conclusion of a delay waiting for external resources such as network access. For the remainder of time, the performance required is lower and less likely to be at a level that would be affected by a frequency cap. In effect, when an interactive application is executed under a frequency cap, the brief critical events are throttled and the sustained non-critical periods (where throttling would be less noticeable) are not throttled.

An approach to thermal throttling, that considers interactive performance, (including frame rate janks), as well as average frame rate, must throttle the system during non-critical periods as well as critical periods. The proposed DPM lever Task Utilisation Scaling (TUS) is designed to achieve this purpose while serving as a drop-in replacement for frequency capping in existing thermal throttling systems. The SchedTune.boost functionality may be repurposed for negative boosts to artificially deflate the reported task utilisations. TUS uses a Task Utilisation Scaling factor, α, for this purpose as shown in Figure 4.7. By throttling all periods of execution equally using a scaling factor, α, task utilisation scaling permits a higher maximum f_{thr} than the equivalent frequency cap, f_{cap}, allows. An α of less than 1.0 artificially deflates the utilisation of all tasks such that the tasks are mapped to cores by EAS with a smaller or negative performance margin. Other modifications to EAS are required to remove over-utilisation thresholds and limit schedutil's ability to raise performance when throttled. There is no additional computation requirement to calculate TUS compared to FC in Android as it is already used by EAS, in systems not deploying EAS there may be a minimal associated overhead with this.

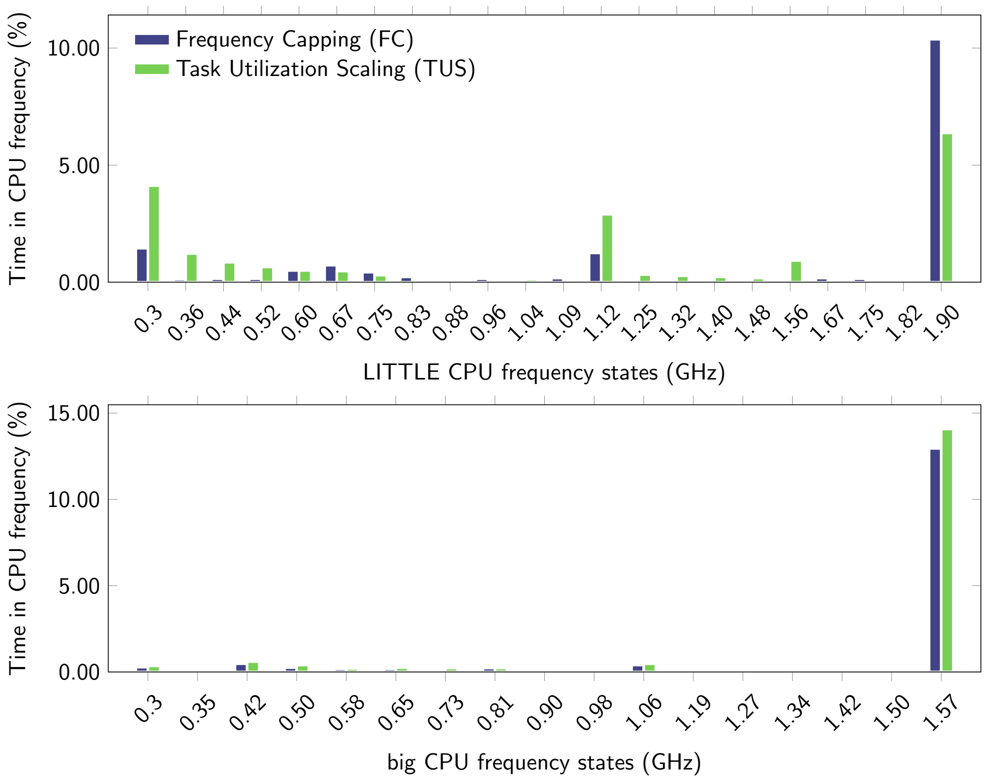


FIGURE 4.8: Frequency occupancy compared for an execution of Gmail with a frequency cap of 1.57GHz on the big CPU cluster. For Task Utilisation Scaling this is implemented by setting $\alpha = 0.63$.

4.2.2 Comparison between levers

In theory by throttling the performance of non-critical periods as well as critical periods of execution, TUS will be able to reallocate the power saved to increase maximum frequency during critical periods. In this section, FC and TUS are compared to understand these differences in practice.

In Figure 4.8, the CPU frequency occupancies under FC and TUS are compared for an execution of the Gmail application sequence. FC is configured to enforce a frequency cap, f_{cap}, of 1.57GHz on the big CPU core. The TUS factor, α, is set to 0.63 which also allows a maximum frequency of 1.57GHz on the big CPU core but will also throttle performance lower than this. The data shows that execution on the big CPU core is largely unchanged as the critical periods are throttled equally by both FC and TUS; however, the data from the LITTLE core shows that far more time is spent in slower CPU frequency states under TUS than under FC. TUS throttles background tasks into lower performance states than FC, in particular, the lowest frequencies of the LITTLE CPU cluster. This saves power that may optionally be reinvested into executing higher priority tasks with more performance.

The effect on frame rate janks at all frequencies plotted in Figure 4.9 confirms the finding suggested by the frequency occupancy data, that the critical periods are throttled equally

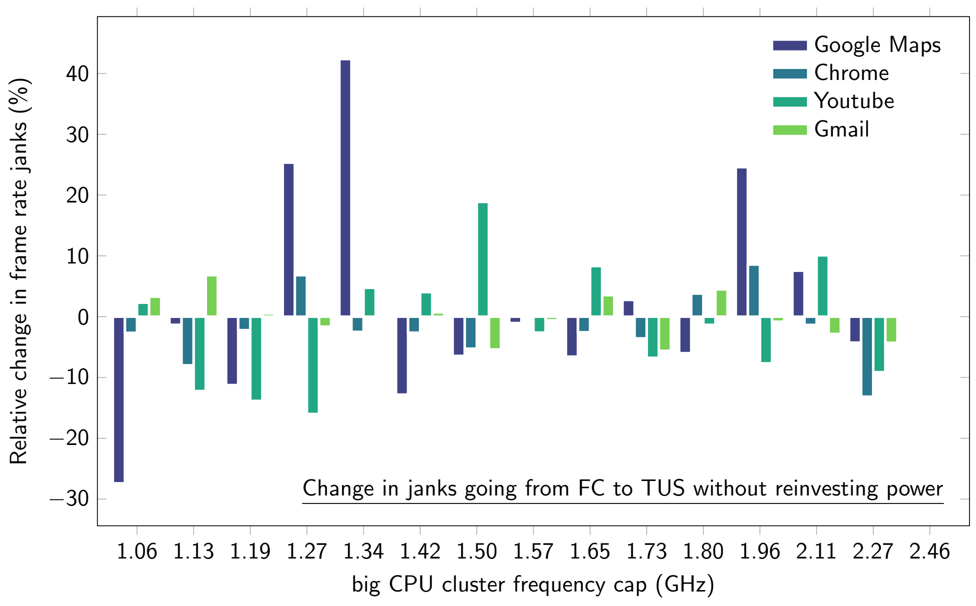


FIGURE 4.9: Change in frame rate janks from swapping to Task Utilisation Scaling from Frequency Capping relative to the value using Frequency Capping.

by both FC and TUS. The measurement for frame rate janks is noisy and insufficient repeats are possible to reduce mitigate this fully (Section 5.1 has more detail and a proposed solution). It is a challenge to draw conclusions comparing the individual applications given this noise, but the application that shows the most variance, Google Maps, was also shown in Figure 4.5, to have the most variance in frame rate janks under FC. There is no difference in frame-rate janks between FC and TUS in Figure 4.9 when averaged across all frequencies, there is not sufficient data to prove if there any underlying trends associated with individual frequencies due to the measurement challenge discussed.

The data in Figure 4.9 can be averaged across all frequencies and applications to show that overall there is no meaningful increase or decrease in frame rate janks when α is set such that TUS throttles at the same maximum frequency as FC. This implies that if this configuration TUS consumes less power than FC for the same performance then there is the potential to reinvest this power by increasing α. A sufficient increase in α would also raise the maximum frequency possible under TUS to 1 or more levels above FC and is likely, therefore, to improve interactive performance including frame rate janks. Figures 4.10 and 4.11 show this effect, up to 4% power consumption reduction is possible by switching to TUS from FC and this may be reinvested to increase α with a relative improvement of up 8.5% in frame rate janks. Power savings are largest when the frequency cap is lower for all four applications.

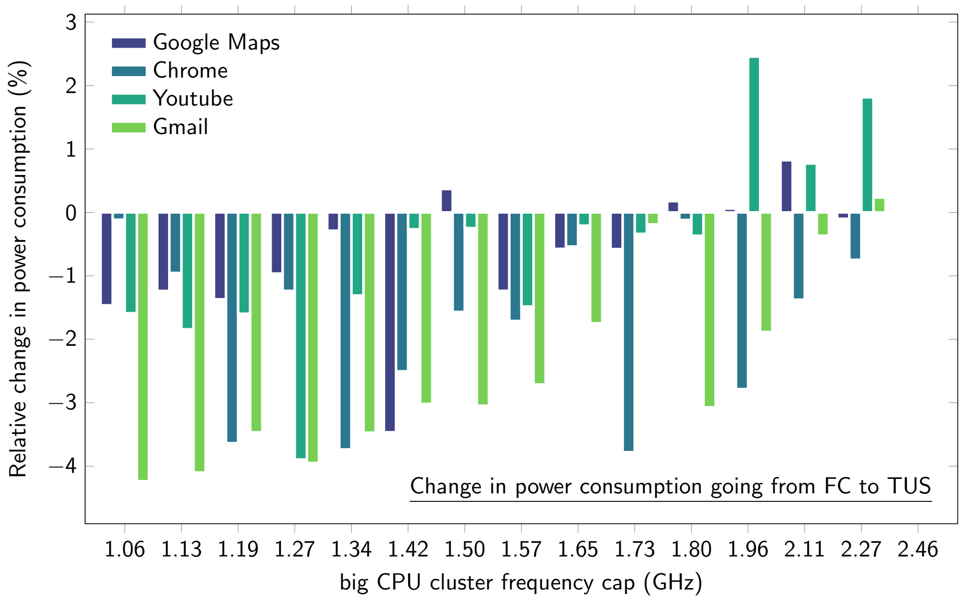


FIGURE 4.10: Change in CPU power consumption from swapping to Task Utilisation Scaling from Frequency Capping relative to the value using Frequency Capping.

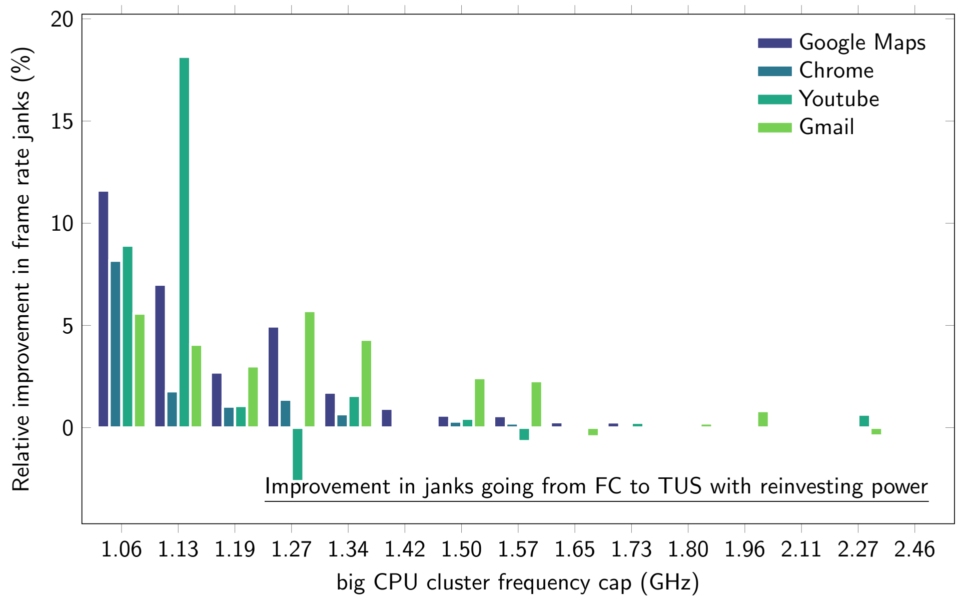


FIGURE 4.11: Improvement in frame rate janks from swapping to Task Utilisation Scaling from Frequency Capping relative to the value using Frequency Capping when power saved is reinvested to increase the maximum CPU frequency.

The comparisons between FC and TUS using Android SDK applications (Google Maps, Gmail, YouTube, and Chrome) has demonstrated that TUS can save improve interactive performance including frame rate janks at the same power consumption as FC. However, TUS must also be evaluated against its primary purpose of device skin temperature throttling. This requires experimentation using sustained performance applications such as Android SDK applications are not usually thermally constrained.

Figure 4.12 showcases a comparison between FC and TUS for executing the mobile

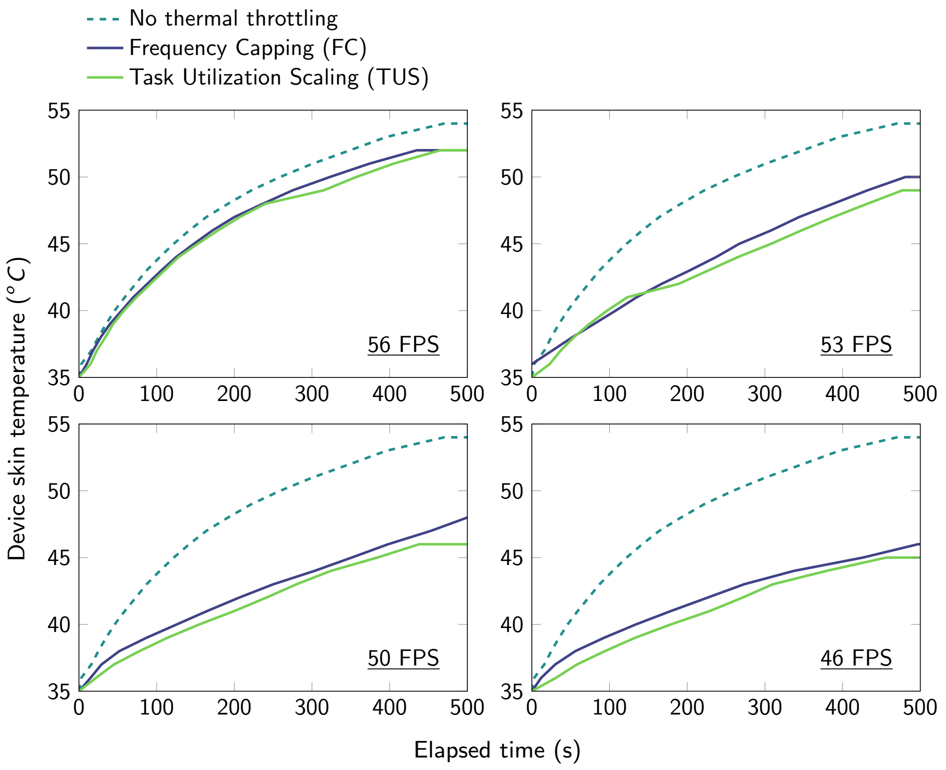


FIGURE 4.12: Device skin temperature measured over time for frequency capping (FC) and task utilisation scaling (TUS) techniques at {46, 50, 53, 56} frames-per-second performance in 3D mobile game Player Unknown Battlegrounds.

game Player Unknown Battlegrounds as in Figure 4.2. This Android NDK application is extremely computationally demanding when configured to the maximum display resolution, graphics and FPS limits. As before, the Task Utilisation Scaling factor α was configured to match the f_{cap} used by FC. For all four f_{cap}s measured this resulted in the same average FPS, this was expected as the maximum CPU frequency was the same. However, for all four f_{cap}s, TUS remained 1-2^oC below FC in device skin temperature. This difference is observed due to the reduced power consumption from throttling non-critical periods of execution.

4.3 Summary

Device skin temperature throttling has emerged as significant power challenge in mobile devices as cooling techniques cannot keep pace with the power dissipation from emerging sustained performance applications. This chapter has investigated the Dynamic Power Management lever used by many device thermal throttling systems, Frequency Capping, showing through experimentation the increase in frame rate janks it causes in interactive applications. A new DPM lever was proposed, Task Utilisation Scaling, that throttles both non-critical and critical periods of execution. When compared with Frequency Capping it consumes less power at the same performance, this power can be reinvested in higher interactive performance including frame rate janks. Task Utilisation Scaling was also evaluated for efficacy in device thermal throttling itself comparing favourably with Frequency Capping. Further investigation is required to determine if Task Utilisation Scaling is viable across other devices and applications, but it shows promise in mitigating some of the negative effects of device thermal throttling.

Chapter 5

Accelerating the Profiling of Mobile DPM Policy Parameters

The challenges facing mobile Dynamic Power Management (DPM) are well understood and were summarised in Chapter 2. However, limited empirical evidence of tuning DPM policies has been recorded so challenges in this process are less clear. Commercial implementations of the DPM tuning process in industry are confidential and the results rarely shared as it is a differentiator between competing device manufacturers. In Chapter 3, one such tuning challenge was shown to be the scale of the DPM policy parameter space and corresponding time required by offline profiling for static and dynamic configuration. Furthermore, in Chapter 4, the high-variance of the frame rate janks measurement was a challenge for both the evaluation and analysis that followed. In this chapter, these new challenges uncovered by the work of this thesis are further explored with solutions proposed. In Section 5.1, the reasons behind the high variance in frame rate janks measurements are investigated (Section: 5.1.1) and a new metric is proposed, PPPx, as an estimator for frame rate janks to accelerate evaluation (Section 5.1.2). PPPx is validated in a DPM policy tuning process of greater scale than that of Chapter 3 in Section 5.2. Three separate DPM policies are included along with their respective threshold, feature flag and timer parameters (Section 5.2.1). A tuning process was derived to tune these policy parameters and then put into practice to evaluate the potential of static configuration of DPM parameters for both performance and energy improvement when executing interactive mobile applications (Section (5.2.2).

