An Intelligent Framework for Energy-Aware Mobile Computing Subject to Stochastic System Dynamics

by

Benjamin Gaudette

A Dissertation Presented in Partial Fulfillment
of the Requirement for the Degree
Doctor of Philosophy

Approved September 2017 by the
Graduate Supervisory Committee:

Sarma Vrudhula, Co-Chair
Carole-Jean Wu, Co-Chair
Georgios Fainekos
Aviral Shrivastava

ARIZONA STATE UNIVERSITY

December 2017

ABSTRACT

User satisfaction is pivotal to the success of mobile applications. At the same time, it is imperative to maximize the energy efficiency of the mobile device to ensure optimal usage of the limited energy source available to mobile devices while maintaining the necessary levels of user satisfaction. However, this is complicated due to user interactions, numerous shared resources, and network conditions that produce substantial uncertainty to the mobile device's performance and power characteristics. In this dissertation, a new approach is presented to characterize and control mobile devices that accurately models these uncertainties. The proposed modeling framework is a completely data-driven approach to predicting power and performance. The approach makes no assumptions on the distributions of the underlying sources of uncertainty and is capable of predicting power and performance with over 93% accuracy.

Using this data-driven prediction framework, a closed-loop solution to the DEM problem is derived to maximize the energy-efficiency of the mobile device subject to various thermal, reliability and deadline constraints. The design of the controller imposes minimal operational overhead and is able to tune the performance and power prediction models to changing system conditions. The proposed controller is implemented on a real mobile platform, the Google Pixel smartphone, and demonstrates a 19% improvement in energy efficiency over the standard frequency governor implemented on all Android devices.

DEDICATION

To my mother and father

Without your support, all of this could never have happened

ACKNOWLEDGMENTS

I would like to thank my advisor, Prof. Sarma Vrudhula, for all his guidance and advice throughout the years. I would additionally like to thank Prof. Carole-Jean Wu for her guidance, and insight on all things related to computer architecture. Without these two people, this work would not have been possible. My sincere thanks to my committee members for taking the time to review my work, attend my presentations and offer many insightful suggestions. I would like to thanks Vinay Hanumaiah, whose work in dynamic energy management served as a foundation for my own.

I would also like to thank the technical support I have received from members of the Center of Embedded Systems as well as from other industry support. Specifically, I would like to mention Rudy Beraha, Chris Lott, and Vikram Gupta of Qualcomm for providing me with a summer internship and a fantastic opportunity to learn about mobile SoC development. Without these individuals, design, implementation, and experimentation of my work would have been much more fractious.

Most importantly, my parents have been a constant source of encouragement and support through these many years. I am grateful for everything they have done.

Finally, my thanks to the many sponsoring agencies and academic units that contributed to funding my research over the years. Specifically, the NSF I/UCRC Center for Embedded Systems and National Science Foundation grants #1361926 and #0856090; National Science Foundation grants for CNS #1358805 and CCF #1525462; and the Science Foundation Arizona under the Bisgrove Early Career Scholarship. Thank you to ASUs School of Computing, Informatics & Decision Systems Engineering for granting me a research assistantship, to the Ira A. Fulton Schools of Engineering Deans Office for the Deans Fellowship, and to the ASU Graduate College for doctoral stipend support.

TABLE OF CONTENTS

CHAPTER Page

CHAPTER Page

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

Figure Page

Chapter 1

INTRODUCTION

In recent years, there has been an explosive growth in the use of mobile devices–especially smartphones–becoming ubiquitous for everyday computing needs. These devices serve as a pervasive platform in which business can implement services and applications to directly interact with end users. However, pivotal to the success of the mobile system is *user satisfaction*–a general term used in business and marketing to encompass the overall satisfaction a user has with a product. This user satisfaction is an amalgamation of many factors such as application interface, feature set, and *quality of service* (QoS). QoS is some quantifiable measure of the application or system's performance. This can refer to the response time of an application (e.g. loading a web page), a rate of processing (e.g. the frames per second of a video decoder), or any other measure directly associated with the user's satisfaction of the application and device. In this dissertation, QoS is considered as the primary feature of ensuring user satisfaction since it is a quantifiable attribute which is controllable by the mobile system.

To highlight the importance of high QoS consider a recent survey [1] of e-commerce shoppers using mobile devices as shown in Figure 1.1. The study found that 19% of the users would abandon a mobile web page requiring more than 5 seconds to load and 49% of the users would abandon the page after 10 seconds. Perhaps worse than this, 79% of the participants stated that if they are dissatisfied with their user experience, they are less likely to use the application again. By extrapolating this data, the study concluded that "[i]f an e-commerce application is making $100,000 per day, a one second page delay could potentially cost you $2.5 million in lost sales every

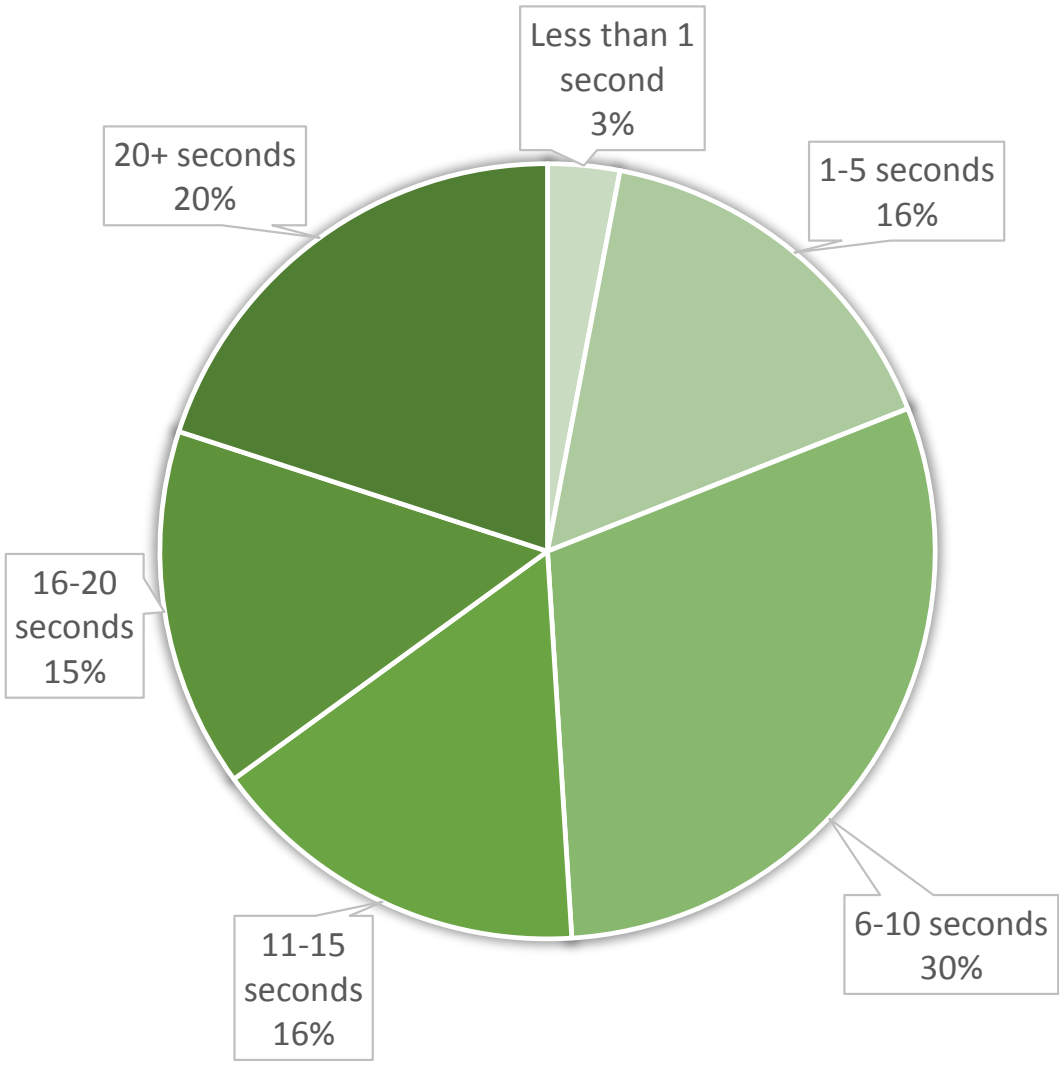


Figure 1.1: The Results of a survey of mobile web users asking how long they would be willing to wait for a web site to load. Recreated from [1].

year." From this perspective, it is imperative to ensure user satisfaction even at the cost of additional power; however, due to finite energy capacity and lack of active cooling mechanisms, it is not typically optimal for mobile platforms to sustain high performance states.

This dissertation addresses the issue of managing mobile processors to efficiently balance high quality of service and low energy consumption. This is complicated due to (1) the large number of interactions between system elements (Figure 1.2) and (2)

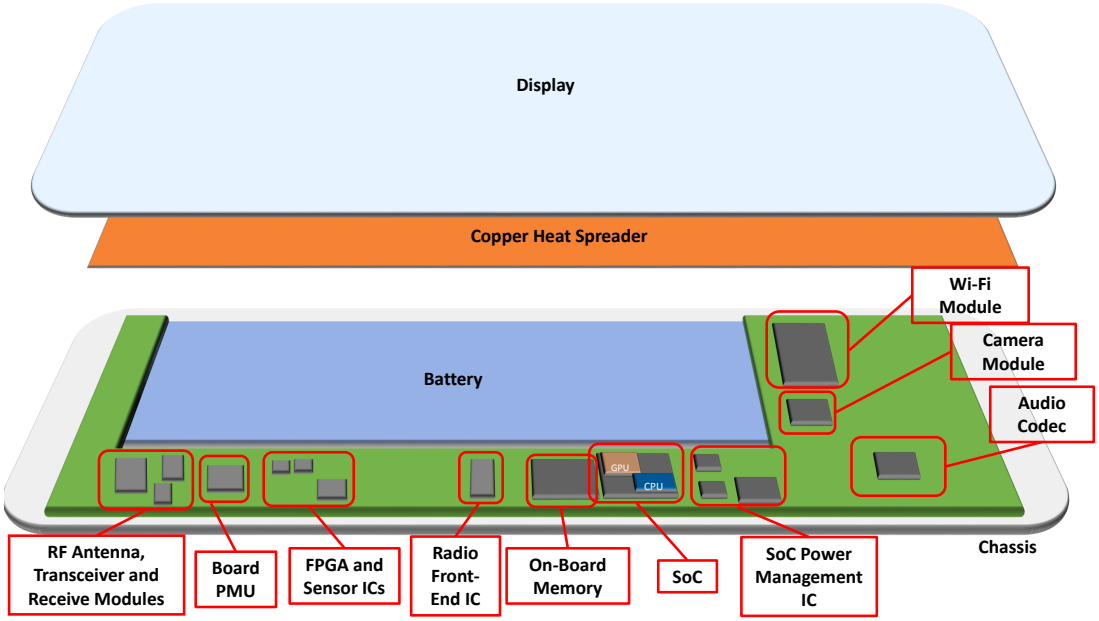


Figure 1.2: A cut-away diagram displaying numerous system components present in mobile devices.

the presence of system non-determinisms. This dissertation provides two frameworks to accurately model and predict QoS and power in the presence of numerous system non-determinisms. The first, applicable to offline analysis and provides mechanisms for insightful design space exploration. The second is a light-weight, black box approach which can be learned and evaluated in real-time with minimal difference in error compared to the offline approach. A major contribution of this dissertation is the consideration that QoS is a non-deterministic value – an important attribute previously neglected by previous dynamic energy management systems. The modeling frameworks provide comprehensive methods of analyzing variations in QoS subject to various controllable and uncontrollable parameters. Finally, this additional knowledge of QoS variation is leveraged to introduce new dynamic energy management techniques. These techniques are software controlled at the operating system level to achieve various objectives such as maximizing QoS, energy-efficiency and battery life

subject to thermal, energy, and deadline constraints.

1.1 Non-determinisms in Mobile Devices

As discussed in the previous section, designing and operating mobile devices to ensure high levels of user satisfaction is vital to the success of the system. In order to properly predict the power and QoS of an application, an accurate prediction framework is needed. However, this is complicated in the presence of numerous sources of non-determinisms. In this section, the challenges of modeling QoS of mobile systems are briefly discussed.

For desktop and server processors, mechanistic modeling methods have proven to be an effective mechanism to model system performance [2, 3, 4, 5]. Mechanistic methods relate a set of deterministic system observables to the modeled quantity of interest. These methods relay on underlying assumptions of the system architecture and the software design in order to correctly capture the mathematical form of the model. These models begin to lose fidelity should system uncertainties be introduced, such as architectural delays, network conditions, and input complexity. In this section (and more formally in Chapter 2), this dissertation will argue that a key difference between the desktop/server environment and the mobile environment is the high level of system uncertainty and, as such, mechanistic models lack the flexibility needed to be applicable to mobile systems.