5.1 Evaluation of frame rate janks

Mobile device DPM policy parameters have proved to be particularly challenging to tune as a result of diverse interactive workloads and performance metrics focused on tail latencies and n$^{\text{th}}$ percentile performance. In this section, evaluation of frame rate janks

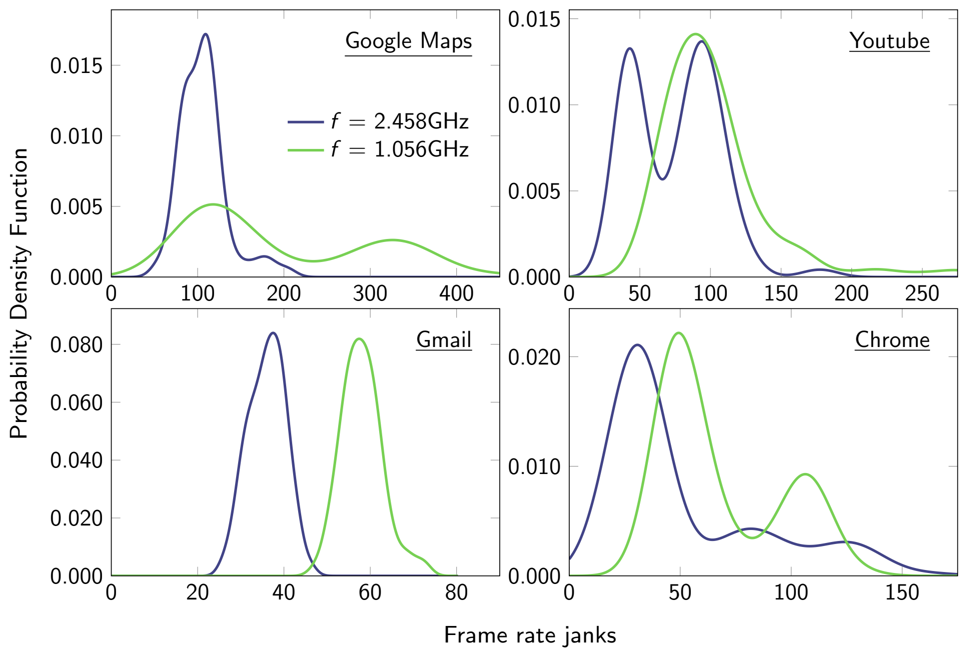


FIGURE 5.1: Probability Density Functions for frame rate janks during application execution on a Google Pixel 2 smartphone at two CPU frequency caps {1.056GHz, 2.458GHz}.

is investigated to explore the high variance in measurement compared to other metrics such as CPU energy (Section 5.1.1). It is proposed that a new metric, PPPx, is used to estimate frame rate janks measurement to overcome this challenge (Section 5.1.2).

5.1.1 Multimodal frame rate jank distributions

In mobile devices, important interactive performance metrics include frame rate janks and latency (Section 2.3.1). Frame rate janks is a measure of the number of frames for which the rendering deadline is missed, 16.7ms in the case of 60 FPS. In Chapter 4, frame rate janks was shown to be a high variance measurement with the noise induced proving a challenge for analysis of the effect of thermal throttling on interactive performance. Investigation used the same experimental setup shown in Figure 4.3 including the same four interactive mobile applications (Google Maps, Chrome, Youtube and Gmail) with execution sequences from Workload Automation [258].

Each application execution sequence was repeated 150 times on the Google Pixel 2 smartphone for two CPU frequency caps, 1.056GHz and 2.458GHz, the Probability Density Functions (PDFs) are shown in Figure 5.1. The selection of 150 repeats is due to a constraints on total execution sequences per experiment of 750 due to limitations of the experimental setup. There are three patterns observable from the data: first the probability density function of frame rate janks is multimodal for the three of the four applications that require the most network resource accesses. Secondly, frame rate janks measurements exhibit high variance due to reliance on nth percentile performance

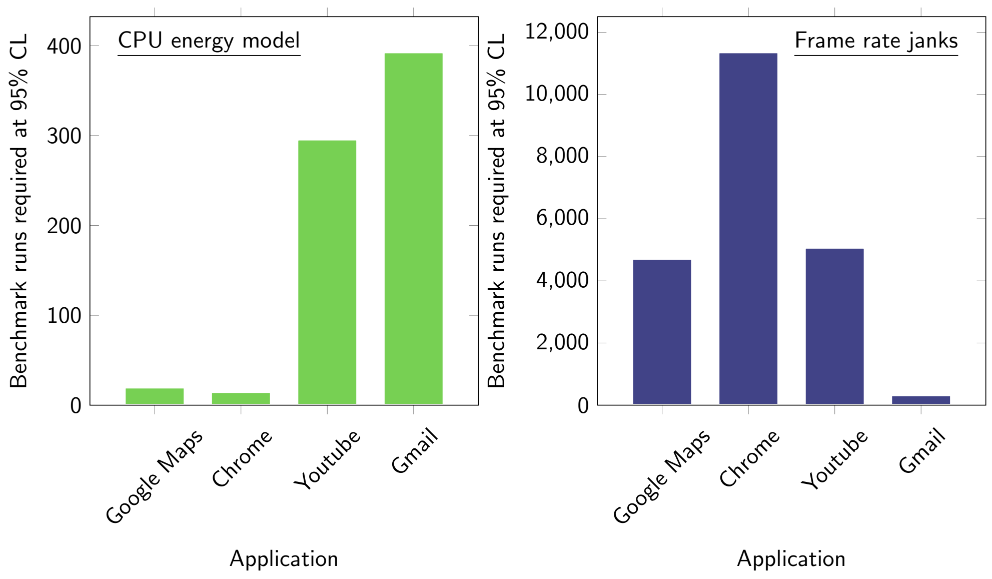


Figure 5.2: Number of benchmark runs required to detect a 1% difference in mean CPU energy model and frame rate janks between two parameter configurations at the 95% confidence level.

[102]. Thirdly, the distributions change when the CPU frequency cap and, therefore, performance are varied.

The multimodal probability density functions observed exacerbate the high variance measurement of frame rate janks. In Figure 5.2, the number of benchmark runs required to detect a 1% change in mean in either a CPU energy model metric or frame rate janks is compared using a 95% confidence level. The CPU energy model uses CPU frequency and idle state occupancy to estimate energy consumption (Section 4.2.2). For the three applications exhibiting a multimodal probability density function, the number of benchmark runs required for frame rate janks is between a 17x and a 758x multiple.

In absolute terms, to detect a 1% difference in mean frame rate janks at the 95% confidence level can require over 11,000 repeats of a benchmark lasting approximately one minute. If this is converted into device days (a device day is 24-hours of continuous benchmarks on one mobile device), then the result is over seven device days. The experiments in Chapters 3 and 4 would both have taken multiple years to run if evaluated at this confidence level. Yet both experiments require significantly more data to prove efficacy, perhaps orders of magnitude more results when further applications and devices are considered. A further problem is that a 1% difference in mean frame rate janks is not necessarily a marginal gain when considering tuning DPM policy parameters and it would be useful to compare smaller differences than this.

Clearly there is a challenge in evaluating frame rate janks within a reasonable timeframe, in a commercial mobile device lab they proposed DPM policies are disabled to increase repeatability [112], evidently this is not an option for evaluation of DPM policy tuning. The standard metric used as a proxy for frame rate janks is to measure the

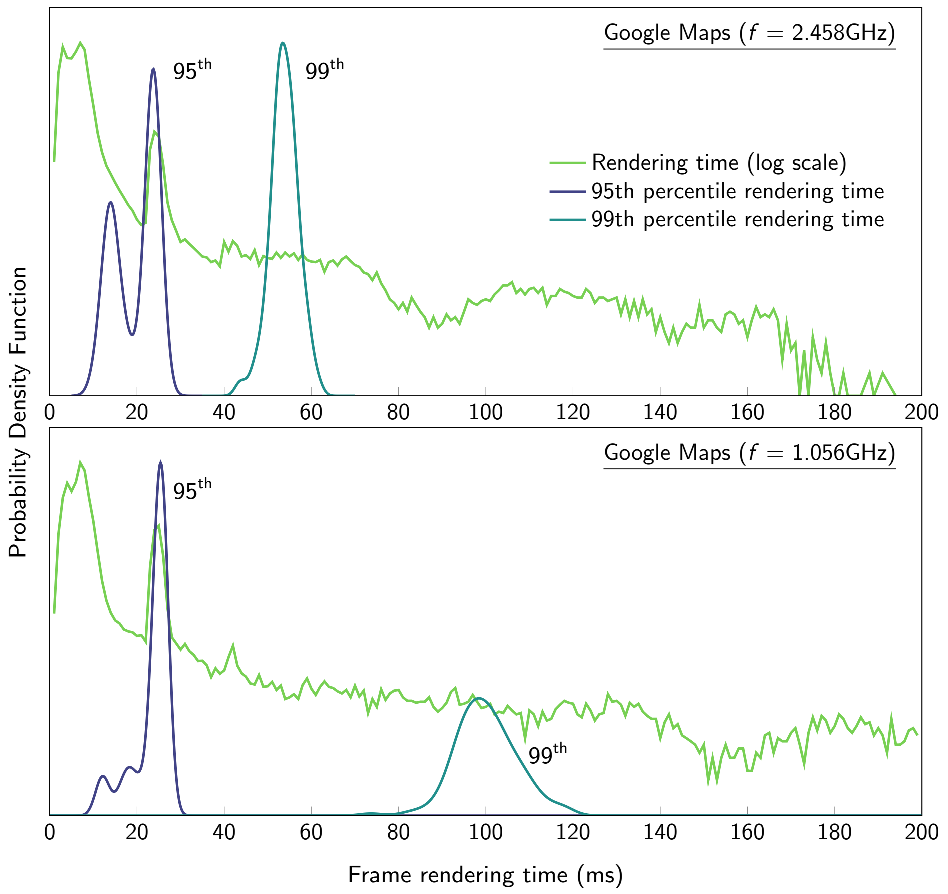


FIGURE 5.3: Frame rendering times probability density functions at 2.458GHz and 1.056GHz CPU frequencies, with peak probabilities at 8.3ms, 25.0ms, 41.7ms.

n$^{\text{th}}$ percentile frame rendering time [102]. The PDFs of frame rendering times using the same experiment data as Figure 5.1, are shown in Figure 5.3 for Google Maps. The PDF of frame rendering times are plotted using a logarithmic scale because they exhibit a long-tailed distribution. The majority of frame deadlines are met successfully therefore the probability of a frame rendering time being between 0 and 16.7ms is significantly higher.

The PDF curves are not smooth, with peak probabilities at 8.3ms, 25.0ms, 41.7ms. The significance of these times is that 8.33ms is halfway between 0ms and the 60 FPS frame deadline of 16.7ms, 25.0ms is halfway to a second missed deadline of 33.3ms, and this pattern continues. If the n$^{\text{th}}$ percentile frame rendering time chosen is between two of these peaks, then the distribution is multimodal with increased variance. For Google Maps at 1.056GHz, 95$^{\text{th}}$ percentile would be a low variance choice; however, if the frequency cap is increased to 2.458GHz then this would be a high variance choice. Across all four applications this pattern is visible, in particular, the peak probability at 25.0ms causes the largest increases in variance. The variation of n$^{\text{th}}$ percentile variance presents a problem when tuning DPM policy parameters as it will vary between configurations.

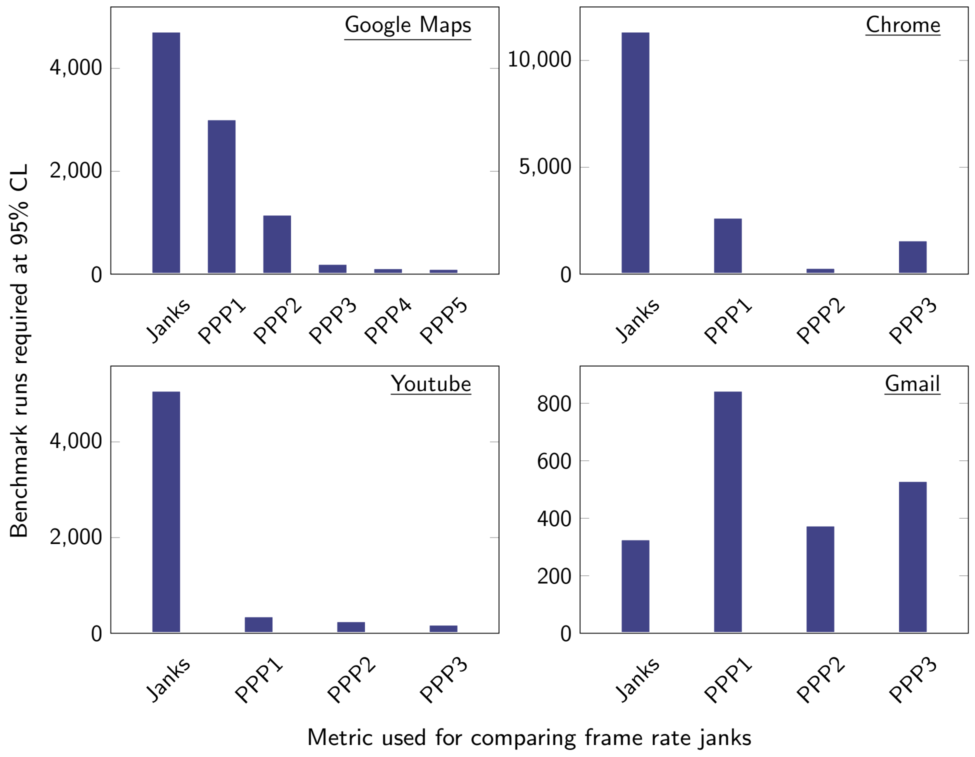


FIGURE 5.4: Number of benchmark runs required to detect a 1% difference in mean frame rate janks between two parameter configurations at the 95% confidence level. The value for janks is for measuring janks directly, the values for {PPP1, PPP2, PPP3, PPP4, PPP5} are for comparing indirectly PPPx.

5.1.2 PPPx: A new metric for estimating frame rate janks

The peak probabilities in the probability density function in Figure 5.3 are problematic for estimating frame rate janks using a nth percentile approach. However, the peak probabilities also offer an opportunity as their locations are fixed at 8.3ms, 25.0ms, 41.7ms regardless of CPU performance. Instead of measuring the nth percentile frame rendering time, this can be reversed such that the Peak Probability Percentiles are measured in a new metric, namely PPPx. PPPx, like nth percentile frame rendering times is designed as an estimator for frame rate janks.

The initial PPPx is at 8.3ms (PPP0), the next at 25.0ms (PPP1), the next at 41.7ms (PPP2) and this pattern continues. For example, at 1.056GHz, the PPP1 is 93.0% and at 2.458GHz it is 96.7%. From this we can conclude that 7.0% of frames take longer than 25.0ms to render at 1.056GHz and 3.3% at 2.458GHz, this is compared to a mean of 198.8 and 106.5 janks respectively. There is over an order of magnitude difference in frequency between frame rendering times that miss one deadline and two deadlines. In addition, frame rendering times for one deadline miss are approximately normally distributed unless DPM is configured to shift this. It follows that 1-PPP1 is approximately equal to half the overall frame rate janks.

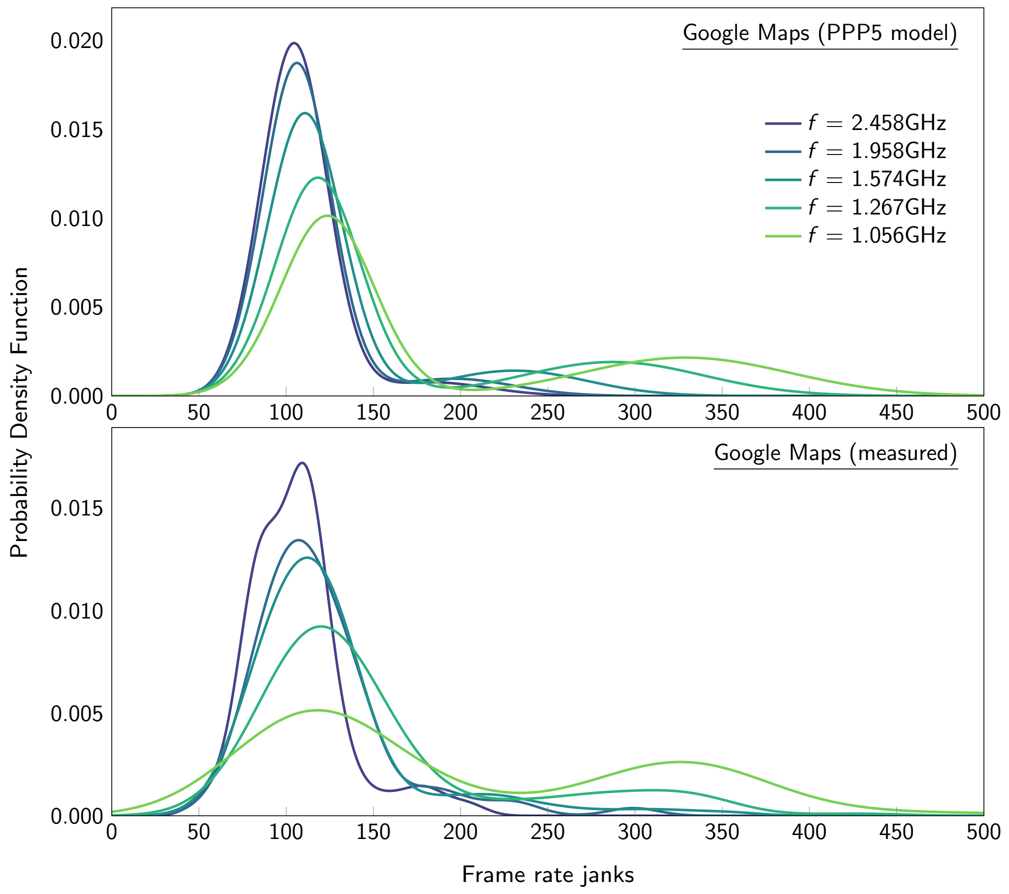


FIGURE 5.5: Comparison between a model derived from the PPP5 (percentile at 91.7ms) and measured janks for Google Maps. The measured plot uses the same data as Figure 5.1. A Gaussian Mixed Model was constructed by fitting a linear relationship between the mean, weight, standard deviation and the PPP5 for two components as shown in Figure 5.6.

The best PPPx to use varies between application and must be profiled in the same way as the n$^{\text{th}}$ percentile frame rendering time metric. In general, the PPP1 is best correlated with frame rate janks, but higher PPPx choices have less relative variance and, therefore, require fewer benchmark repeats. In Figure 5.4, the PPPx choices are compared for each of four applications. For the three applications with multimodal jank distributions, the PPP5, PPP3 and PPP2 are optimal; for the unimodal jank distribution application, Gmail, it is optimal to instead measure the janks directly. The speed-up compared to measuring janks directly is up to 43.3x.