One can think of variations in an application's execution as a sequence of interruptions from various sources in the normal flow of computation. These interruptions are defined as intervals of idle periods, whose endpoints are random points in time, and whose lengths are random variables. The cause and severity of these variations are numerous, dependent on the application, the user, and the mobile system. To illustrate that variations are significant on mobile applications and devices consider

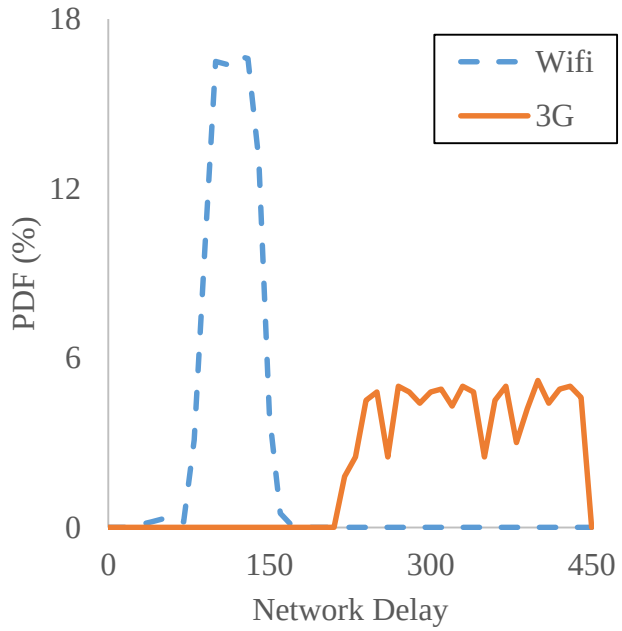


Figure 1.3: The probability density function of delay for 3G and Wifi networks.

a simple use case of web browsing. As input, the web browser receives an address of a web page from the user, either directly or by evaluating a link. The browser must then establish a connection with the remote server which stores the page in order to download its assets. As the assets are downloading, the page must evaluate the HTML code and render the resulting web page.

The first and perhaps most obvious source of variation is the delay due to network conditions. Unlike, server and desktop environments in which the network conditions are relatively static (i.e. a wired or singular wifi connection), mobile devices utilize numerous types of networks, each with their own delay characteristics. Figure 1.3 depicts the distributions of network delay for two types of connections, wifi and 3G based on the recents results of [6]. One can see in the case of a 3G connection, there exists significant variations in the delay ranging from 200ms to 450ms with approximately uniform distribution. In comparison the wifi network provides a more normal-like and tighter distribution between 50ms and 150ms. It should be noted that

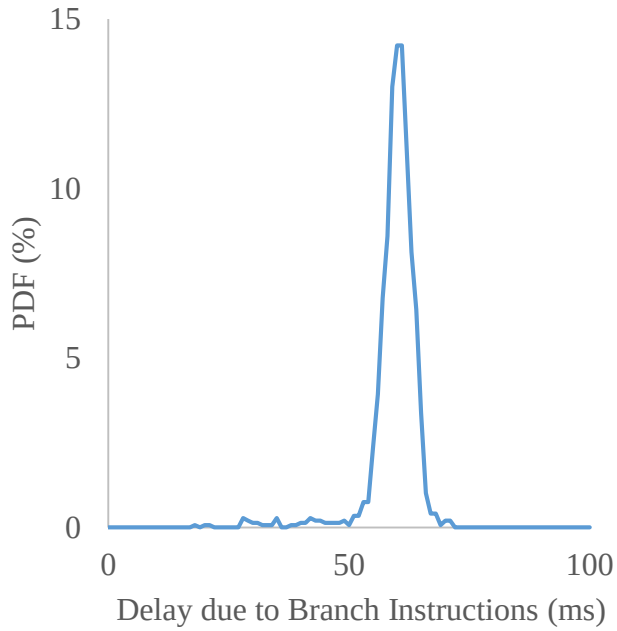


Figure 1.4: The probability density function of delay due to branch instructions when loading a web page.

as network bandwidth and delay improve, the majority of the application overhead is shifted to the performance of the mobile processor rather than the network [7].

A less obvious source of variation is due to the architectural events of the mobile processor such as the number of cache misses, page faults, tlb misses, branch miss-predictions and so on. While any one given occurrence of these delays may only result in an interruption of several processor cycles, these events occur in large quantities especially on the more simplistic architectures seen in low power mobile devices. Figure 1.4 shows an example of the delay caused by branch instructions while decoding a video file and demonstrates just how severe these architectural delays can be. This shows that from run-to-run, the the interruptions due to architectural events can vary greatly; in the case of branch miss-predictions this variance can be between 50 and 70 ms. Ultimately, all of these sources of variation combine to create a large variation in the processing time of the video stream (i.e. the QoS).

In order to understand the likelihood of achieving any given QoS and power combination as well as providing a measure of prediction uncertainty, it is necessary to relate the parameters of the distribution to the control variables, input characteristics, and system states. For instance, one could capture the severity of architectural delays using on-chip performance monitoring counters. Additionally, the application could provide hints about the complexity of the input. For example, with web browsing one could include quantifiable characteristics of the web page such as the number of Uniform Resource Identifiers (URIs) and the types of objects in the URIs that affect the execution time. In this way, the modeling of execution time of a web browser in processing a web page can be made sensitive to the complexity of the web page along with the system states. Such a model provides a simple and effective means to relate the distribution function of the execution times to the processor core frequencies for dynamic energy management.

1.2 Optimal Control of Mobile Devices Subject to Non-deterministic Operation

To illustrate the importance of considering the likelihood of satisfying constraints, and the potential trade-offs, once more consider the example, *mobile web browsing*. Suppose that a processor allows multiple frequency settings, f_1-f_6. Figure 1.5(a) shows a *hypothetical* plot of the *average* load time (dotted curve) for a given web page (which involves loading, processing, and rendering the web page) over multiple invocations, at different frequency settings. As stated earlier, the different invocations at a fixed frequency lead to different load times due to various factors such as the non-deterministic nature of the system states. The vertical bars at each frequency indicate the observed minimum and maximum load times. Figure 1.5(b) shows the average *energy efficiency*, expressed as *performance per watt* (PPW) of the application at each frequency.

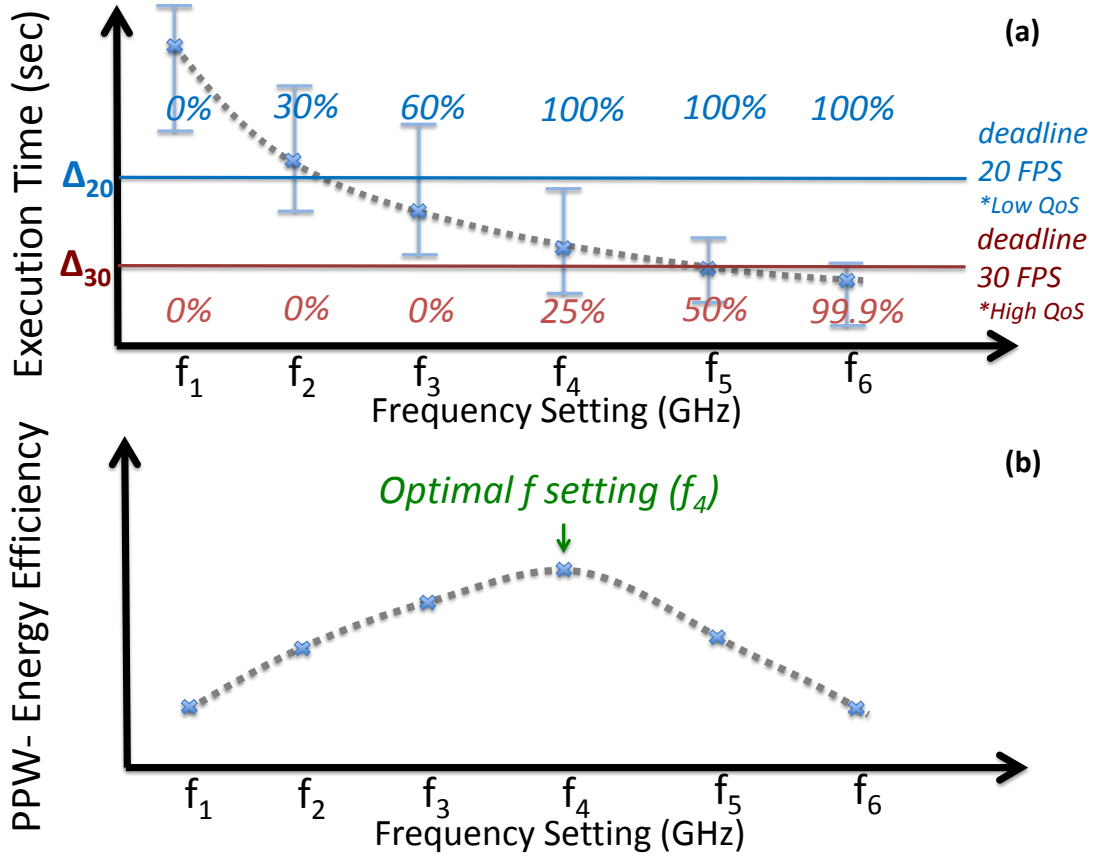


Figure 1.5: Conceptual plots of mean execution time and PPW of an application running at various clock frequencies, with error bars depicting range of variation, and probability of meeting a given deadline at each frequency.

The QoS here is a composite measure of how stringent the deadline is for loading a web page, as well as the likelihood of meeting that deadline. Suppose that in order to satisfy a *high* QoS target the web page must be loaded within 3 seconds. Let Δ_3 denote this deadline (see Figure 1.5(a)). To meet this deadline with a likelihood of 99.9% will require running the processor at frequency f_6. However, at this frequency the average energy efficiency or PPW will be very low. This may result, for instance, in substantially reducing the residual battery charge, raising the possibility of an inoperable phone until the next charging opportunity. However, if the user is willing to wait additional time for the web page to load for a longer lasting battery, then the deadline can be increased to 5 seconds (Δ_5). Then the phone can be operated at

f_4, at which the energy efficiency is maximum, with a near certainty of meeting the looser deadline.

Due to the stochastic nature of mobile workloads, this energy efficiency to performance trade off becomes a non-deterministic analysis of determining the likelihood of the system's performance and power characteristics. As such, this type of analysis can only be achieved by modeling the numerous sources of variations within the mobile systems and workloads.

1.3 State-of-the-Art in Dynamic Energy Management

Over the past decade, a large body of research has been published on optimizing energy efficiency of computing devices. Energy-efficient and low-power computing aims at reducing power wastage via circuit, architectural and algorithmic-level techniques such as stand-by mode, clock-gating, and dynamic voltage and frequency scaling (DVFS). However, due to the high power density of modern processors, high processor temperatures have become a critical issue, causing device failures, increased leakage power, and throttling performance if left unchecked. For mobile devices, the thermal issue is further exasperated by the lack of active cooling devices (e.g. fans) which are typically present in desktops and laptops. Thus dynamic thermal management (DTM) and dynamic energy management (DEM) techniques are even more imperative in the mobile domain.

One of the earliest and simplest DEM methods is the stop-and go policy [8]. This policy simply states that for a given maximum temperature threshold, shut off the processor if the processor temperature exceeds the threshold and allow it to cool to a predefined value before starting up again. This scheme will certainly address thermal issues raised earlier; however, it does so at significant performance penalties.

To reduce the impact on performance, DVFS techniques were introduced. DVFS

allows for either discrete or continuous control over the operational voltage and frequency of the processor. As a result, up to cubic savings in power can be achieved for linear reduction in performance.

1.3.1 Classification of Related Work

This section primarily focuses on energy management techniques that operate at the operating system level. These techniques are primarily software-based approaches that use built-in hardware support to alter the power, performance, and thermal characteristics of the processor to optimize a given objective while ensuring all constraints are satisfied. The related techniques are categorized broadly into three main topics: optimization goals, controlling mechanisms, and modeling and prediction.

Every energy management problem starts with a specification of optimization goals and constraints that need to be met. While the nuances of these goals can vary wildly, these goals can typically categorize them into one of three types: *maximizing performance*, *minimizing energy*, or *maximizing energy-efficiency*.

In order to achieve the optimization goal and constraints, control mechanisms are needed. Typical control knobs include *dynamic voltage and frequency scaling*, *task migration*, and *task scheduling*. *Dynamic voltage and frequency scaling* as the name suggests, allows for the alteration of cpu-core voltage and frequency; thereby affecting performance, power consumption, and temperature. Task migration is the process of moving tasks between computational units. This provides a better matching of workload to cores for performance and power. In the event of heterogeneous architecture, the effect of task migration is magnified. *Task scheduling* is the ordering of which tasks to execute. By intelligently determining the task schedule, the controller is capable of shaping the power, temperature, and performance time-profile of the processor.