The probability density function for janks may be fitted from the percentiles measured by decomposing the multimodal function into a mixture of gaussians. In Figure 5.5, a Gaussian Mixed Model (GMM) with two components was created from the janks probability density function. Each of the two components has a mean, weight and standard deviation estimated at five frequency caps between 1.056GHz and 2.458GHz.

The relationship between PPPx and frame rate janks is approximately linear unlike the standard metric using a fixed n$^{\text{th}}$ percentile which is related by a polynomial function.

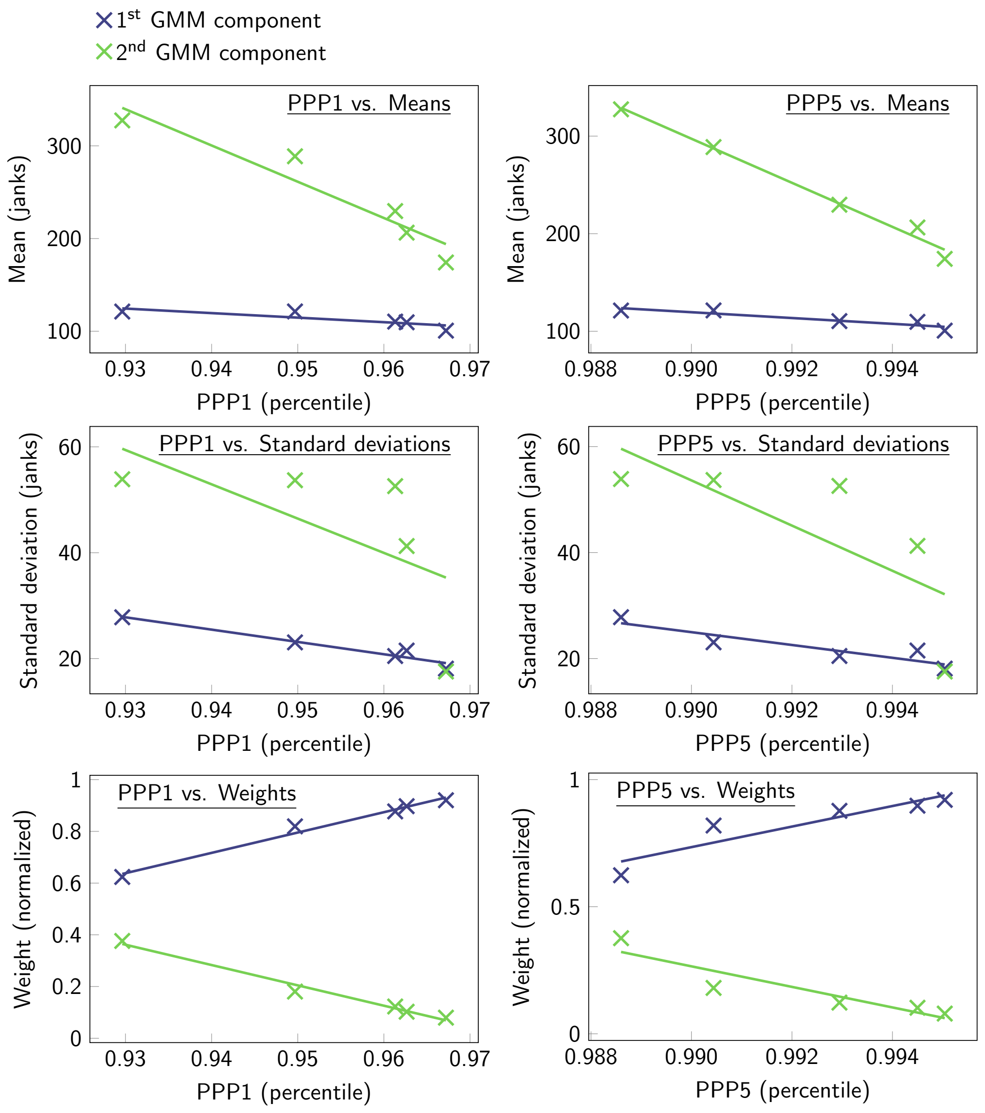


FIGURE 5.6: Correlation between PPPx metrics (PPP1 and PPP5) and the components of a Gaussian Mixed Model for Google Maps frame rate janks.

GMM component means and weights are simple to fit using a linear relationship whereas GMM component standard deviation can be more difficult due to its reliance on more data for an accurate estimation. In Figure 5.6, PPP1 and PPP5 are fitted for Google Maps GMM mean, weight and standard deviation components. It is observable that the lack of data at higher performance levels for the 2nd GMM component means the standard deviation correlation is not as strong, an alternative approach would be to extrapolate results from the performance levels where there was sufficient data. Irrespective of this, the proposed metric of measuring the PPPx is shown to be sufficiently correlated to use as a reasonable estimator for frame rate janks and brings along with a speedup of evaluation of up to 43.3x.

Parameter	Category	Minimum	Maximum	Default
scaling_max_freq	DVFS	300,000	2,457,600	2,457,600
up_rate_limit_us	DVFS	1	N/A	500
down_rate_limit_us	DVFS	1	N/A	20,000
sched_use_walt_cpu_util	Load Tracking	0	1	1
sched_use_walt_task_util	Load Tracking	0	1	1
sched_walt_init_task_load_pct	Scheduler	0	100	0
sched_tunable_scaling	Scheduler	0	2	0
sched_child_runs_first	Scheduler	0	1	1
sched_latency_ns	Scheduler	100,000	N/A	10,000,000
sched_migration_cost_ns	Scheduler	1,000	N/A	500,000
sched_min_granularity_ns	Scheduler	100,000	N/A	3,000,000
sched_wakeup_granularity_ns	Scheduler	1,000	N/A	2,000,000

Table 5.1: Dynamic Power Management parameters selected for tuning in the Google Pixel 2 kernel across Dynamic Voltage and Frequency Scaling, load tracking and the scheduler. Valid parameter values are heterogeneous including binary, categorical and integer types.

5.2 Tuning Dynamic Power Management policies using PPPx

The DPM policy parameter tuning experiments in Chapter 3 only considered the parameters of DVFS policies. In this section, static configuration is evaluated for the parameters from three DPM policies. An order for static configuration is derived based on characteristics of the DPM policy parameters (Section 5.2.1). The order is then implemented using the proposed metric - PPPx - to determine potential improvements in energy and frame rate janks across the four applications (Section 5.2).

5.2.1 Policies, parameters and the tuning process

DPM policies included for tuning were a DVFS policy, a Load Tracking (LT) policy and the scheduling policy. Parameters selected for configuration are listed in Table

5.1 along with their category, minimum, maximum and default values. The selection includes three DVFS policy parameters, two LT policy parameters and seven scheduler parameters. Value types include enumerations, thresholds, feature flags and timers.

Using the example of the benchmarks from Workload Automation [258] in the previous experiments, it is only possible to execute around 1000 benchmark runs per device day. Over a sustained period of running benchmarks on a device without restarting, the performance drifts, which limits the effective sample set size. It is not desirable to restart the device every run as that is not representative of real-world usage; empirically it was determined that the maximum set size for the Google Pixel 2 setup was 500 benchmark runs that can be compared reliably. Furthermore, the parameter configurations under test should be executed in turn before repeating to reduce the effect of performance drift.

The limited sample set size for comparison of benchmark runs means that static configuration must take place in stages to allow sufficient repeats. Manual configuration instructions are available for some parameters, but others have limited published information about their function. To understand the function of the LT policy and scheduler policy parameters, initial profiling experiments were carried out using the Google Maps execution sequence from the experimental setup used in Section 5.1. The 500 benchmark run sample size was divided into 25 configurations with 20 repeats each, a compromise between wider comparison and frame rate jank measurement accuracy.

LT policy parameters are two binary feature flags which define the LT algorithm used for tracking CPU utilisation (*sched_use_walt_cpu_util*) and individual task utilisation (*sched_use_walt_task_util*). The feature flags select between Per Entity Load Tracking (PELT) and Window Assisted Load Tracking (WALT), the default configuration is for WALT to be used for both CPU and task utilisation tracking. In Figure 5.7, the four combinations of PELT and WALT were evaluated with random scheduler parameters. PPP1 and a CPU energy model (Section 5.1) were evaluated as metrics. The performance and energy metrics are clearly divided into clusters by the LT feature flags, of which the default setting (cpu=WALT, task=WALT) is lower performance than (cpu=PELT, task=WALT). For Google Maps it is therefore optimal to use PELT to track CPU load and WALT to track task load. Mixing and matching PELT and WALT is defined as experimental in a tuning guide [51]; it is standard practice to use either WALT or PELT but not both. This finding of ignoring tuning guide recommendations for improved performance was also found in datacentre application tuning [105].

The scheduler parameters each make marginal adjustments to the task scheduling logic. In Figure 5.8, the aforementioned LT policy combination of (cpu=PELT, task=WALT) was used but varied the scheduler parameters over 25 configurations. There were variations in energy and PPP1 between parameter configurations, albeit smaller than those of the LT feature flags. Additionally, it was observed there was noise added to the PDF

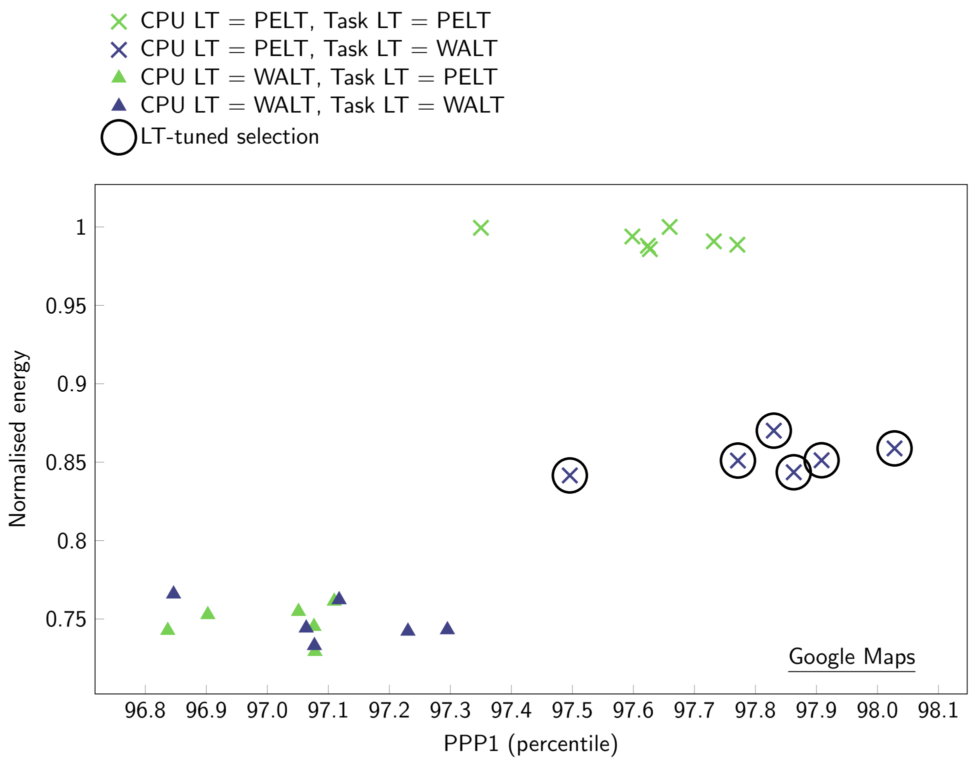


FIGURE 5.7: Comparison of 25 parameter configurations varying LT and scheduler parameters.

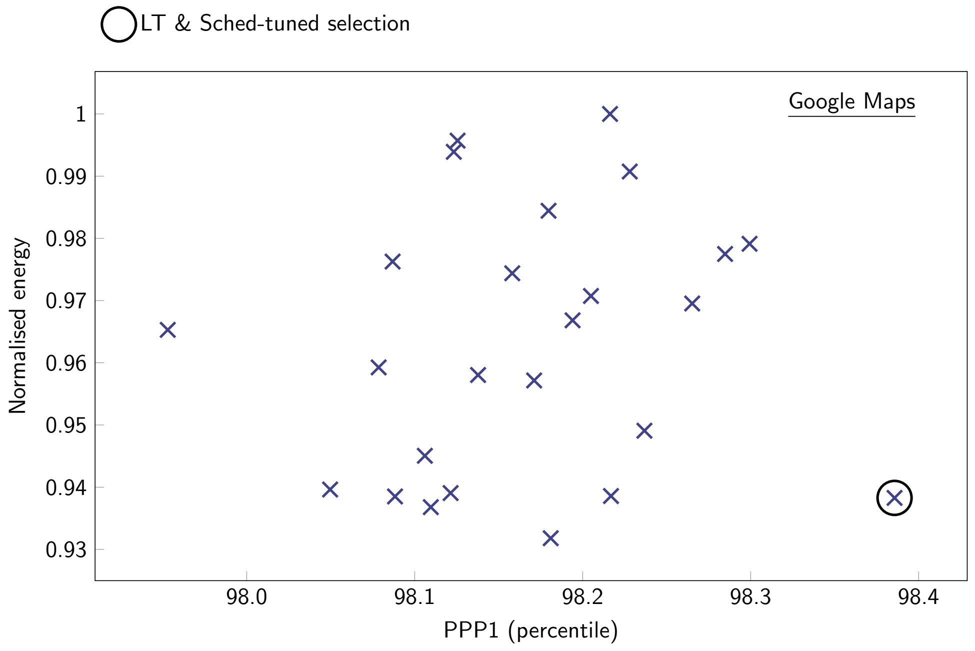


FIGURE 5.8: Comparison of 25 scheduler parameter configurations using the LT-tuned configuration from Figure 5.10.

for frame rendering times which reduced the correlation between the PPPx and the corresponding frame rate janks.

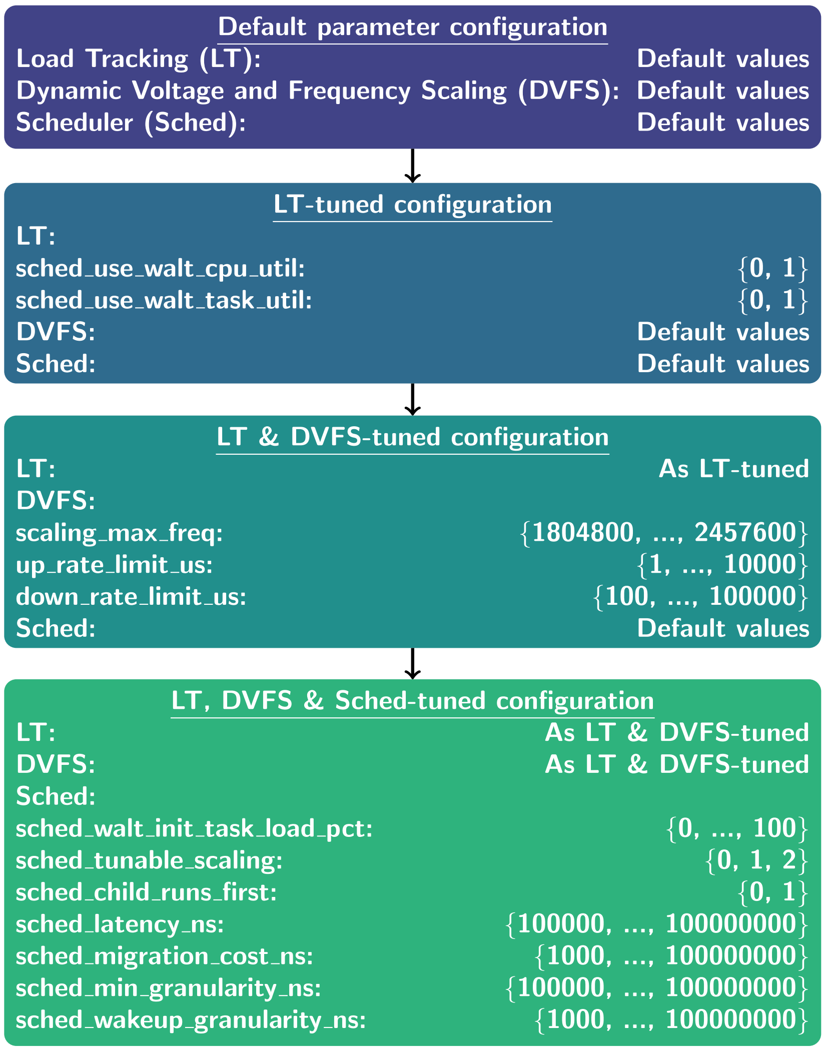

Default parameter configuration
Load Tracking (LT): Default values
Dynamic Voltage and Frequency Scaling (DVFS): Default values
Scheduler (Sched): Default values
LT-tuned configuration
LT:
sched_use_walt_cpu_util: {0, 1}
sched_use_walt_task_util: {0, 1}
DVFS: Default values
Sched: Default values
LT & DVFS-tuned configuration
LT: As LT-tuned
DVFS:
scaling_max_freq: {1804800, ..., 2457600}
up_rate_limit_us: {1, ..., 10000}
down_rate_limit_us: {100, ..., 100000}
Sched: Default values
LT, DVFS & Sched-tuned configuration
LT: As LT & DVFS-tuned
DVFS: As LT & DVFS-tuned
Sched:
sched_walt_init_task_load_pct: {0, ..., 100}
sched_tunable_scaling: {0, 1, 2}
sched_child_runs_first: {0, 1}
sched_latency_ns: {100000, ..., 100000000}
sched_migration_cost_ns: {1000, ..., 100000000}
sched_min_granularity_ns: {100000, ..., 100000000}
sched_wakeup_granularity_ns: {1000, ..., 100000000}

FIGURE 5.9: Stages of the static configuration process for mobile DPM parameters determined through initial experiments. At each stage the parameters from the best configurations of the previous stage are used unless specified otherwise.

Based on these initial experiments, the static configuration order in Figure 5.9 was determined. Starting from a base of default parameter configuration for DVFS, LT and scheduler, there are three stages of tuning. First the LT feature flags are configured; experiments clearly showed that these divide the parameter space into four clusters and the distance between these clusters is large enough that interactions with other parameters should not be relevant. Secondly, the DVFS parameters are tuned with the best LT configuration; in this case each of the parameters have had their ranges reduced to a subset of the total space but still extending above and below default values. Thirdly, the scheduler parameters are configured using the best LT and DVFS configuration.