In order to solve DEM optimization problems efficient and accurate modeling techniques are needed to capture the relationship between the system environment, control knobs, and the given objectives and constraints. These methods can be model equations provided a priori by manufactures, or derived by correlating the output measurements with input control values. Modeling and prediction mechanisms play a very crucial role in determining the quality of the solutions and the complexity of the resulting DEM implementation, and for this reason, choosing the right power and thermal models is the most challenging aspect in a DEM optimization. However, most if not all of the works have assumed that the underlying execution time quantities are deterministic. In modern systems, outside sources of influences such as cache misses, branch mispredictions, OS scheduling, and so-on cause numerous uncertainties to system performance.

A summary of the classification of DEM techniques is shown in Figure 1.6. The following sections briefly compare the existing techniques in each of the above DEM categories.

1.3.2 Optimization Goals

Over the years, researchers have expanded the scope of DEM to many objective functions. One of the earliest and still most attractive is the maximizing of performance. Performance is an ambiguous term and as such its meaning can vary from work to work. One such performance metric is processor throughput represented by instructions-per-cycle (IPC) or *instructions-per-second* (IPS). There has been an extensive amount of literature on this subject. [9, 10, 11] considers the case when the system is constrained by power or energy budgets, [12, 13, 8, 14, 15, 16] addresses thermal limitations, and [17] considers task deadline constraints.

Another form of performance comes to us in Quality of Service (QoS) and user

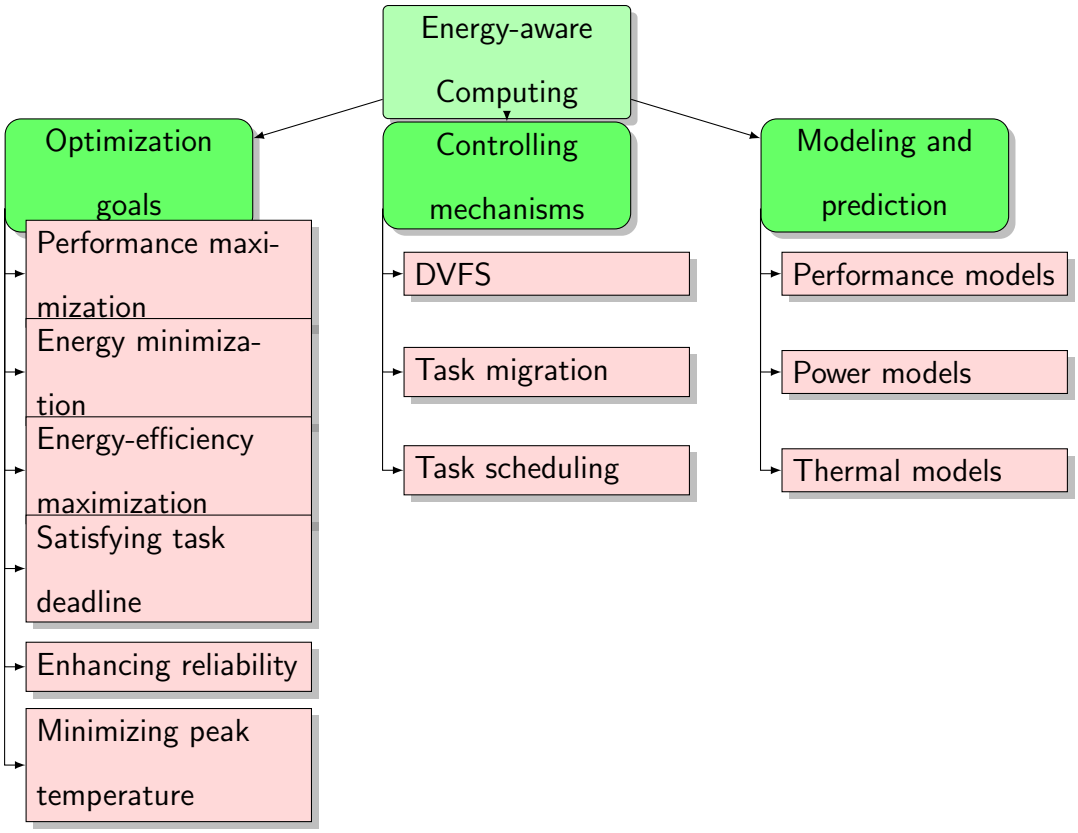


Figure 1.6: Hierarchal organization of the literature survey.

satisfaction. QoS metrics are specific to the application domain, for example in video playback the frames-per-second rendered to the display is a measure of QoS or for network communication the data bandwidth (after packet loss) could also be considered a metric of QoS. In [18], the effect of DVFS on the load time of web pages was examined. The authors present a static frequency scheduling algorithm to determine performance optimal operation subject to deadline constraints.

The performance-per-watt (PPW) metric has been studied by several researchers. It represents a systems energy-efficiency and serves as a means of balancing performance and power consumption. The authors of [19] formulated the non-linear optimization problem for a thermally constrained, leakage-dependent multi-core processor. They then proposed a greedy based DVFS-based controller which outperformed traditional Linux governors by a 20 to 30%.

While most of the prior works focus on optimizing system energy efficiency for the application domain of CMPs, a few more recent works started examining and designing energy efficiency optimization algorithms for user-centric, interactive smartphone applications. Egilmez et al. [15] specifically aimed at improving user satisfaction through DVFS-based control knobs to maintain a low and comfortable device skin temperature. Singla et al. [16] developed power and thermal models for a modern mobile platform and proposed a closed loop thermal control to adjust processor frequencies.

1.3.3 Control Mechanisms

Of all the control mechanisms that exist for DEM, DVFS is the most prevalent. DVFS provides up to cubic reduction in power consumption for a linear performance penalty. In [20] the DVFS problems is proposed using a constrained convex optimization formulation; however, due to the complexity of the formulation, the solution was presented only for offline calculations. In [13], the authors simplified the solution by providing the optimal zero slack control policy which is efficient enough to calculate at run time.

DVFS has also been applied to application specific QoS optimizations as well. In [21] the authors characterized the tradeoffs between power management and tail latency for server applications. From this analysis they determined that the commonly used service techniques were not sufficient to reach the stringent server QoS requirements and did not provide the most energy efficient method of service either. Instead they suggest a careful, workload specific frequency scaling policy.

Another popular control mechanism is task migration. In this problem, the task to core allocation and scheduling is determined in order to achieve the objective and constraints. In homogeneous processors, this technique is primarily used for altering

the temperature gradient across the die of a processor [22]; however, heterogeneous processors allow for stark power/performance tradeoffs when using task migration [23, 24, 25]. Typically, context switching between heterogeneous components comes at a large overhead; therefore a poorly designed task scheduler can have severe impact to performance and power.

1.3.4 Modeling and Prediction

Modeling methods can vary widely in accuracy and by extension, computational complexity. It is common to represent the performance of tasks bound to processing elements as a collection of linear equations based on the expected throughput of each task [26, 27, 28]. These methods are advantageous in their simplicity; however, they typically overlook crucial characteristics of the system and application which can lead to inaccurate results. To overcome some of the limitations caused by the use of linear equations, Bogdan et al. [29] proposed a performance prediction model which leveraged fractional calculus. This extension proved to be effective in deterministic routing systems. Another common methodology for performance analysis involves sampling. Similar to the problem setup of this paper, Wernsing and Stitt [30] suggested that the execution time of a task is a function of some work metric or characteristic. In this case the authors limited this function to a deterministic, monotonically non-decreasing function which is determined via regression analysis. From these models, the expected function execution time is directly predicted based on the work metric and proved to be an effective means of performance prediction in systems with low execution time variance.

To model pipelined parallel applications, scenario-aware dataflow models offer an accurate albeit, complex solution [31]. Real-time calculus and network calculus [32] offer an alternative which provides some solutions to the complexities of scenario-

aware dataflow modeling. For example, a round robin scheduler which used real-time calculus for data flow prediction was developed for image processing on multiprocessor system-on-chip [33]. Qian et al. proposed a network calculus performance model for a general multi-router system with contention [34]. However, these prior network calculus-based works assumed that the system is a deterministic queuing system.

Power and thermal models share similar implementation methodologies. Icepak [35] is a thermal modeling method using Finite-element methods (FEM) to represent the spatial thermal-power response of an SoC. It is one of the most accurate methods available however it can take hours or even days to simulate several seconds of an SoC's operation. A more computationally efficient methodology is proposed by HotSpot [36] via compact thermal modeling (CTM). In CTM the SoC is represented as a circuit schematic where nodes may represent processing blocks (e.g. FPUs, CPUs, memory, etc.) and between nodes are interconnects of resistors and capacitance representing the thermal properties of the SoC. In [37] this formulation was represented as a linear state model and computation time was reduced to the point necessary for dynamic control.

1.4 Novelty and Paper Structure

In this section a brief autobiographical sketch is provided, covering my research areas, publications, and patents while attending Arizona State University. Following this, an outline of the dissertation structure is provided which details both the flow of the this document as well as the novelty of this research.

1.4.1 Publications, Patents, and Research

I began my study at Arizona State University in 2006 while seeking a bachelors of science and engineering in computer systems engineering. In 2009, I began working

with Dr. Sarma Vrudhula, investigating the applications of renewable energy in wireless sensor networks. Along with another undergraduate student, Eric Munson, we developed a prototype sensor node which could be used to monitor water conditions in areas difficult to travel to. The node was completely self-sufficient, utilizing a solar panel to harvest and store energy.

This work later evolved into the research area of my Master's thesis. During this time, I developed algorithms to manage a distributed network of solar-powered wireless sensor nodes which (1) would actively manage each sensor's sensing range to maintain maximum coverage of some region and (2) maintain connectivity to some centralized node for data collection. Condition (1) was presented in my very first publication at IEEE INFOCOM 2012 [38] while (2) was added to the journal article published in ACM Transactions on Sensor Networks (TOSN) [39].

As I transitioned from Master's to PhD, I began my study of energy management of computing systems – following the work of a former PhD student, Vinay Hanumiah. Dr. Hanumiah researched DVFS control of multi-core desktop processors, developing a control algorithm named STEAM [19]. As a co-author of this work, I provided the implementation of the algorithm in C on an Intel Sandybridge processor and aided in the data collection and evaluation. This work was awarded a US patent [40].

As covered in this dissertation, I extended Dr. Hanumiah's research into the mobile domain, where system non-determinisms play a much larger role in power, performance, and energy efficiency. As a first step, a formalization of the energy management problem of smartphones was presented at the 2016 IEEE International Symposium on High Performance Computer Architecture (HPCA) [41]. This work used heuristic arguments to justify that the distribution of execution times for a given mobile application follows the gamma distribution function. Additionally, the work provided a methodology to calculate the execution time distributions of mulit-

threaded applications which could be used to evaluate different control strategies in an offline manor. However, this work was limited by several factors: (1) the assumption of a known distribution and (2) the overhead of the workload analysis. These factors are rectified in the latest work (covered in Chapters 4 and 5) using polynomial chaos theory to represent the power and performance of mobile applications. This work has been submitted to IEEE Transactions on Mobile Computing (TMC) for publication.

1.4.2 Dissertation Outline

In this dissertation, a complete framework for DEM in mobile systems is presented. This dissertation features various methodologies to model and predict system power and QoS in the presence of numerous sources of non-determinisms–a defining condition of mobile workloads. The provided methods allow system designers to perform detailed offline system state exploration or online system control in a computationally efficient manner. In addition, an architecture independent closed loop, energy-aware controller is presented along with a practical implementation for most modern smartphones. The models are highly accurate, reducing prediction error by a factor of **3X** over state-of-the-art methods. Thanks to this the proposed controller achieves a 19% improvement in energy-efficiency over the stock Android governors.

The following outlines the topics covered in this dissertation (see also Figure 1.7).

- Chapter 2 presents an overview of the challenges around modeling QoS, thermal, and power for mobile systems. This chapter provides the motivation for the need to model performance and power of mobile applications as a stochastic quantity will be used by the prediction frameworks and the DEM controller presented in later chapters.