In each stage of the tuning process, the Divide and Diverge Sampling (DDS) technique, proposed independently by Chapter 3 of this thesis and another work [1, 105], is used to sample the 25 parameter configurations. DDS divides the full range of each parameter by the number of configurations, in this case 25, and mixes the ranges together (diverges)

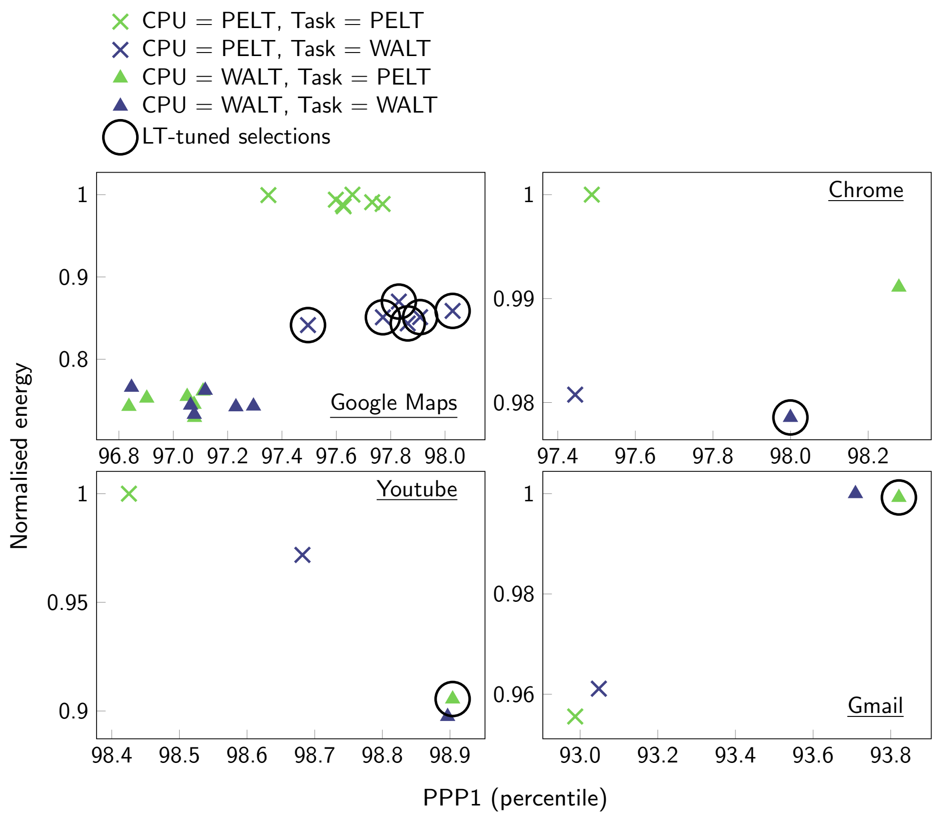


FIGURE 5.10: Comparison of the four LT configurations across all four applications. Three of the four LT permutations are selected across the applications which is evidence of the value in tuning even binary feature flags specific to each application.

for all the parameters. In the case of timer parameters, a logarithmic scale was used for sampling due to the wide range of possible values.

5.2.2 Performance improvements and energy savings

The static configuration order detailed in Figure 5.9 was carried out for four applications on the Google Pixel 2 using the Workload Automation execution sequences [258]. In each stage of the tuning process, the sample set size of 500 was divided into up to 25 parameter configurations with 20 repeats each. For the summary results, the best configuration from each tuning stage was repeated 100 times in the same experiment to achieve more accurate results for comparison.

The LT tuning stage for Google Maps has already been shown in Figure 5.7; the default configuration (cpu=WALT, task=WALT) was shown to be inferior in performance to (cpu=PELT, task=WALT). The respective LT tuning stages for all applications are illustrated in Figure 5.10. The configuration selected for Chrome was the default configuration of (cpu=WALT, task=WALT), for Youtube either (cpu=PELT, task=WALT) or (cpu=WALT, task=WALT) could have been selected and for Gmail (cpu=WALT, task=PELT) was selected. Across the four applications (cpu=PELT, task=PELT) was

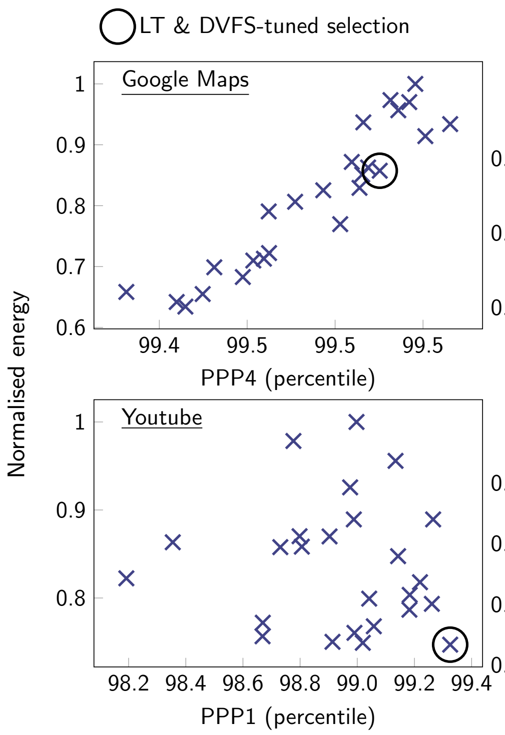


FIGURE 5.11: Comparison of 25 DVFS parameter configurations for all four applications Maps using the optimal LT configuration. The selected configurations are circled; they were selected due to lower CPU energy at near-maximum performance. A Pareto front of other parameter configurations is visible.

not selected but the other three combinations were. This static configuration stage further reinforces the application dependent response of tuning DPM policy parameters.

In Figure 5.11, 25 configurations of DVFS parameters are compared; the LT configuration was brought forward from the previous stage. Whilst there were higher performance configurations along the Pareto-optimal curve, the selected configurations were chosen for reduced energy consumption. The best configurations had minimal reductions in CPU frequency but had lower rate-limits to ramp up and down in frequency faster.

PPP4 was used for Google Maps as a trade-off between the lowest variance metric - PPP5 - and the highest correlation with frame rate janks metric - PPP1. PPP1 was used for Chrome, Gmail and Youtube. Data from a wider range of applications is required to understand why the frame-rate janks for some applications has a stronger correlation between PPP4 and PPP5 than other applications. Google Maps offers the widest range of trade-offs between performance and energy, followed by Chrome. Gmail and Youtube do not benefit from large performance gains and therefore the optimisation is focussed on energy.

If more granularity in trade-offs was required, then the 500 benchmark runs could be divided into more configurations with fewer repeats and the reverse is true for more

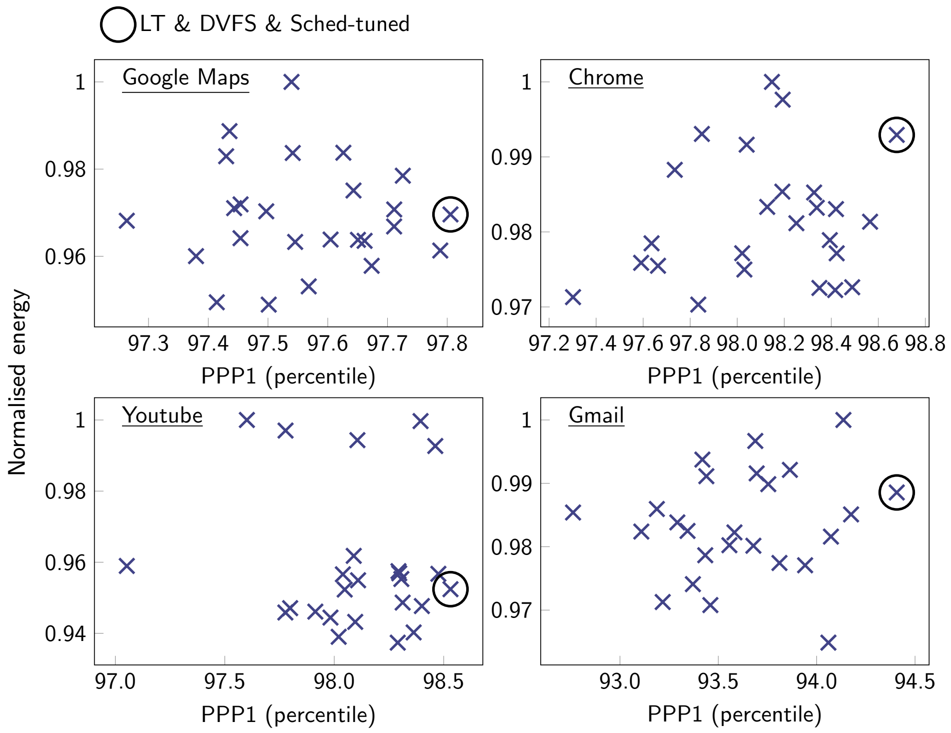


Figure 5.12: Comparison of 25 scheduler parameter configurations for all four applications using the optimal LT and DVFS configurations. The configurations selected due to their improved performance are circled; however, there is minimal difference in energy or performance between the configurations tested.

accurate results. Moreover, the range of valid values for the DVFS policy parameters could be reduced or increased based on preliminary profiling.

In Figure 5.12, 25 configurations of scheduler parameters are compared; the LT and DVFS configurations were brought forward as before. As in Figure 5.8, there appears to be minimal difference between configurations but this is evaluated with greater accuracy in the 100 repeat comparison. Irrespective of this, optimal parameter configurations are selected using PPP1 and energy model metrics. As there was more performance variation than for energy, selected configurations were picked based on highest performance only.

A representative idea of performance may be achieved by comparing percentiles of frame rendering time over 20 benchmark runs; however, as shown in Figure 5.5 the frame rate janks model derived from PPPx is not perfect and 20 runs is not a large sample. To overcome this and to obtain comparable results, the best configuration from each stage of the tuning process was compared in the same test over 100 runs. The result for frame rate janks is in Figure 5.13 and that of energy in Figure 5.14. It is observable from the results that, for Google Maps the largest differential in frame rate janks comes from tuning the LT feature flag parameters. In the case of energy, it is the expected result that tuning DVFS parameters has the most effect. Either the LT & DVFS or LT, DVFS & Sched configurations are improvements over the default configuration. The improvement in mean frame rate janks is 6% and the mean energy reduction is 4%. In

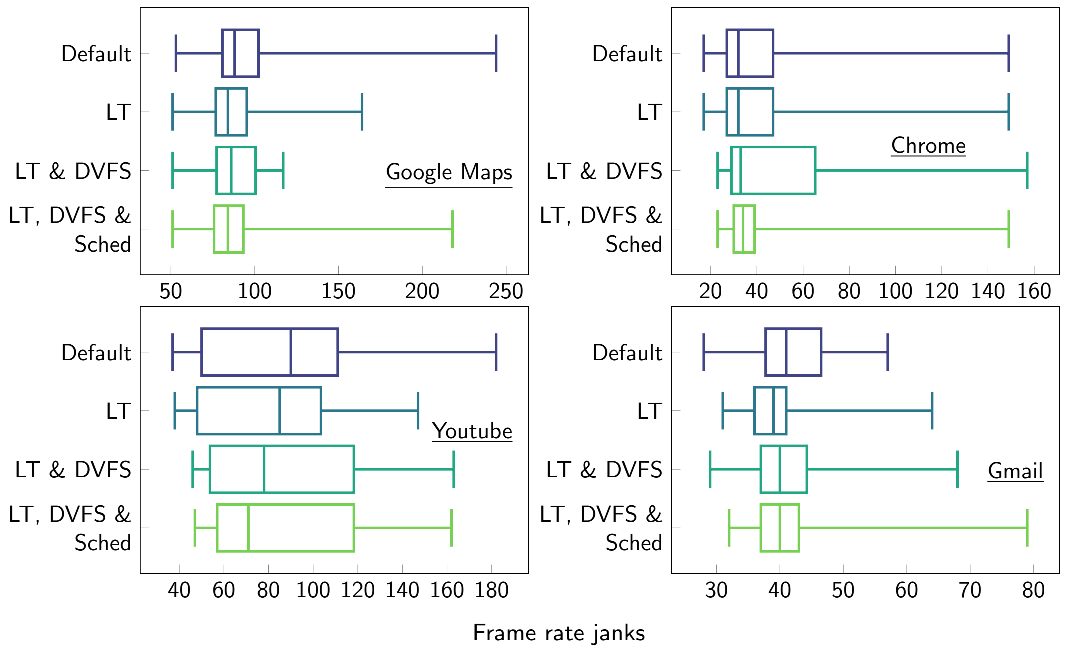


Figure 5.13: Frame rate janks compared over 100 benchmark runs between the stages of the static configuration including the default configuration.

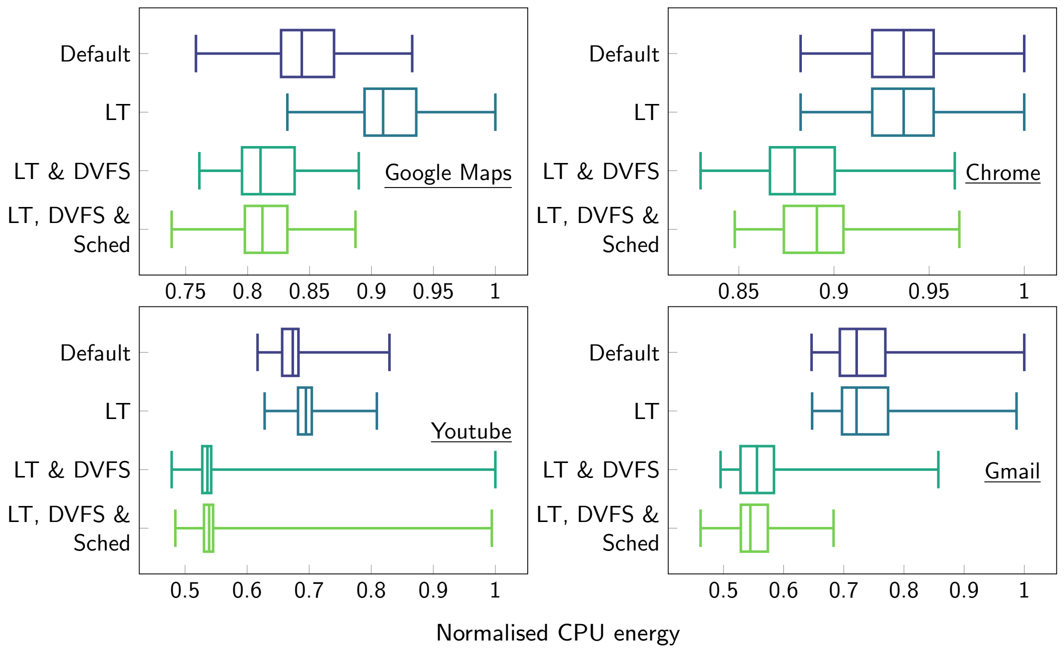


Figure 5.14: CPU energy compared over 100 benchmark runs between the stages of the static configuration including the default configuration.

addition, there is a significant reduction in 99^{th} percentile frame rate janks for LT & DVFS, although it is not clear if this is repeatable over a larger sample size.

Application	Mean janks	90th %ile janks	99th %ile janks	Mean energy	90th %ile energy	99th %ile energy
Google Maps						
Default	90.8	110.1	149.0	0.849	0.896	0.931
LT	85.7	110.0	134.3	0.917	0.968	0.999
LT & DVFS	**85.2**	**108.0**	**113.0**	0.817	0.855	0.884
LT, DVFS & Sched	88.4	119.4	201.2	**0.813**	**0.847**	**0.875**
Chrome						
Default	50.3	125.1	148.0	0.939	0.974	0.999
LT	50.3	125.1	148.0	0.939	0.974	0.999
LT & DVFS	52.5	130.1	**143.1**	**0.884**	0.918	0.957
LT, DVFS & Sched	**45.4**	**99.0**	**148.0**	0.890	**0.915**	**0.952**
Youtube						
Default	83.5	128.0	141.4	0.673	0.692	**0.771**
LT	**79.7**	**118.0**	**139.1**	0.695	0.718	0.800
LT & DVFS	86.5	133.1	149.1	**0.543**	0.552	0.952
LT, DVFS & Sched	87.6	131.0	160.0	0.553	**0.549**	0.988
Gmail						
Default	42.2	52.0	**55.0**	0.735	0.791	0.938
LT	**39.8**	**51.1**	62.0	0.738	0.812	0.958
LT & DVFS	41.7	53.0	58.1	0.567	0.636	0.753
LT, DVFS & Sched	42.2	53.1	65.1	**0.551**	**0.591**	**0.682**

Table 5.2: Summary results for mean, 90th and 99th percentile frame rate janks and CPU energy.

The results from each static configuration stage are not identical across all applications tested. There is the possibility that some of this effect results from the random sampling used in DDS; however, it is mostly related to each application's behaviour requiring different DPM tuning. In Table 5.2, the summary results across all four applications and for all tuning stages are listed. For example, for Chrome, the default LT configuration was optimal without modification, but the mean frame rate janks was reduced by 10% and the 90th percentile janks by 24% when the scheduler parameters were tuned. Neither Youtube nor Gmail saw notable reductions in frame rate janks from tuning; however, tuning both DVFS and the scheduler resulted in reductions in CPU energy consumption by 14% and 25% respectively. In some cases, it may have been advantageous to test more than 25 parameter configurations during a stage to obtain an improved configuration. There is naturally a trade-off between profiling time and results with diminishing returns as the sample size of parameter configurations is increased.

Trends in optimal parameter values across all application were as follows: all four applications used a CPU frequency below maximum and benefit from lower rate-limits to speed-up frequency ramp-up and down; out of the four permutations of LT configuration, three were selected across the four applications; and, finally, for scheduler parameters, reductions in the wakeup granularity (*sched_wakeup_granularity_ns*) improved performance at the expense of energy.

In total across the four applications, the results show reductions in frame rate janks by up to 10% and CPU energy by up to 25%. Traditionally, DPM policies require reductions in performance to conserve energy and vice versa on a Pareto-optimal front. Results from this case study permit simultaneous improvements in frame rate janks and CPU energy. Implementation of the static configuration process using was executed in two device-days per application, whereas if frame rate janks were measured, the process would have required more than two device-months per application. This was made possible by the up to 43.3x speedup gained from evaluation using the proposed metric PPPx.

5.3 Summary

An important interactive performance metric for mobile applications - frame rate janks - exhibits a high variance that is prohibitive for a DPM tuning process. The standard metric used as a proxy for frame rate janks is also vulnerable to high variance under variations in performance. This chapter has proposed a new metric - PPPx - measuring the Peak Probability Percentiles of frame rendering times, which can speed-up tuning by up to 43.3x. PPPx was validated in a case study tuning the parameters of three DPM policies across four applications. The static configuration process that was executed in two device-days per application would otherwise have taken more than two device-months per application of benchmark runs. Results showed differences in response to tuning for each application with reductions in frame rate janks by up to 10% and CPU energy by up to 25%. These figures are the best available to quantify the potential gains by tuning Dynamic Power Management in commercial devices compared to the current state-of-the-art in manual configuration. It is likely that further gains may be achieved by continuing research in this area with a focus on scaling performance evaluation such that the state space of policy parameters and applications can be enlarged.

Chapter 6

Conclusions and Research Opportunities

Dynamic Power Management (DPM) of mobile devices remains a key challenge faced by device manufacturers. Battery life extension and skin temperature throttling must be traded-off with user Quality of Experience (QoE) to determine a performance operating point. This is in the face of the increasing computer architecture complexity and growing DPM policy parameter state spaces. Chapters 2, 3, 4 and 5 have targeted the research aims listed in Section 1.4. In this chapter, the contributions towards these research aims are discussed along with both limitations of the findings and their wider implications for the field of DPM for mobile devices (Section 6.1). Following these conclusions, future research opportunities are highlighted that offer the potential to further develop the contributions of this thesis (Section 6.2).

6.1 Conclusions

Evidence presented in Chapters 1 and 2 leaves no doubt that mobile device Dynamic Power Management will be stretched further in the future. Application performance demands from mobile System-on-Chips continue to grow unabated outpacing both battery storage capacity and passive cooling technique innovation. In computer architecture, design metrics such as security and reliability have added to an ever-growing list. System-on-Chips are becoming more modular with reconfigurable IP-based design and with this approach the parameter space increases further. Despite this, researchers continue to place importance on adaptive hardware and software solutions to solve this introducing even more parameters. The requirement to tune mobile device Dynamic Power Management policy parameters only grows larger as the effect of these trends are accumulated. This thesis has not sought to follow the well-trodden path of developing a complex adaptive power management policy that seeks a one-size-fits-all approach to runtime Dynamic Power Management. On the contrary, the problem of tuning Dynamic Power Management policy parameters has been addressed directly, proving that there is value in the idea of not ripping up the existing Power Management frameworks and instead harnessing their capabilities through systematic tuning. The research aims in Section 1.4 embrace this concept and the contributions towards these will now be presented.

The beginning of Chapter 1 reviewed the rapid growth and then commoditisation of the mobile device market since the release of the first smartphone. In a commodity market, margins have fallen, and device manufacturers must focus their resources on research areas with the best return on investment. A business case for tuning mobile Dynamic Power Management parameters may not be made without quantification of the performance and power improvements that are achievable through its use. In Section 3.1, an experiment that tuned a mobile Dynamic Power Management policy for executing web browsing applications efficiently was documented. Variability in resource access-times and user input was exploited using offline profiling and the results used in dynamic configuration for up to 13% CPU energy savings or a user Quality of Experience improvement of 27%. The size of these figures is not revolutionary by any means but when put in context of other architectural improvements to Dynamic Power Management policies not insignificant. The limitations of this experiment were twofold. First, despite using a mobile System-on-Chip and Operating System, a real mobile device wasn't used. Secondly, only three user input types were tested in a single application, this is far from the scale required to develop a business case. The experiment in Section 5.2 went some way to remedying these limitations. A modern mobile device was used - the Google Pixel 2 smartphone - and four applications were evaluated with configuration of three Dynamic Power Management policies. The results were similar in magnitude to the first experiment but swapped between performance and energy, with a user Quality of Service improvement of up to 10% and a CPU energy reduction of up to 25%. Multiple data points from devices, applications and policies forms the start of quantifying the potential improvements achievable but there will always be an insatiable demand for more data given the scale of the research problem.

If this thesis was compiled even half a decade previously, device skin temperature throttling would have been a minor concern. The technology race in mobile devices means that research must be constantly evaluated to ensure relevancy with current and future challenges. Skin temperature throttling solutions have been rapidly developed in industry to tackle the challenge, but the only side-effects considered have been on average performance metrics. Section 4.1.2 demonstrated that this approach was short-sighted in disregarding the effect of Frequency Capping on interactive performance such as frame rate janks. The study did have technical limitations requiring data from sustained performance Android NDK applications to be combined with Android SDK applications to reach its conclusions. Nevertheless, the importance of thorough performance analysis when designing and tuning Dynamic Power Management systems was highlighted. The new Dynamic Power Management lever proposed - Task Utilisation Scaling - was designed to use existing capabilities in the Dynamic Power Management framework of the Google Pixel 2. The performance or power gain possible would likely be categorised as marginable, yet in its favour is the design which permits drop-in replacement of Frequency Capping in existing systems with minimal friction. It should not be underestimated the roles of simplicity and ease of implementation in driving adoption of new techniques.

Global deployment of datacentre applications by major technology companies has created the economies of scale for profitable investments in performance analysis and tuning research. Static configuration of software and Operating System parameters using systematic processes has been shown to outperform manual tuning by human experts. The number of datacentre applications is far lower than in mobile applications so to achieve the same economies of scale, the cost of tuning must be lowered. Section 5.1 investigated the reasons why detecting a small change in frame rate janks, an interactive performance measurement, can require over 750x the benchmark repeats as a CPU energy model. This finding has so far only included interactive mobile applications with significant network usage but may prove widely applicable. Latency measurements were not been considered but with a similar long-tailed distribution may also suffer from the same problem. The solution proposed - a new metric PPPx - was shown to speedup evaluation of frame rate janks by up to 43.3x. This is an order and a half magnitude improvement, more of which will be required to propel static configuration into widespread use. Performance and power models have thus far largely been focussed on absolute accuracy, by considering relative accuracy to compare configurations more gains could be achieved. Validation of PPPx highlighted it is vulnerable to noise introduced in the distribution of frame rendering times, but this was overcome by choosing more conservative values of x such as PPP1 rather than PPP5. The validation experiments requiring 8 device-days in total rather than 8 device-months was testament to the potential in accelerated evaluation of performance and power metrics.

In conclusion, this thesis cannot present a definitive business case for widescale deployment of tuning processes for mobile Dynamic Power Management policies; however, results are sufficiently promising to recommend pilot studies at a moderate scale across more mobile devices and applications. These pilot studies would require capital investment in approximately 100 mobile devices and at that point the quantification of potential performance and power improvements would be clear enough to determine if the business case exists.

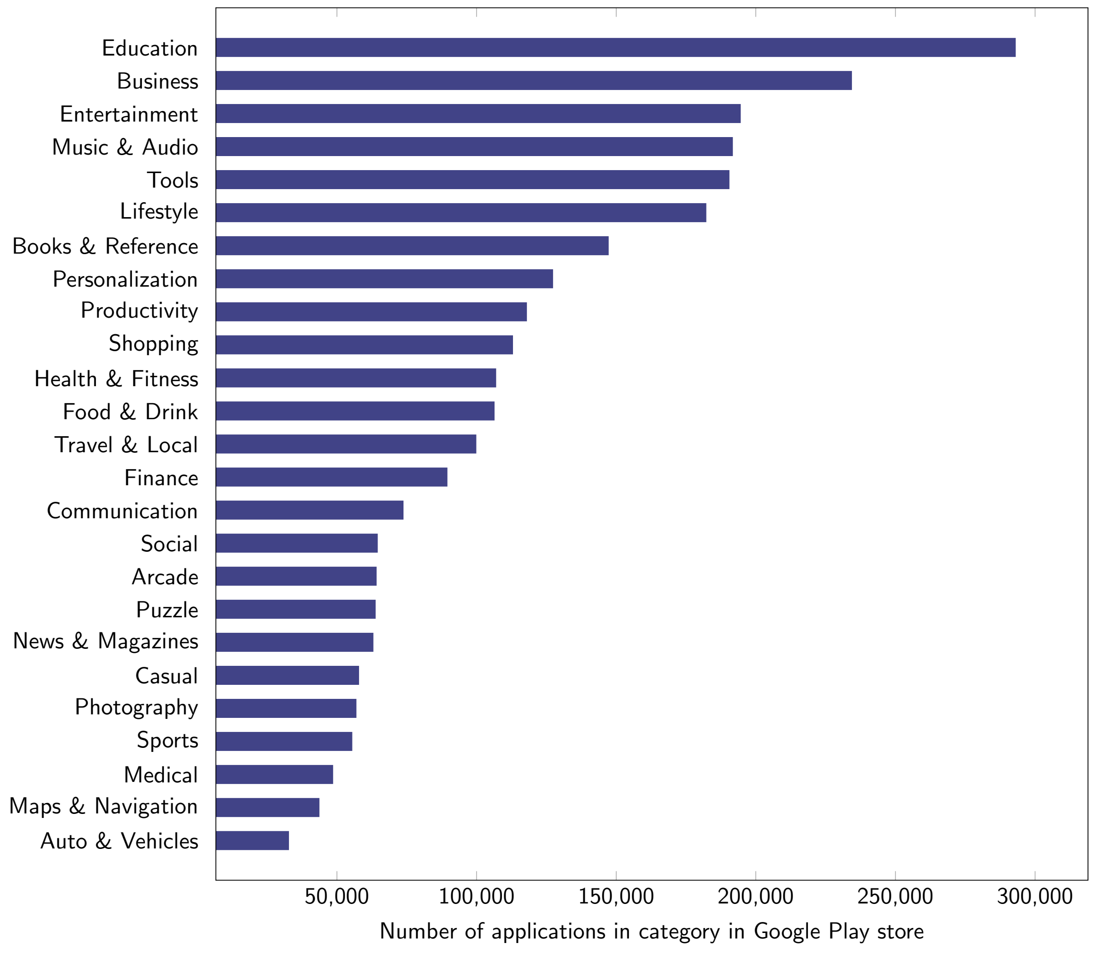


FIGURE 6.1: Breakdown of applications in the Google Play Store in the top 25 categories - data from [259].

6.2 Research opportunities

6.2.1 Scale and accelerating performance evaluation

Experiments in both Chapter 3 and 5 raised implications that the size of the DPM policy parameter space is one of the greatest barriers to static and dynamic configuration. These experiments only considered one and four applications respectively, with each application demonstrating substantially different responses to changes in DPM policy parameters. These applications do not come close to representing the full diversity of applications executed on mobile devices. In Figure 6.1, a breakdown of the top 25 categories of applications in the Google Play Store is presented. Despite there being over 3,000,000 applications already listed, over 100,000 new applications are released each month. A small number of these may account for a majority of overall downloads but it is still apparent that the surface has barely been scratched in application diversity. Each application can exhibit different performance characteristics that affect the optimal tuning of Dynamic Power Management policies.

There are a number of possibilities to scale the size of tuning experiments to both more DPM policy parameters and more applications, some of these are listed below.

1. Mobile devices could be deployed at scale for performance tuning in the same manner as mobile device labs have been deployed for compatibility and regression testing [110, 111, 112].
2. PPPx shows the potential of developing new interactive performance metrics to estimate frame rate janks and latency measurement, this could also be extended to energy and thermal models.
3. Each new DPM policy parameter to be tuned requires additional experimentation, some of the parameters tuned in Section 5.2 provided marginal or no gains, if DPM policy parameters and the range of values were selected systematically it would be more efficient [109].
4. The order for the static configuration process in Section 5.2 was derived empirically, if interference between DPM policy parameters was systematically computed then fewer benchmark repeats would be required.
5. Some applications exhibit similar performance characteristics or are based on the same underlying application engine, learning between these applications about optimal DPM policy parameter configurations could be shared to tackle the volume of applications detailed in Figure 6.1.
6. Notwithstanding privacy concerns, profiling of DPM parameter settings could be deployed on user devices to form part of the initial profiling of DPM policy parameters and their effect on application performance, this data could be combined across devices in a federated learning process.

Investment in mobile device DPM tuning equipment and technology requires a business case that will depend on further case studies quantifying the potential benefit in performance, energy and skin temperature. These case studies could demonstrate that investment in systematic tuning of mobile DPM policy parameters is more cost effective than investment in designing new computer architectures, software application tuning or other techniques currently employed to benefit performance.

6.2.2 Hierarchical Reinforcement Learning

Dynamic configuration of DPM policy parameters was compared using approaches of both offline profiling and application API communication. There remain many avenues to be explored in architectures for dynamic configuration that use these techniques as well as new approaches. Conflicts in policy have plagued the traditional model of isolated

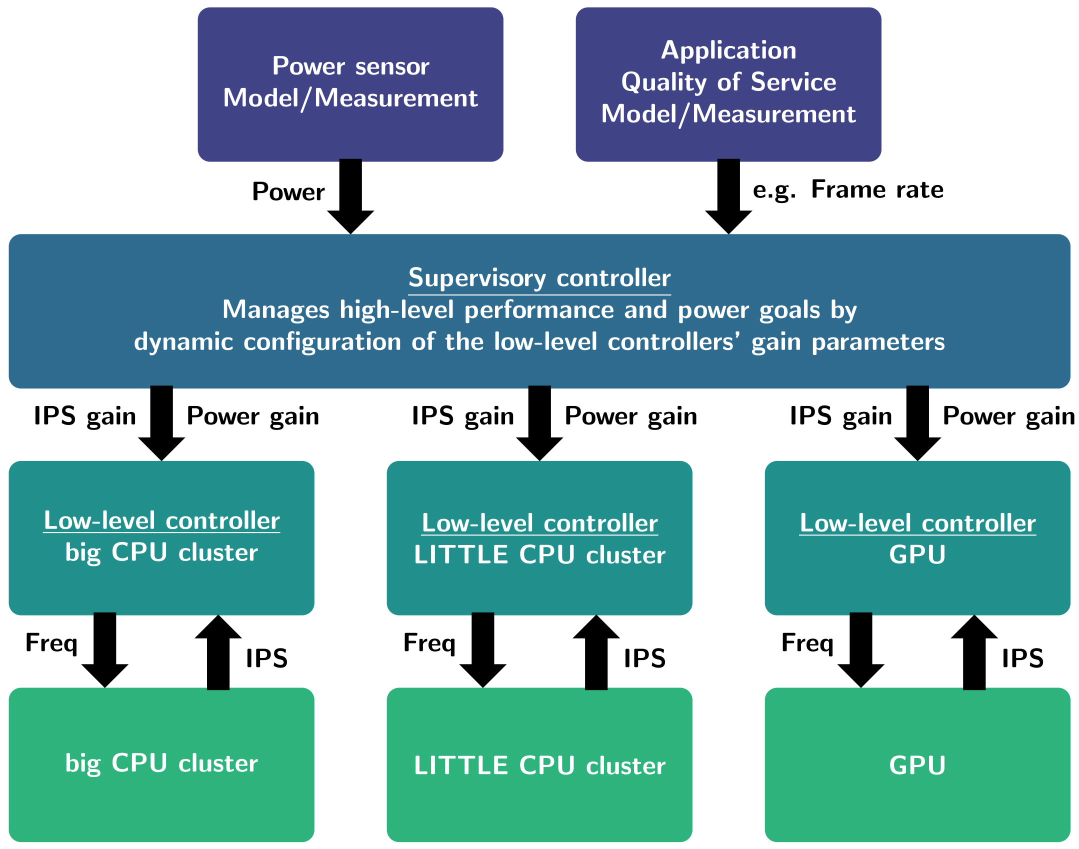


Figure 6.2: Hierarchical control system architecture for on-chip dynamic resource management [191, 193, 194].

DPM control for individual SoC subsystems such as the Linux power governors [38, 39]. Research has been undertaken into hierarchical control systems for dynamic configuration that allow the transfer of DPM policy between SoC subsystems [191, 193, 194]. These hierarchical approaches are promising in their ability to scale to more complex hardware and applications due to their localised low-latency control of the underlying DPM levers that centralised control systems are unable to provide. An example hierarchical control system architecture is shown in Figure 6.2. The supervisory controller manages high-level goals of power (temperature and energy) and application Quality of Service. Policy transfer to low-level controllers is made by dynamic configuration of gain parameters for Instructions-per-second (IPS) and power. The low-level controller uses the gains in a closed control loop to set core frequency.

State-of-the-art DPM controllers are being implemented using Reinforcement Learning (RL) [52, 196, 197, 198], these present a problem for DPM policy transfer between levels of the hierarchy. Current hierarchical control systems must retrain the low-level controllers each time the performance and power constraints imposed from above change [194]. This contrasts with non-RL solutions where gains may be adjusted continuously without disruption [191]. There is an open research question in how to incorporate DPM policy transfer in hierarchical reinforcement learning continuously, one possibility is through use of variable reward functions [260].

Bibliography

[1] J. Bantock, V. Tenentes, B. Al-Hashimi, and G. Merrett, "Online tuning of dynamic power management for efficient execution of interactive workloads," in *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED)*, 2017, pp. 1–6.

[2] Deloitte, "Global mobile consumer survey 2019: The uk cut (accessed 2020)," 2019. [Online]. Available: https://www2.deloitte.com/content/dam/Deloitte/uk/Documents/technology-media-telecommunications/deloitte-uk-plateauing-at-the-peak-the-state-of-the-smartphone.pdf

[3] Deloitte, "Global mobile consumer survey 2018: Us edition (accessed 2020)," 2018. [Online]. Available: https://www2.deloitte.com/us/en/pages/technology-media-and-telecommunications/articles/global-mobile-consumer-survey-us-edition.html

[4] G. Forman and J. Zahorjan, "The challenges of mobile computing," *Computer*, vol. 27, no. 4, pp. 38–47, 1994.

[5] M. Satyanarayanan, "Fundamental challenges in mobile computing," in *ACM Symposium on Principles of Distributed Computing*, 1996.

[6] G. Wagner and W. Maltz, "On the thermal management challenges in next generation handheld devices," in *ASME InterPACK*, 2013.