- Chapter 3 presents the formulation of an offline QoS analysis framework. The

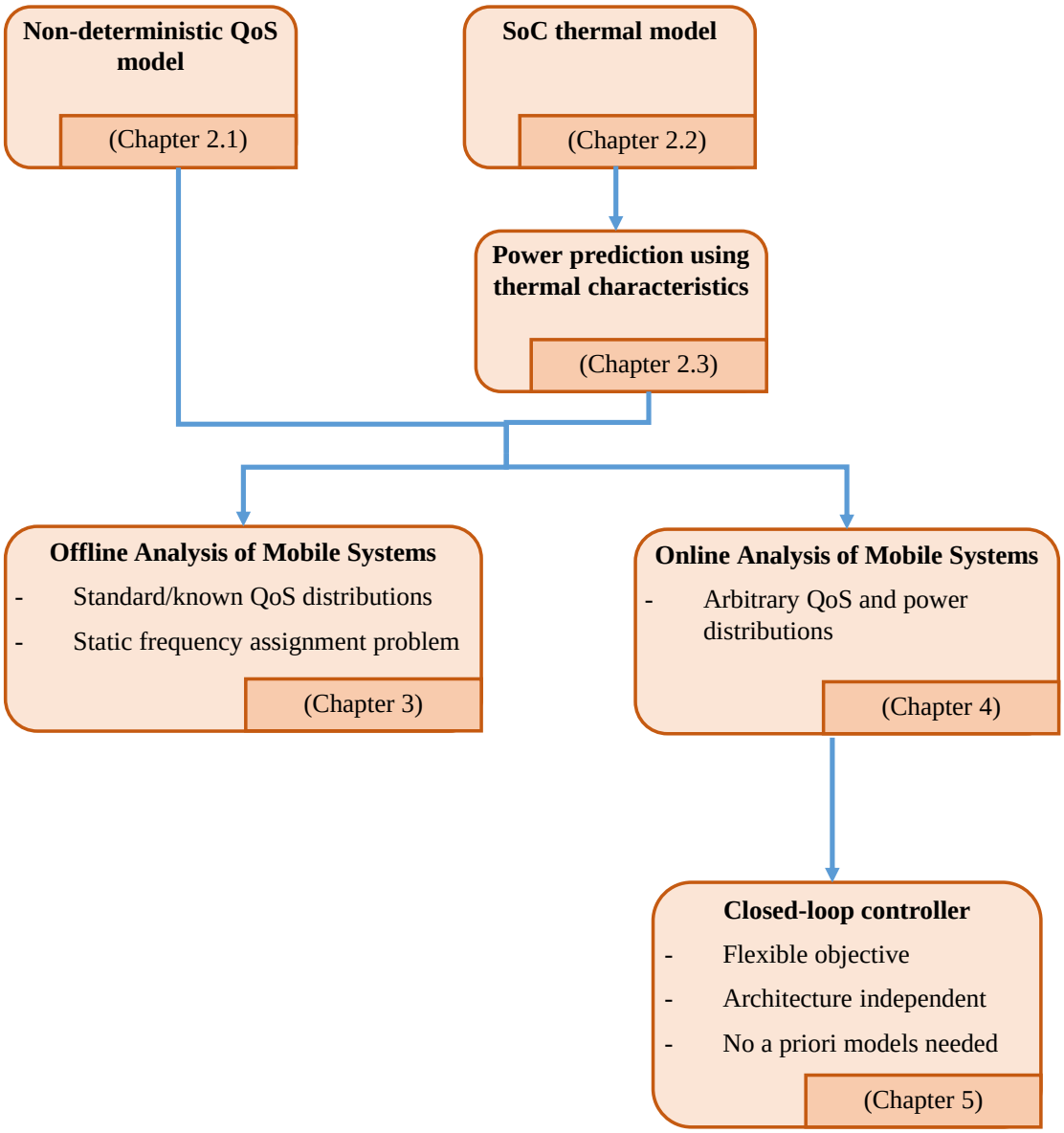


Figure 1.7: Outline of topics covered in this dissertation.

proposed approach provides a means to improve mobile user experience by balancing performance and energy with probabilistic guarantee of QoS. The chapter first hypothesizes, and then experimentally validates a closed form model to represent the execution times of single threaded mobile applications. The model parameters are functions of the control variables (e.g., voltage and frequency), as well as characteristics of the data being processed (e.g., complexity of the web page component being processed). The single-threaded model is then extended to several multi-threaded configurations including independent parallel execution and dependent sequential/pipelined execution. The proposed statistical models are used in the formulation of an optimization problem, the solution to which is a static, lightweight controller that optimizes energy efficiency of mobile applications, subject to constraints on the *likelihood* that the application execution time meets given deadlines.

- Chapter 4 proposes an efficient and accurate means of estimating QoS subject to numerous sources of uncertainty, suitable for online system control and feature learning. The proposed method constructs a QoS model and learns its parameters directly from the data, making no assumptions of the underlying distributions of performance. A demonstration of the versatility of the model is presented by developing QoS prediction techniques for the interactive mobile workload, web browsing. Additionally, a general model is proposed to predict system performance (instructions per second) for a wide variety of mobile workloads. Additionally, an online mechanism to dynamically tune our prediction models is derived, requiring little to no offline training. Finally, the models are used to provide a system analysis to generate per-frequency distributions of system energy efficiency – allowing one not only to determine the expected

energy efficiency as a function of frequency, but also the likelihood that this energy efficiency is achieved.

- Chapter 5 proposes a robust, architecture independent, closed-loop controller. This controller is capable of handling various objectives and constraints including thermal, QoS, delay, and energy. In addition, a formulation is given to estimate power consumption as a function of the SoC thermal process. This is especially critical in the mobile domain where power sensors are rarely included or provide updates too infrequently to be applied to DEM controllers. The controller design includes state-of-the-art error reduction techniques to learn both the system dynamics and user usage patterns.

- The dissertation is concluded with a summary of contributions and major findings. Additionally, a brief survey of open problems is presented as potential future research topics.

Chapter 2

SOC SYSTEM MODELS

As shown in Figure 2.1, the thermal, performance, and power characteristics of mobile computing devices have a strong interaction with one another. User demands place expectations on the mobile device. In order to service these expectations, the mobile device must alter its working conditions. For example, the device may increase its cpu clock frequency in order to achieve the highest possible quality of service. However, in doing so, the device will have to consume additional energy to met this demand. Because of the limited energy storage available to mobile devices, increasing the device performance arbitrarily high is not possible. Additionally, high power consumption results in increased thermal conditions do to circuit inefficiencies