[7] S. Engineering, "Adapting mobile to a post-moores law era (accessed 2020)," 2019. [Online]. Available: https://semiengineering.com/adapting-mobile-to-a-post-moores-law-era/

[8] IDC, "Worldwide smartphone forecast update, 20192023: September 2019 (accessed 2020)," 2019. [Online]. Available: https://www.idc.com/getdoc.jsp?containerId=US45526119

[9] Gartner, "Gartner says global smartphone sales declined 2.7of 2019 (accessed 2020)," 2019. [Online]. Available: https://www.gartner.com/en/newsroom/press-releases/2019-05-28-gartner-says-global-smartphone-sales-declined-2-7--in

[10] Statista, “Global average selling price (asp) of smartphones from 2016 to 2021 (in u.s. dollars) (accessed 2020),” 2019. [Online]. Available: https://www.statista.com/statistics/788557/global-average-selling-price-smartphones/

[11] BCG, “The mobile revolution (accessed 2020),” 2015. [Online]. Available: http://image-src.bcg.com/Images/The_Mobile_Revolution_Jan_2015_tcm38-80158.pdf

[12] BCG, “Digital consumers, emerging markets, and the 4 trillion future (accessed 2020),” 2018. [Online]. Available: http://image-src.bcg.com/Images/BCG-Digital-Consumers-Emerging-Markets-and-the-4-Trillion-Future-Sep-2018_tcm38-202652.pdf

[13] W. I. P. Office, “Intangible assets and value capture in global value chains: the smartphone industry (accessed 2020),” 2017. [Online]. Available: https://www.wipo.int/edocs/pubdocs/en/wipo_pub_econstat_wp_41.pdf

[14] Bloomberg, “Apple reinvention as services company starts for real monday (accessed 2020),” 2019. [Online]. Available: https://www.bloomberg.com/news/articles/2019-03-23/apple-s-reinvention-as-a-services-company-starts-for-real-monday

[15] McKinsey, “Winning in digital ecosystems (accessed 2020),” 2018. [Online]. Available: https://www.mckinsey.com/~/media/McKinsey/BusinessFunctions/McKinseyDigital/OurInsights/DigitalMcKinseyInsightsNumber3/Digital-McKinsey-Insights-Issue-3-revised.ashx

[16] Statista, “Number of apps available in leading app stores as of 2nd quarter 2019 (accessed 2020),” 2019. [Online]. Available: https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/

[17] McKinsey, “Anatomy of a smartphone purchase (accessed 2020),” 2012. [Online]. Available: https://www.mckinsey.com/business-functions/marketing-and-sales/our-insights/infographic-anatomy-of-a-smartphone-purchase

[18] H. Chen, Y. Dai, H. Meng, Y. Chen, and T. Li, “Understanding the characteristics of mobile augmented reality applications,” in *Proceedings of International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2018, pp. 128–138.

[19] AppAnnie, “2019 in mobile: 5 things you need to know (accessed 2020),” 2019. [Online]. Available: https://www.appannie.com/en/insights/market-data/2019-in-mobile-5-things-to-know/

[20] Gartner, “Gartner highlights 10 uses for ai-powered smartphones (accessed 2020),” 2018. [Online]. Available: https://www.gartner.com/en/newsroom/press-releases/2018-03-20-gartner-highlights-10-uses-for-ai-powered-smartphones

[21] G. Sachs, “Virtual and augmented reality: Understanding the race for the next computing platform (accessed 2020),” 2016. [Online]. Available: https://www.goldmansachs.com/insights/pages/technology-driving-innovation-folder/virtual-and-augmented-reality/report.pdf

[22] Y. Leng, C. Chen, Q. Sun, J. Huang, and Y. Zhu, “Energy-efficient video processing for virtual reality,” in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2019, pp. 91–103.

[23] A. Pathania, A. Irimiea, A. Prakash, and T. Mitra, “Power-performance modelling of mobile gaming workloads on heterogeneous mpsocs,” in *Proceedings of the Design Automation Conference (DAC)*, 2015.

[24] C. Wu, D. Brooks, K. Chen, D. Chen, S. Choudhury, M. Dukhan, K. Hazelwood, E. Isaac, Y. Jia, B. Jia, T. Leyvand, H. Lu, Y. Lu, L. Qiao, B. Reagen, J. Spisak, F. Sun, A. Tulloch, P. Vajda, X. Wang, Y. Wang, B. Wasti, Y. Wu, R. Xian, S. Yoo, and P. Zhang, “Machine learning at facebook: Understanding inference at the edge,” in *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, 2019, pp. 331–344.

[25] N. Lane, S. Bhattacharya, A. Mathur, P. Georgiev, C. Forlivesi, and F. Kawsar, “Squeezing deep learning into mobile and embedded devices,” *IEEE Pervasive Computing*, vol. 16, no. 3, pp. 82–88, 2017.

[26] L. Huynh, Y. Lee, and R. Balan, “Deepmon: Mobile gpu-based deep learning framework for continuous vision applications,” in *Proceedings of the International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2017, pp. 82–95.

[27] C. Hsieh, A. Sani, and N. Dutt, “The case for exploiting underutilized resources in heterogeneous mobile architecture,” in *Proceedings of the Design Automation & Test in Europe Conference (DATE)*, 2019, pp. 1265–1268.

[28] Anandtech, “The samsung galaxy s10+ snapdragon exynos review: Almost perfect, yet so flawed (accessed 2020),” 2019. [Online]. Available: https://www.anandtech.com/show/14072/the-samsung-galaxy-s10plus-review

[29] Anandtech, “The iphone xs and xs max review: Unveiling the silicon secrets (accessed 2020),” 2018. [Online]. Available: https://www.anandtech.com/show/13392/the-iphone-xs-xs-max-review-unveiling-the-silicon-secrets/7

[30] Arstechnica, “The future of mobile cpus, part 1: Todays fork in the road (accessed 2020),” 2013. [Online]. Available: https://arstechnica.com/gadgets/2013/02/the-future-of-mobile-cpus-part-1-todays-fork-in-the-road/

[31] M. Halpern, Y. Zhu, and V. Reddi, "Mobile cpu's rise to power: Quantifying the impact of generational mobile cpu design trends on performance, energy, and user satisfaction," in *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, 2016, pp. 64–76.

[32] Anandtech, "The apple iphone 11, 11 pro 11 pro max review: Performance, battery, camera elevated (accessed 2020)," 2019. [Online]. Available: https://www.anandtech.com/show/14892/the-apple-iphone-11-pro-and-max-review/6

[33] L. Benini, A. Bogliolo, and G. D. Micheli, "A survey of design techniques for system-level dynamic power management," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 8, no. 3, pp. 299–316, 2000.

[34] J. Yang, X. Zhou, M. Chrobak, Y. Zhang, and L. Jin, "Dynamic thermal management through task scheduling," in *Proceedings of the International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2008, pp. 191–201.

[35] O. Sahin and A. Coskun, "On the impacts of greedy thermal management in mobile devices," *IEEE Embedded Systems Letters*, vol. 7, no. 2, pp. 55–58, 2015.

[36] L. Benini, A. Bogliolo, S. Cavallucci, and B. Ricco, "Monitoring system activity for os-directed dynamic power management," in *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED)*, 1998, pp. 185–190.

[37] C. Wong, I. Tan, R. Kumari, J. Lam, and W. Fun, "Fairness and interactive performance of o(1) and cfs linux kernel schedulers," in *Proceedings of the International Symposium on Information Technology*, 2008.

[38] D. Brodowski, "cpufreq: sysfs interface (accessed 2020)," 2003. [Online]. Available: https://lwn.net/Articles/19798/

[39] V. Pallipadi, "Introducing cpuidle: core cpuidle infrastructure (accessed 2020)," 2007. [Online]. Available: https://lwn.net/Articles/221791/

[40] M. Chan, "cpufreq: interactive: New 'interactive' governor (accessed 2020)," 2015. [Online]. Available: https://lwn.net/Articles/662209/

[41] D. Snowdon, E. L. Sueur, S. Petters, and G. Heiser, "Koala a platform for os-level power management," in *Proceedings of the European Conference on Computer Systems (EuroSys)*, 2009, pp. 289–302.

[42] A. Gutierrez, R. Dreslinski, T. Wenisch, T. Mudge, A. Saidi, C. Emmons, and N. Paver, "Full-system analysis and characterization of interactive smartphone applications," in *Proceedings of the International Symposium on Workload Characterization (IISWC)*, 2011, pp. 81–90.

[43] C. Gao, A. Gutierrez, M. Rajan, R. Dreslinski, T. Mudge, and C. Wu, "A study of mobile device utilization," in *Proceedings of International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2015.

[44] F. Paterna and T. Rosing, "Modeling and mitigation of extra-soc thermal coupling effects and heat transfer variations in mobile devices," in *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*, 2015, pp. 831–838.

[45] F. Paterna, J. Zanotelli, and T. Rosing, "Ambient variation-tolerant and inter components aware thermal management for mobile system on chips," in *Proceedings of the Design Automation & Test in Europe Conference (DATE)*, 2014.

[46] S. Li, J. Ahn, R. Strong, J. Brockman, D. Tullsen, and N. Jouppi, "Mcpat: an integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2009, pp. 469–480.

[47] (2009) Reduce linux power consumption (accessed 2020). [Online]. Available: https://www.ibm.com/developerworks/library/l-cpufreq-3/index.html

[48] SUSE, "Tuning the task scheduler (accessed 2020)." [Online]. Available: https://doc.opensuse.org/documentation/leap/archive/42.1/tuning/html/book.sle.tuning/book.sle.tuning.html

[49] G. Mariani, P. Avasare, G. Vanmeerbeeck, C. Ykman-Couvreur, G. Palermo, C. Silvano, and V. Zaccaria, "An industrial design space exploration framework for supporting run-time resource management on multi-core systems," in *Proceedings of the Design Automation & Test in Europe Conference (DATE)*, 2010, pp. 196–201.

[50] L. Benini, A. Bogliolo, G. Paleologo, and G. D. Micheli, "Policy optimization for dynamic power management," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 6, pp. 813–833, 1999.

[51] ARM, "Energy aware scheduling (eas) (accessed 2020)," 2019. [Online]. Available: https://developer.arm.com/tools-and-software/open-source-software/linux-kernel/energy-aware-scheduling

[52] Y. Tan, W. Liu, and Q. Qiu, "Adaptive power management using reinforcement learning," in *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*, 2009, pp. 461–467.

[53] N. Liu, Z. Li, J. Xu, Z. Xu, S. Lin, Q. Qiu, J. Tang, and Y. Wang, "A hierarchical framework of cloud resource allocation and power management using deep reinforcement learning," in *Proceedings of the International Conference on Distributed Computing Systems (ICDCS)*, 2017, pp. 372–382.

[54] J. Hennessy and D. Patterson, *Computer Architecture, Fifth Edition: A Quantitative Approach*, 5th ed. Morgan Kaufmann Publishers Inc., 2011.

[55] S. Kaxiras and M. Martonosi, “Computer architecture techniques for power-efficiency,” *Synthesis Lectures on Computer Architecture*, vol. 3, no. 1, pp. 1–207, 2008.

[56] O. Azizi, A. Mahesri, B. Lee, S. Patel, and M. Horowitz, “Energy-performance tradeoffs in processor architecture and circuit design: A marginal cost analysis,” in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2010, pp. 26–36.

[57] M. Rostami, F. Koushanfar, J. Rajendran, and R. Karri, “Hardware security: threat models and metrics,” in *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*, 2013, pp. 819–823.

[58] C. Constantinescu, “Trends and challenges in vlsi circuit reliability,” *IEEE Micro*, vol. 23, no. 4, pp. 14–19, 2003.

[59] N. Abu-Ghazaleh, D. Ponomarev, and D. Evtyushkin, “How the spectre and meltdown hacks really worked,” *IEEE Spectrum*, vol. 56, no. 3, pp. 42–49, 2019.

[60] O. Mutlu, “The rowhammer problem and other issues we may face as memory becomes denser,” in *Proceedings of the Design Automation & Test in Europe Conference (DATE)*, 2017, pp. 1116–1121.

[61] International symposium on high-performance computer architecture (accessed 2020). [Online]. Available: https://www.hpca-conf.org

[62] International symposium on computer architecture (accessed 2020). [Online]. Available: https://iscaconf.org

[63] International symposium on microarchitecture (accessed 2020). [Online]. Available: https://www.microarch.org

[64] D. He, S. Chan, and M. Guizani, “Mobile application security: malware threats and defenses,” *IEEE Wireless Communications*, vol. 22, no. 1, pp. 138–144, 2015.

[65] A. Althoff, J. McMahan, L. Vega, S. Davidson, T. Sherwood, M. Taylor, and R. Kastner, “Hiding intermittent information leakage with architectural support for blinking,” in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2018, pp. 638–649.

[66] S. Quadir, J. Chen, D. Forte, N. Asadizanjani, S. Shahbazmohamadi, L. Wang, J. Chandy, and M. Tehranipoor, “A survey on chip to system reverse engineering,” *ACM Journal of Emerging Technologies in Computing Systems*, vol. 13, no. 1, pp. 6:1–6:34, 2016.

[67] Economist, “Chip wars: American, china and silicon supremacy,” 2018.

[68] J. Suckling and J. Lee, “Redefining scope: the true environmental impact of smartphones?” *The International Journal of Life Cycle Assessment*, vol. 20, pp. 1181–1196, 2015.

[69] X. Li, P. Ortiz, J. Browne, D. Franklin, J. Oliver, R. Geyer, Y. Zhou, and F. Chong, “Smartphone evolution and reuse: Establishing a more sustainable model,” in *International Conference on Parallel Processing Workshops*, 2010, pp. 476–484.

[70] L. Belkhir and A. Elmeligi, “Assessing ict global emissions footprint: Trends to 2040 and recommendations,” *Journal of Cleaner Production*, vol. 177, pp. 448–463, 2018.

[71] G. Alliance, “A circular economy for smart devices (accessed 2020),” 2015. [Online]. Available: https://www.green-alliance.org.uk/resources/Acirculareconomyforsmartdevices.pdf

[72] BCG, “When chip makers look through the value lens (accessed 2020),” 2017. [Online]. Available: https://www.bcg.com/en-gb/publications/2017/when-chip-makers-look-through-value-lens.aspx

[73] Big trouble at 3nm (accessed 2020). [Online]. Available: https://semiengineering.com/big-trouble-at-3nm/

[74] DARPA, “Three dimensional monolithic system-on-a-chip (3dsoc) (accessed 2020),” 2018. [Online]. Available: https://www.darpa.mil/program/three-dimensional-monolithic-system-on-a-chip

[75] S. Engineering, “Whats next in advanced packaging (accessed 2020),” 2019. [Online]. Available: https://semiengineering.com/whats-next-in-advanced-packaging/

[76] S. Engineering, “Advanced packaging options increase (accessed 2020),” 2019. [Online]. Available: https://semiengineering.com/advanced-packaging-options-increase/

[77] C. Tseng, C. Liu, C. Wu, and D. Yu, “Info (wafer level integrated fan-out) technology,” in *Proceedings of the Electronic Components and Technology Conference (ECTC)*, 2016.

[78] C. Hu, M. Chen, W. Chiou, and D. Yu, “3d multi-chip integration with system on integrated chips (soic),” in *Symposium on VLSI Technology*, 2019, pp. T20–T21.

[79] S. Hou, W. Chen, C. Hu, C. Chiu, K. Ting, T. Lin, W. Wei, W. Chiou, V. Lin, V. Chang, C. Wang, C. Wu, and D. Yu, “Wafer-level integration of an advanced logic-memory system through the second-generation cowos technology,” *IEEE Transactions on Electron Devices*, vol. 65, no. 10, pp. 4071–4077, 2017.

[80] M. Lin, T. Huang, C. Tsai, K. Tam, C. Hsieh, T. Chen, W. Huang, J. Hu, Y. Chen, S. Goel, C. Fu, S. Rusu, C. Li, S. Yang, M. Wong, S. Yang, and F. Lee, "A 7nm 4ghz arm-core-based cowos chiplet design for high performance computing," in *Symposium on VLSI Circuits*, 2019, pp. C28–C29.

[81] Anandtech, "Amd rome second generation epyc review: 2x 64-core benchmarked (accessed 2020)," 2019. [Online]. Available: https://www.anandtech.com/show/14694/amd-rome-epyc-2nd-gen/2

[82] Anandtech, "Intel's interconnected future: Combining chiplets, emib, and foveros (accessed 2020)," 2019. [Online]. Available: https://www.anandtech.com/show/14211/intels-interconnected-future-chipslets-emib-foveros

[83] A. Arunkumar, E. Bolotin, B. Cho, U. Milic, E. Ebrahimi, O. Villa, A. Jaleel, C. Wu, and D. Nellans, "Mcm-gpu: Multi-chip-module gpus for continued performance scalability," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2017, pp. 320–332.

[84] DARPA, "Common heterogeneous integration and ip reuse strategies (chips) (accessed 2020)," 2017. [Online]. Available: https://www.darpa.mil/program/common-heterogeneous-integration-and-ip-reuse-strategies

[85] M. Papamichael, P. Milder, and J. Hoe, "Nautilus: Fast automated ip design space search using guided genetic algorithms," in *Proceedings of the Design Automation Conference (DAC)*, 2015.

[86] L. Liu and X. Luo, "The reconfigurable ip modules and design," in *Proceedings of the International Conference on Electronic & Mechanical Engineering and Information Technology (EMEIT)*, 2011, pp. 1324–1327.

[87] T. Ristimaki and J. Nurmi, "Reconfigurable ip blocks: a survey [soc]," in *Proceedings of the International Symposium on System-on-Chip (SoC)*, 2004, pp. 117–122.

[88] A. Danowitz, K. Kelley, J. Mao, J. Stevenson, and M. Horowitz, "Cpu db: Recording microprocessor history," *Communications of the ACM*, vol. 55, no. 4, pp. 55–63, 2012.

[89] T. Bjerregaard and S. Mahadevan, "A survey of research and practices of network-on-chip," *ACM Computing Surveys*, vol. 38, no. 1, 2006.

[90] E. Ipek, O. Mutlu, J. Martinez, and R. Caruana, "Self-optimizing memory controllers: A reinforcement learning approach," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2008, pp. 39–50.

[91] X. Wang, M. Yang, Y. Jiang, P. Liu, M. Daneshtalab, M. Palesi, and T. Mak, "On self-tuning networks-on-chip for dynamic network-flow dominance adaptation,"

ACM Transactions on Embedded Computing Systems, vol. 13, no. 2s, pp. 73:1–73:21, 2014.

[92] "Performance management (accessed 2020)." [Online]. Available: https://source.android.com/devices/tech/power/performance

[93] S. Kang, H. Choi, S. Park, C. Park, J. Lee, U. Lee, and S. Lee, "Fire in your hands: Understanding thermal behavior of smartphones," in *Proceedings of the International Conference on Mobile Computing and Networking (MobiCom)*, 2019, pp. 13:1–13:16.

[94] B. Egilmez, G. Memik, S. Ogrenci-Memik, and O. Ergin, "User-specific skin temperature-aware dvfs for smartphones," in *Proceedings of the Design Automation & Test in Europe Conference (DATE)*, 2015, pp. 1217–1220.

[95] K. Yan, X. Zhang, J. Tan, and X. Fu, "Redefining qos and customizing the power management policy to satisfy individual mobile users," in *Proceedings of International Symposium on Microarchitecture (MICRO)*, 2016.

[96] G. Bhat, S. Gumussoy, and U. Ogras, "Power and thermal analysis of commercial mobile platforms: Experiments and case studies," in *Proceedings of the Design Automation & Test in Europe Conference (DATE)*, 2019, pp. 144–149.

[97] A. Prakash, H. Amrouch, M. Shafique, T. Mitra, and J. Henkel, "Improving mobile gaming performance through cooperative cpu-gpu thermal management," in *Proceedings of the Design Automation Conference (DAC)*, 2016.

[98] K. Yan, X. Zhang, and X. Fu, "Characterizing, modeling, and improving the qoe of mobile devices with low battery level," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2015, pp. 713–724.

[99] P. Kong, L. Li, J. Gao, K. Liu, T. Bissyande, and J. Klein, "Automated testing of android apps: A systematic literature review," *IEEE Transactions on Reliability*, vol. 68, no. 1, pp. 45–66, 2019.

[100] C. Bienia, "Benchmarking modern multiprocessors," Ph.D. dissertation, Princeton University, 2011.

[101] J. Henning, "Spec cpu2006 benchmark descriptions," *SIGARCH Computer Architecture News*, vol. 34, no. 4, 2006.

[102] V. Reddi, H. Yoon, and A. Knies, "Two billion devices and counting," *IEEE Micro*, vol. 38, no. 1, pp. 6–21, 2018.

[103] Y. Huang, Z. Zha, M. Chen, and L. Zhang, "Moby: A mobile benchmark suite for architectural simulators," in *Proceedings of the International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2014, pp. 45–54.

[104] McKinsey, "How high-tech suppliers are responding to the hyperscaler opportunity (accessed 2020)," 2018. [Online]. Available: https://www.mckinsey.com/industries/technology-media-and-telecommunications/our-insights/how-high-tech-suppliers-are-responding-to-the-hyperscaler-opportunity

[105] Y. Zhu, J. Liu, M. Guo, Y. Bao, W. Ma, Z. Liu, K. Song, and Y. Yang, "Bestconfig: Tapping the performance potential of systems via automatic configuration tuning," in *Proceedings of the Symposium on Cloud Computing (SoCC)*, 2017, pp. 338–350.

[106] D. Aken, A. Pavlo, G. Gordon, and B. Zhang, "Automatic database management system tuning through large-scale machine learning," in *Proceedings of the International Conference on Management of Data (SIGMOD)*, 2017, pp. 1009–1024.

[107] M. Bilal and M. Canini, "Towards automatic parameter tuning of stream processing systems," in *Proceedings of the Symposium on Cloud Computing (SoCC)*, 2017, pp. 189–200.

[108] T. Xu, L. Jin, X. F. an Y. Zhou, S. Pasupathy, and R. Talwadker, "Hey, you have given me too many knobs! understanding and dealing with over-designed configuration in system software," in *Proceedings of the Symposium on the Foundations of Software Engineering (FSE)*, 2015.

[109] S. Wang, C. Li, H. Hoffman, S. Lu, W. Sentosa, and A. Kistijantoro, "Understanding and auto-adjusting performance-sensitive configurations," in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2018, pp. 154–168.

[110] "Managing resources for large-scale testing (accessed 2020)," 2016. [Online]. Available: https://engineering.fb.com/android/managing-resources-for-large-scale-testing/

[111] "Mobile performance: Tooling infrastructure at facebook (accessed 2020)," 2015. [Online]. Available: https://engineering.fb.com/developer-tools/mobile-performance-tooling-infrastructure-at-facebook/

[112] "Mobilelab: Highly accurate testing to prevent mobile performance regressions (accessed 2020)," 2018. [Online]. Available: https://engineering.fb.com/android/mobilelab/

[113] Y. Endo, Z. Wang, J. Chen, and M. Seltzer, "Using latency to evaluate interactive system performance," in *Proceedings of the Symposium on Operating Systems Design and Implementation (OSDI)*, 1996, pp. 185–199.

[114] J. Bantock, B. Al-Hashimi, and G. Merrett, "Mitigating interactive performance degradation from mobile device thermal throttling," *IEEE Embedded System Letters*, 2020.

[115] A. Boroumand, S. Ghose, Y. Kim, R. Ausavarungnirun, E. Shiu, R. Thakur, D. Kim, A. Kuusela, A. Knies, P. Ranganathan, and O. Mutlu, "Google workloads for consumer devices: Mitigating data movement bottlenecks," in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2018, pp. 316–331.

[116] X. Liu, T. Chen, F. Qian, Z. Guo, F. Lin, X. Wang, and K. Chen, "Characterizing smartwatch usage in the wild," in *Proceedings of the International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2017, pp. 385–398.

[117] P. Phothilimthana, J. Ansel, J. Ragan-Kelley, and S. Amarasinghe, "Portable performance on heterogeneous architectures," in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2013, pp. 431–444.

[118] N. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, and F. Kawsar, "An early resource characterization of deep learning on wearables, smartphones and internet-of-things devices," in *Proceedings of the International Workshop on Internet of Things towards Applications (IoT-App)*, 2015, pp. 7–12.

[119] A. Shye, B. Scholbrock, and G. Memik, "Into the wild: Studying real user activity patterns to guide power optimizations for mobile architectures," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2009, pp. 168–178.

[120] A. Carroll and G. Heiser, "An analysis of power consumption in a smartphone," in *Proceedings of USENIX Annual Technical Conference*, 2010.

[121] A. Nandugudi, A. Maiti, T. Ki, F. Bulut, M. Demirbas, T. Kosar, C. Qiao, S. Ko, and G. Challen, "Phonelab: A large programmable smartphone testbed," in *Proceedings of the International Workshop on Sensing and Big Data Mining (SENSEMINE)*, 2013, pp. 4:1–4:6.

[122] M. Dasari, S. Vargas, A. Bhattacharya, A. Balasubramanian, S. Das, and M. Ferdman, "Impact of device performance on mobile internet qoe," in *Proceedings of the International Measurement Conference (IMC)*, 2018.

[123] R. Liu and F. Lin, "Understanding the characteristics of android wear os," in *Proceedings of the International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2016, pp. 151–164.

[124] J. Whitehouse, Q. Wu, S. Song, E. John, A. Gerstlauer, and L. John, "A study of core utilization and residency in heterogeneous smart phone architectures," in *Proceedings of the International Conference on Performance Engineering (ICPE)*, 2019, pp. 67–78.

[125] P. Rengasamy, H. Zhang, N. Nachiappan, S. Zhao, A. Sivasubramanian, M. Kandemir, and C. Das, "Characterizing diverse handheld apps for customized hardware acceleration," in *Proceedings of International Symposium on Workload Characterization (IISWC)*, 2017, pp. 187–196.

[126] Y. Zhu and V. Reddi, "Optimizing general-purpose cpus for energy-efficient mobile web computing," *ACM Transactions on Computer Systems*, 2017.

[127] X. Ma, Z. Deng, M. Dong, and L. Zhong, "Characterizing the performance and power consumption of 3d mobile games," *IEEE Computer*, vol. 46, no. 4, pp. 76–82, 2013.

[128] N. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, L. Jiao, L. Qendro, and F. Kawsar, "Deepx: A software accelerator for low-power deep learning inference on mobile devices," in *Proceedings of the 15th International Conference on Information Processing in Sensor Networks (IPSN)*, 2016, pp. 23:1–23:12.

[129] H. Falaki, R. Mahajan, S. Kandula, D. Lymberopoulos, R. Govindan, and D. Estrin, "Diversity in smartphone usage," in *Proceedings of the International Conference on Mobile Systems Applications and Services (MobiSys)*, 2010, pp. 179–194.

[130] S. Ickin, K. Wac, M. Fiedler, L. Janowski, J. Hong, and A. Dey, "Factors influencing quality of experience of commonly used mobile applications," *IEEE Communications Magazine*, vol. 50, no. 4, pp. 48–56, 2012.

[131] X. Chen, A. Jindal, N. Ding, Y. Hu, M. Gupta, and R. Vannithamby, "Smartphone background activities in the wild: Origin, energy drain, and optimization," in *Proceedings of the International Conference on Mobile Computing and Networking (MobiCom)*, 2015, pp. 40–52.

[132] X. Chen, N. Ding, A. Jindal, Y. Hu, M. Gupta, and R. Vannithamby, "Smartphone energy drain in the wild: Analysis and implications," in *Proceedings of the International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, 2015, pp. 151–164.

[133] V. Chiriac, S. Molloy, J. Anderson, and K. Goodson, "A figure of merit for mobile device thermal management," in *Proceedings of the Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems (ITherm)*, 2016, pp. 1393–1397.

[134] H. Hsiao, H. Chiou, and Y. Lee, "Multi-angle bended heat pipe design using x-architecture routing with dynamic thermal weight on mobile devices," in *Proceedings of the Asia and South Pacific Design Automation Conference (ASPDAC)*, 2019, pp. 70–75.

[135] AnandTech, “The nubia red magic 3 review: A 90hz gaming phone with active cooling (accessed 2020),” 2019. [Online]. Available: https://www.anandtech.com/show/14717/the-red-magic-3-review

[136] AnandTech, “Asus rog phone ii ultimate edition: 120 hz, 12 gb / 1 tb with 6000 mah (accessed 2020),” 2019. [Online]. Available: https://www.anandtech.com/show/14834/asus-rog-phone-ii-ultimate-edition

[137] The snapdragon 865 performance preview: Setting the stage for flagship android 2020 (accessed 2020). [Online]. Available: https://www.anandtech.com/show/15207/the-snapdragon-865-performance-preview-setting-the-stage-for-flagship-android-2020

[138] The huawei mate 30 pro review: Top hardware without google? (accessed 2020). [Online]. Available: https://www.anandtech.com/show/15099/the-huawei-mate-30-pro-review-top-hardware-without-google/2

[139] The samsung galaxy s20+, s20 ultra exynos snapdragon review: Megalomania devices (accessed 2020). [Online]. Available: https://www.anandtech.com/show/15603/the-samsung-galaxy-s20-s20-ultra-exynos-snapdragon-review-megalomania-devices/4

[140] Y. Choi, S. Park, and H. Cha, “Graphics-aware power governing for mobile devices,” in *Proceedings of the International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2019, pp. 469–481.

[141] J. Park, N. Dutt, K. Hoyeonjiki, and S. Lim, “Hicap: Hierarchical fsm-based dynamic integrated cpu-gpu frequency capping governor for energy-efficient mobile gaming,” in *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED)*, 2016, pp. 218–223.

[142] H. Esmaeilzadeh, E. Blem, R. Amant, K. Sankaralingam, and D. Burger, “Dark silicon and the end of multicore scaling,” in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2011, pp. 365–376.

[143] M. Shafique, S. Garg, T. Mitra, S. Parameswaran, and J. Henkel, “Dark silicon as a challenge for hardware/software co-design,” in *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2014.

[144] H. Zhang, P. Rengasamy, S. Zhao, N. Nachiappan, A. Sivasubramanian, M. Kandemir, R. Iyer, and C. Das, “Race-to-sleep + content caching + display caching: A recipe for energy-efficient video streaming on handhelds,” in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2017, pp. 517–531.

[145] D. Kim, C. Imes, and H. Hoffman, "Racing and pacing to idle: Theoretical and empirical analysis of energy optimization heuristics," in *Proceedings of the International Conference on Cyber-Physical Systems, Networks and Applications*, 2015, pp. 78–85.

[146] Y. Kim, M. Kim, J. Kim, and S. Chung, "M-dtm: Migration-based dynamic thermal management for heterogeneous mobile multi-core processors," in *Proceedings of the Design Automation & Test in Europe Conference (DATE)*, 2015, pp. 1533–1538.

[147] X. Li, G. Chen, and W. Wen, "Energy-efficient execution for repetitive app usages on big.little architectures," in *Proceedings of the Design Automation Conference (DAC)*, 2017.

[148] W. Seo, D. Im, J. Choi, and J. Huh, "Big or little: A study of mobile interactive applications on an asymmetric multi-core platform," in *Proceedings of the International Symposium on Workload Characterization (IISWC)*, 2015.

[149] M. Hill and V. Reddi, "Gables: A roofline model for mobile socs," in *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, 2019, pp. 317–330.

[150] L. Ravindranath, J. Padhye, R. Mahajan, and H. Balakrishnan, "Timecard: Controlling user-perceived delays in server-based mobile applications," in *Proceedings of the Symposium on Operating Systems Principles (SOSP)*, 2013, pp. 85–100.

[151] W. Yuan and K. Nahrstedt, "Energy-efficient cpu scheduling for multimedia applications," *ACM Transactions on Computer Systems*, vol. 24, no. 3, pp. 292–331, 2006.

[152] Y. Feng and Y. Zhu, "Pes: Proactive event scheduling for responsive and energy-efficient mobile web computing," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2019, pp. 66–78.

[153] G. Singla, G. Kaur, A. Unver, and U. Ogras, "Predictive dynamic thermal and power management for heterogeneous mobile platforms," in *Proceedings of the Design Automation & Test in Europe Conference (DATE)*, 2015, pp. 960–965.

[154] O. Sahin and A. Coskun, "Qscale: Thermally-efficient qos management on heterogeneous mobile platforms," in *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*, 2016, pp. 125:1–125:8.

[155] O. Sahin, P. Varghese, and A. Coskun, "Just enough is more: Achieving sustainable performance in mobile devices under thermal limitations," in *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*, 2015, pp. 839–846.

[156] D. Lo, T. Song, and G. Suh, "Prediction-guided performance-energy trade-off for interactive applications," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2015, pp. 508–520.

[157] J. Lorch and A. Smith, "Using user interface event information in dynamic voltage scaling algorithms," in *Proceedings of the International Symposium on Modeling, Analysis and Simulation of Computer Telecommunications Systems (MASCOTS)*, 2003, pp. 46–55.

[158] K. Flautner and T. Mudge, "Vertigo: Automatic performance-setting for linux," in *Proceedings of the Symposium on Operating Systems Design and Implementation (OSDI)*, 2002, pp. 105–116.

[159] X. Li, G. Yan, Y. Han, and X. Li, "Smartcap: User experience-oriented power adaptation for smartphones application processor," in *Proceedings of the Design Automation & Test in Europe Conference (DATE)*, 2013.

[160] L. Li, J. Wang, X. Wang, H. Ye, and Z. Hu, "Sceneman: Bridging mobile apps with system energy manager via scenario notification," in *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED)*, 2017.

[161] N. Nachiappan, P. Yedlapalli, N. Soundararajan, A. Sivasubramaniam, M. Kandemir, R. Iyer, and C. Das, "Domain knowledge based energy management in handhelds," in *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, 2015.

[162] B. Dietich, N. Peters, S. Park, and S. Chakraborty, "Estimating the limits of cpu power management for mobile games," in *Proceedings of the International Conference on Computer Design (ICCD)*, 2017.

[163] Y. Shao, B. Reagen, G. Wei, and D. Brooks, "The aladdin approach to accelerator design and modeling," *IEEE Micro*, vol. 35, no. 3, pp. 58–70, 2015.

[164] A. Papathanasiou and M. Scott, "Energy efficient prefetching and caching," in *Proceedings of the USENIX Annual Technical Conference*, 2004.

[165] J. Lee, K. Lee, E. Jeong, J. Jo, and N. Shroff, "Context-aware application scheduling in mobile systems: What will users do and not do next?" in *Proceedings of the International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp)*, 2016, pp. 1235–1246.

[166] C. Shin, J. Hong, and A. Dey, "Understanding and prediction of mobile application usage for smart phones," in *Proceedings of the Conference on Ubiquitous Computing*, 2012, pp. 173–182.

[167] J. Flinn and M. Satyanarayanan, "Managing battery lifetime with energy-aware adaptation," *ACM Transactions on Computer Systems*, vol. 22, no. 2, 2004.

[168] P. Mercati, T. Rosing, V. Hanumaiah, J. Kulkarni, and S. Bloch, "User-centric joint power and thermal management for smartphones," in *Proceedings of the International Conference on Mobile Computing, Applications and Services (MobiCASE)*, 2014, pp. 98–105.

[169] K. Saravanan, D. Ansorregui, and J. Bantock, "Setting operating system parameters based on workload types," United Kingdom Patent Application GB2 569 109A, 11 30, 2017.

[170] R. Schone and D. Hackenberg, "On-line analysis of hardware performance events for workload characterization and processor frequency scaling decisions," in *Proceedings of the International Conference on Performance Engineering (ICPE)*, 2011, pp. 481–486.

[171] Q. Xie, J. Kim, Y. Wang, D. Shin, N. Chang, and M. 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)*, 2013, pp. 242–247.

[172] S. Lee, R. Ha, and H. Cha, "Click sequence prediction in android mobile applications," *IEEE Transactions on Human-Machine Systems*, vol. 49, no. 3, pp. 278–289, 2019.

[173] P. Holleis, F. Otto, H. Hussmann, and A. Schmidt, "Keystroke-level model for advanced mobile phone interaction," in *Proceedings of the Conference on Human Factors in Computing Systems (CHI)*, 2007, pp. 1505–1514.

[174] Y. Kwon, S. Lee, H. Yi, D. Kwon, S. Yang, B. Chun, L. Huang, P. Maniatis, M. Naik, and Y. Paek, "Mantis: Automatic performance prediction for smartphone applications," in *Proceedings of the USENIX Annual Technical Conference*, 2013.

[175] B. Chen, Y. Liu, and W. Le, "Generating performance distributions via probabilistic symbolic execution," in *Proceedings of the International Conference on Software Engineering*, 2016, pp. 49–60.

[176] B. Gaudette, C. Wu, and S. Vrudhula, "Improving smartphone user experience by balancing performance and energy with probabilistic qos guarantee," in *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, 2016, pp. 52–63.

[177] Y. Lin, E. Chu, E. Chang, and Y. Lai, "Smoothed graphic user interaction on smartphones with motion prediction," *IEEE Transactions on Systems, Man and Cybernetics*, 2017.

[178] Y. Zhu, M. Halpern, and V. Reddi, "Event-based scheduling for energy-efficient qos (eqos) in mobile web applications," in *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, 2015.

[179] Android - graphics (accessed 2020). [Online]. Available: https://source.android.com/devices/graphics

[180] L. Zhang, D. Bild, R. Dick, Z. Mao, and P. Dinda, “Panappticon: Event-based tracing to measure mobile application and platform performance,” in *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2013.

[181] S. Kim, H. Kim, J. Hwang, J. Lee, and E. Seo, “An event-driven power management scheme for mobile consumer electronics,” *IEEE Transactions on Consumer Electronics*, vol. 59, no. 1, pp. 259–266, 2013.

[182] E. Rotem, A. Naveh, A. Ananthakrishnan, E. Weissman, and D. Rajwan, “Power-management architecture of the intel microarchitecture code-named sandy bridge,” *IEEE Micro*, vol. 32, no. 2, pp. 20–27, 2012.

[183] S. Pagani, H. Khdr, W. Munawar, J. Chen, M. Shafique, M. Li, and J. Henkel, “Tsp: Thermal safe power - efficient power budgeting for many-core systems in dark silicon,” in *Proceedings of the International Conference on Hardware/software Codesign and System Synthesis (CODES+ISSS)*, 2014, pp. 10:1–10:10.

[184] Y. Kim, F. Paterna, S. Tilak, and T. Rosing, “Smartphone analysis and optimization based on user activity recognition,” in *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*, 2015.

[185] N. Jung, G. Lee, S. Lee, and H. Cha, “Tbooster: Adaptive touch boosting for mobile texting,” in *Proceedings of the International Workshop on Mobile Computing Systems and Applications (HotMobile)*, 2016, pp. 63–68.

[186] L. Zhong and N. Jha, “Dynamic power optimization targeting user delays in interactive systems,” *IEEE Transactions on Mobile Computing*, vol. 5, no. 11, pp. 1473–1488, 2006.

[187] O. Sahin, L. Thiele, and A. Coskun, “Maestro: Autonomous qos management for mobile applications under thermal constraints,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 8, pp. 1557–1570, 2019.

[188] C. Isci, G. Contreras, and M. Martonisi, “Live, runtime phase monitoring and prediction on real systems with application to dynamic power management,” in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2006, pp. 359–370.

[189] K. Moazzemi, A. Kanduri, D. Juhasz, A. Miele, A. Rahmani, P. Liljeberg, A. Jantsch, and N. Dutt, “Trends in on-chip dynamic resource management,” in *Proceedings of the Euromicro Conference on Digital System Design (DSD)*, 2018.

[190] R. Pothukuchi, S. Pothukuchi, P. Voulgaris, and J. Torrellas, “Yukta: Multilayer resource controllers to maximize efficiency,” in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2018, pp. 505–518.

[191] A. Rahmani, B. Donyanavard, T. Muck, K. Moazzemi, A. Jantsch, O. Mutlu, and N. Dutt, “Spectr: Formal supervisory control and coordination for many-core systems resource management,” in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASP-LOS)*, 2018, pp. 169–183.

[192] Y. Hu, S. Liu, and P. Huang, “A case for lease-based, utilitarian resource management on mobile devices,” in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASP-LOS)*, 2019, pp. 301–315.

[193] A. Rahmani, A. Jantsch, and N. Dutt, “Hdgm: Hierarchical dynamic goal management for many-core resource allocation,” *IEEE Embedded System Letters*, vol. 10, no. 3, 2018.

[194] B. Donyanavard, T. Muck, A. Rahmani, N. Dutt, A. Sadighi, F. Maurer, and A. Herkersdorf, “Sosa: Self-optimizing learning with self-adaptive control for hierarchical system-on-chip management,” in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2019, pp. 685–698.

[195] T. Muthukaruppan, M. Pricopi, V. Venkataramani, T. Mitra, and S. Vishin, “Hierarchical power management for asymmetric multi-core in dark silicon era,” in *Proceedings of the Design Automation Conference (DAC)*, 2013.

[196] Z. Wang, Z. Tian, J. Xu, R. Maeda, H. Li, P. Yang, Z. Wang, L. Duong, Z. Wang, and X. Chen, “Modular reinforcement learning for self-adaptive energy efficiency optimization in multicore system,” in *Proceedings of the Asia and South Pacific Design Automation Conference (ASPDAC)*, 2017, pp. 684–689.

[197] A. Das, B. Al-Hashimi, and G. Merrett, “Adaptive and hierarchical runtime manager for energy-aware thermal management of embedded systems,” *ACM Transactions on Embedded Computing Systems*, vol. 15, no. 2, pp. 24:1–24:25, 2016.

[198] Z. Chen and D. Marculescu, “Distributed reinforcement learning for power limited many-core system performance optimization,” in *Proceedings of the Design Automation & Test in Europe Conference (DATE)*, 2015, pp. 1521–1526.

[199] R. Kim, W. Choi, Z. Chen, J. Doppa, P. Pande, D. Marculescu, and R. Marculescu, “Imitation learning for dynamic vfi control in large-scale manycore systems,” *IEEE Transactions on Very Large Scale Integration Systems*, vol. 25, no. 9, pp. 2458–2471, 2017.

[200] J. Lee, S. Nam, and S. Park, "Energy-efficient control of mobile processors based on long short-term memory," *IEEE Access*, vol. 7, pp. 80 552–80 560, 2019.

[201] Huaweis gpu turbo: Valid technology with overzealous marketing (accessed 2020). [Online]. Available: https://www.anandtech.com/show/13285/huawei-gpu-turbo-investigation/3

[202] Deepmind, meet android (accessed 2020). [Online]. Available: https://deepmind.com/blog/announcements/deepmind-meet-android

[203] J. Vitek and T. Kalibera, "Repeatability, reproducibility and rigor in systems research," in *Proceedings of the International Conference on Embedded Software (EMSOFT)*, 2011, pp. 33–38.

[204] N. Ding, D. Wagner, X. Chen, A. Pathak, Y. Hu, and A. Rice, "Characterizing and modeling the impact of wireless signal strength on smartphone battery drain," in *Proceedings of the International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, 2013, pp. 29–40.

[205] D. Wagner, A. Rice, and A. Beresford, "Device analyzer: Understanding smartphone usage," in *Proceedings of the International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MOBIQUITOUS)*, 2013.

[206] A. Passarella, "Power management policies for mobile computing," Ph.D. dissertation, University of Pisa, 2005.

[207] L. Yan, L. Zhong, and N. Jha, "User-perceived latency driven voltage scaling for interactive applications," in *Proceedings of the Design Automation Conference (DAC)*, 2005, pp. 624–627.

[208] R. Jota, A. Ng, P. Dietz, and D. Wigdor, "How fast is fast enough?: A study of the effects of latency in direct-touch pointing tasks," in *Proceedings of the Conference on Human Factors in Computing Systems (CHI)*, 2013, pp. 2291–2300.

[209] I. Mackenzie and C. Ware, "Lag as a determinant of human performance in interactive systems," in *Proceedings of the Conference on Human Factors in Computing Systems (CHI)*, 1993, pp. 488–493.

[210] M. Brocanelli and X. Wang, "Supervisory performance control of concurrent mobile apps for energy efficiency," *IEEE Transactions on Mobile Computing*, 2019.

[211] V. Seeker, P. Petoumenos, H. Leather, and B. Franke, "Measuring qoe of interactive workloads and characterising frequency governors on mobile devices," in *Proceedings of the International Symposium on Workload Characterization (IISWC)*, 2014, pp. 61–70.

[212] Q. Chen, H. Luo, S. Rosen, Z. Mao, K. Iyer, J. Hui, K. Sontineni, and K. Lau, "Qoe doctor: Diagnosing mobile app qoe with automated ui control and cross-layer analysis," in *Proceedings of the Internet Measurement Conference*, 2014, pp. 151–164.

[213] Y. Zhu and V. Reddi, "High-performance and energy-efficient mobile web browsing on big/little systems," in *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, 2013, pp. 13–24.

[214] F. Nah, "A study on tolerable waiting time: how long are web users willing to wait?" *Behaviour & Information Technology*, vol. 23, no. 3, pp. 153–163, 2004.

[215] D. Shingari, A. Arunkumar, B. Gaudette, S. Vrudhula, and C. Wu, "Dora: Optimizing smartphone energy efficiency and web browser performance under interference," in *Proceedings of the International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2018, pp. 64–75.

[216] N. Duca and T. Wiltzius, "Jank free: Chrome rendering performance," Google I/O 2013.

[217] M. Fiedler, T. Hossfeld, and P. Tran-Gia, "A generic quantitative relationship between quality of experience and quality of service," *IEEE Network*, vol. 24, no. 2, pp. 36–41, 2010.

[218] S. Bischoff, A. Hansson, and B. Al-Hashimi, "Applying of quality of experience to system optimisation," in *Proceedings of the International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, 2013, pp. 91–98.

[219] K. Chen, C. Wu, Y. Chang, and C. Lei, "A crowdsourceable qoe evaluation framework for multimedia content," in *Proceedings of the International Conference on Multimedia*, 2009.

[220] J. McCullough, Y. Agarwal, J. Chandrashekar, S. Kuppuswamy, A. Snoeren, and R. Gupta, "Evaluating the effectiveness of model-based power characterization," in *Proceedings of the USENIX Annual Technical Conference*, 2011.

[221] M. Walker, S. Diestelhorst, A. Hansson, A. Das, S. Yang, B. Al-Hashimi, and G. Merrett, "Accurate and stable run-time power modeling for mobile and embedded cpus," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 1, pp. 106–119, 2017.

[222] M. Said, S. Chetoui, A. Belouchrani, and S. Reda, "Understanding the sources of power consumption in mobile socs," in *Proceedings of the International Green and Sustainable Computing Conference (IGSC)*, 2018.

[223] M. Dong and L. Zhong, "Self-constructive high-rate system energy modeling for battery-powered mobile systems," in *Proceedings of the International Conference on Mobile Systems Applications and Services (MobiSys)*, 2011, pp. 335–348.

[224] A. Pathak, Y. Hu, and M. Zhang, “Where is the energy spent inside my app?: fine grained energy accounting on smartphones with eprof,” in *Proceedings of the European Conference on Computer Systems (EuroSys)*, 2012, pp. 29–42.

[225] C. Yoon, D. Kim, W. Jung, C. Kang, and H. Cha, “Appscope: Application energy metering framework for android smartphones using kernel activity monitoring,” in *Proceedings of the USENIX Annual Technical Conference*, 2012.

[226] K. Kim, D. Shin, Q. Xie, Y. Wang, M. Pedram, and N. Chang, “Fepma: Fine-grained event-driven power meter for android smartphones based on device driver layer event monitoring,” in *Proceedings of the Design Automation & Test in Europe Conference (DATE)*, 2014.

[227] W. Jung, C. Kang, C. Yoon, D. Kim, and H. Cha, “Devscope: A nonintrusive and online power analysis tool for smartphone hardware components,” in *Proceedings of the International Conference on Hardware/software Codesign and System Synthesis (CODES+ISSS)*, 2012, pp. 353–362.

[228] G. Srinivasa, S. Haseley, M. Hempstead, and G. Challen, “Quantifying process variations and its impacts on smartphones,” in *Proceedings of International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2019, pp. 117–126.

[229] Y. Yu and C. Wu, “Designing a temperature model to understand the thermal challenges of portable computing platforms,” in *Proceedings of the Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems (ITherm)*, 2018, pp. 992–999.

[230] M. Dousti, M. Ghasemi-Gol, M. Nazemi, and M. Pedram, “Thermtap: An online power analyzer and thermal simulator for android devices,” in *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED)*, 2015, pp. 341–346.

[231] Y. Gong, J. Yoo, and S. Chung, “Thermal modeling and validation of a real-world mobile ap,” *IEEE Design & Test*, 2017.

[232] J. Park, S. Lee, and H. Cha, “Accurate prediction of smartphones skin temperature by considering exothermic components,” in *Proceedings of the Design Automation & Test in Europe Conference (DATE)*, 2019, pp. 1500–1503.

[233] S. Gurrum, D. Edwards, T. Marchand-Golder, J. Akiyama, S. Yokoya, J. Drouard, and F. Dahan, “Generic thermal analysis for phone and tablet systems,” in *Proceedings of the Electronic Components and Technology Conference*, 2012, pp. 1488–1492.

[234] Q. Xie, M. Dousti, and M. Pedram, “Therminator: A thermal simulator for smartphones producing accurate chip and skin temperature maps,” in *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED)*, 2014, pp. 117–122.

[235] M. Ziegler, H. Liu, G. Gristede, B. Owens, R. Nigaglioni, J. Kwon, and L. Carloni, “Syntunsys: A synthesis parameter autotuning system for optimizing high-performance processors,” *Machine Learning in VLSI Computer-Aided Design*, pp. 539–570, 2019.

[236] M. Ziegler, G. Gristede, and V. Zyuban, “Power reduction by aggressive synthesis design space exploration,” in *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED)*, 2013, pp. 421–426.

[237] M. Ziegler, H. Liu, and L. Carloni, “Scalable auto-tuning of synthesis parameters for optimizing high-performance processors,” in *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED)*, 2016, pp. 180–185.

[238] S. Xydis, G. Palermo, V. Zaccaria, and C. Silvano, “Spirit: Spectral-aware pareto iterative refinement optimization for supervised high-level synthesis,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 1, pp. 155–159, 2015.

[239] O. Azizi, A. Mahesri, J. Stevenson, S. Patel, and M. Horowitz, “An integrated framework for joint design space exploration of microarchitecture and circuit,” in *Proceedings of the Design Automation & Test in Europe Conference (DATE)*, 2010, pp. 250–255.

[240] G. Mariani, G. Palermo, V. Zaccaria, and C. Silvano, “Desperate++: An enhanced design space exploration framework using predictive simulation scheduling,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 2, pp. 293–306, 2015.

[241] J. Kwon, M. Ziegler, and L. Carloni, “A learning-based recommender system for autotuning design flows of industrial high-performance processors,” in *Proceedings of the Design Automation Conference (DAC)*, 2019, pp. 218:1–218:6.

[242] M. Bokhari, B. Bruce, B. Alexander, and M. Wagner, “Deep parameter optimisation on android smartphones for energy minimisation - a tale of woe and a proof-of-concept,” in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, 2017, pp. 1501–1508.

[243] J. Guo, D. Yang, N. Siegmund, S. Apel, A. Sarkar, P. Valov, and K. Czarnecki, “Data-efficient performance learning for configurable systems,” *Empirical Software Engineering*, vol. 23, pp. 1826–1867, 2018.

[244] V. Nair, Z. Yu, T. Menzies, N. Siegmund, and S. Apel, "Finding faster configurations using flash," *IEEE Transactions on Software Engineering*, 2018.

[245] N. Siegmund, A. Grebhahn, S. Apel, and C. Kstner, "Performance-influence models for highly configurable systems," in *Proceedings of the Symposium on the Foundations of Software Engineering (FSE)*, 2015, pp. 284–294.

[246] M. Zuluaga, A. Krause, G. Sergent, and M. Pschel, "Active learning for multi-objective optimization," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2013, pp. 462–470.

[247] Y. Ding, N. Mishra, and H. Hoffman, "Generative and multi-phase learning for computer systems optimization," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2019, pp. 39–52.

[248] N. Siegmund, S. Kolesnikov, C. Kstner, S. Apel, D. Batory, M. Rosenmller, and G. Saake, "Predicting performance via automated feature-interaction detection," in *Proceedings of the International Conference on Software Engineering (ICSE)*, 2012, pp. 167–177.

[249] J. Ansel, Y. Wong, C. Chan, M. Olszewski, A. Edelman, and S. Amarasinghe, "Language and compiler support for auto-tuning variable-accuracy algorithms," in *Proceedings of the International Symposium on Code Generation and Optimization (CGO)*, 2011, pp. 85–96.

[250] S. Mittal, "A survey of techniques for approximate computing," *ACM Computing Surveys*, vol. 48, no. 4, pp. 62:1–62:33, 2016.

[251] A. Morajko, T. Margalef, and E. Luque, "Design and implementation of a dynamic tuning environment," *Journal of Parallel and Distributed Computing*, pp. 474–490, 2006.

[252] A. Martinez, A. Sikora, E. Cesar, and J. Sorribes, "How to determine the topology of hierarchical tuning networks for dynamic auto-tuning in large-scale system," in *Proceedings of the International Conference on Computational Science (ICCS)*, 2013, pp. 1352–1361.

[253] A. Martinez, A. Sikora, E. Cesar, and J. Sorribes, "Elastic: A large scale dynamic tuning environment," *Scientific Programming*, vol. 22, no. 4, pp. 261–271, 2014.

[254] Eembc andebench-pro (accessed 2020). [Online]. Available: https://www.eembc.org/andebench/about_pro.php

[255] M. Otoom, M. Alzubaidi, and N. Otoum, "Novel framework for designing representative usage-based benchmarks for smartphone," *IEEE Transactions on Consumer Electronics*, vol. 64, no. 4, pp. 480–488, 2018.

[256] A. Lochmann, F. Bruckner, and O. Spinczyk, "Reproducible load tests for android systems with trace-based benchmarks," in *Proceedings of the International Conference on Performance Engineering (ICPE)*, 2017, pp. 73–76.

[257] G. Srinivasa, R. Begum, S. Haseley, M. Hempstead, and G. Challen, "Separated by birth: Hidden differences between seemingly-identical smartphone cpus," in *Proceedings of International Workshop on Mobile Computing Systems and Applications (HotMobile)*, 2017, pp. 103–108.

[258] ARM, "Workload automation (accessed 2020)." [Online]. Available: https://github.com/ARM-software/workload-automation

[259] . Matters, "Google play store stats (accessed 2020)," 2020. [Online]. Available: https://42matters.com/stats

[260] N. Mehta, S. Natarajan, P. Tadepalli, and A. Fern, "Transfer in variable-reward hierarchical reinforcement learning," *Machine Learning*, vol. 73, no. 289, 2008.