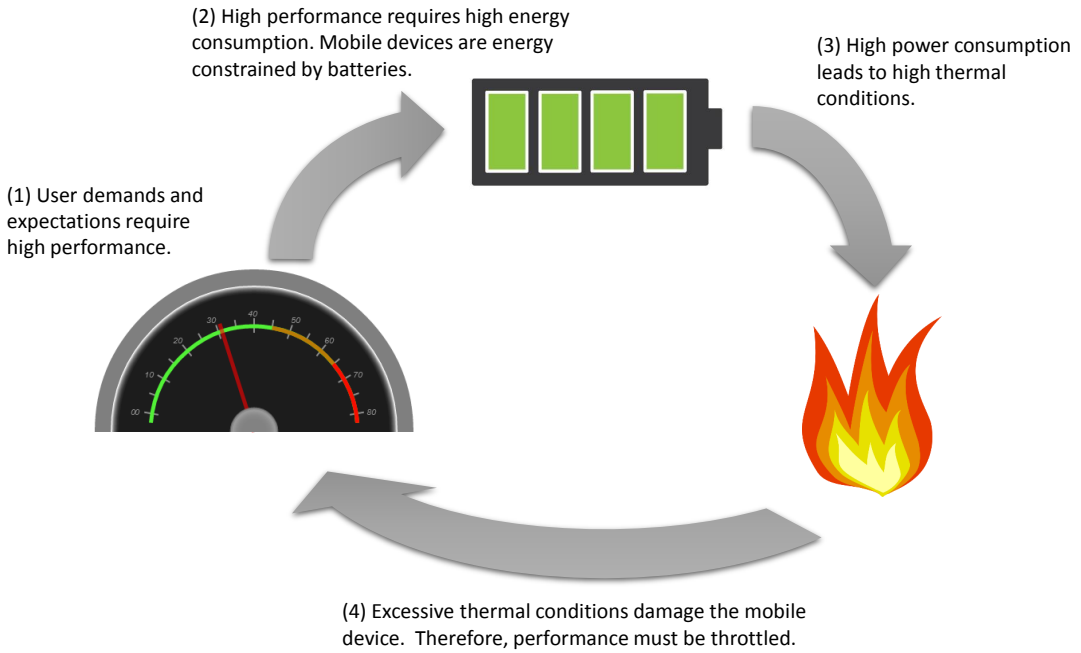


Figure 2.1: Illustration of the interaction between performance, power, and temperature for mobile computing devices.

which result in heat generation. Excessive thermal conditions may lead to unreliable operating conditions and even damage the device or the user. Therefore, the thermal characteristics of the device further limit the maximum possible performance draw.

While prior work in desktop and server energy management has considered the effects of these interactions; the limitations of mobile devices have not been adequately explored. This dissertation proposes the following advancements to performance, power, and thermal modeling for mobile devices:

1. Prior work in performance and power modeling has been developed with desktop and server workloads in mind. That is, workloads are assumed to have long, steady periods of activity. In the mobile domain, this is not true. Rather, workload characteristics are highly variable over time and stochastic in nature due to numerous system non-determinisms. This dissertation, proposes a modeling framework to accurately capture the effect of these variabilities.

2. Power prediction methods presented in literature assume the availability of accurate, real-time power sensors for each major component within the computing device. Mobile devices lack this granularity of power sensing. Instead this dissertation proposes utilizing knowledge of the thermal characteristics of the device to deduce prior power consumption values. This will act as a surrogate to power sensors and make it possible to train the proposed stochastic models.

2.1 QoS, Performance, and Execution Time Models

As discussed in the previous chapter, user satisfaction of mobile applications and devices hinges on several factors–the most important of which being quality of service. While QoS is a rather ambiguous term, here it is specifically defined as a quantifiable/enumerable measure of user satisfaction. Depending on the application, this can

be represented as the response time (e.g. execution time) of an application/task or some measure of throughput (e.g. frames per second or instructions per second). The following section provides a general mathematical representation of QoS while the specifics of modeling this quantity is given in Chapters 3 and 4.

Consider an SoC with n processing elements and q tasks which need to be executed. Each processing element is assumed to be able to execute a single task at any given time and has several controllable parameters which affect the QoS of the task. For the purposes of this dissertation, the controllable parameters of operating speeds/frequencies $\boldsymbol{s}(t)$ as well as operating voltages $\boldsymbol{v}(t)$ are considered. In practice, these two quantities are often dependent upon one another and cannot be changed independently. Therefore, unless otherwise noted this dissertation will discuss only $\boldsymbol{s}(t)$ as the controllable parameter. Note that symbols in **bold** denote vectors or matrices, and all vectors are considered column vectors.

In regards to computer architecture, the most fundamental representation of performance is the rate at which instructions are processed by a computing system. Specifically, the efficiency of task j running on processing element i can be characterized by the instructions per cycle $IPC_{i,j}(t)$. Note, although IPC is presented here as only a function in time, in reality it is an unknown function of many parameters including operating frequency, workload characteristics, architectural events and so on [2]. Additionally, this representation is only true assuming negligible parallelism. Should tasks be dependent upon one another or if the effects of co-scheduling are to great, the $IPC_{i,j}$ is also a function of $IPC_{k,l} \forall k \neq i, l \neq j$. Given an accurate representation of $IPC_{i,j}(t)$, the performance of the SoC is defined by $\sum_{i=1}^{n} \sum_{j=1}^{q} w_{i,j}(t) IPC_{i,j}(t) s_i(t)$ where $w_{i,j}$ is zero in the case that task j is not running on processing element i at time t otherwise it is some specified weighting factor. This weighting factor can be used to give priority to a given task or task-processing

unit combination. In the case that $w_{i,j} = 1, \forall i, j$ this reduces to the throughput of the system in terms of instructions per second.

With knowledge of the number of instructions of task j along with its computational efficiency, it is possible to determine the task's execution time. That is, for a given task with which successful completion requires I_j instructions to be executed, one can define the execution time as the smallest value τ_j such that

$$I_j = \int_{\tau_j}^{\infty} \sum_{i=1}^{n} IPC_{i,j}(t) s_i(t) dt \tag{2.1}$$

holds true. In practice this becomes an extremely difficult quantity to determine due to system non-determinisms which cause $IPC_{i,j}(t)$ and even I_j to be stochastic quantities.

As stated earlier, during the course of execution, flow of computation will be subjected to numerous interruptions, which contribute to its execution time. To better understand the sources of interruptions, an experiment was carried out on an Intel SNB processor[1] in which the architectural sources of interrupts were monitored using the available on-chip performance counters [42]. The Firefox web browser was executed, along with the eleven most visited web pages from the BBench suite [43], including Amazon, BBC, CNN, Craigslist, eBay, ESPN, Google, MSN, Slashdot, Twitter, and YouTube. Because the performance of webpage rendering was the focus of this study, network delay was not included in this specific experiment[2]. The loading and rendering of the webpages was repeated 1,500 times, and the execution times

[1]The Intel SNB processor was selected over other platforms due to the availability of advanced performance counters to measure the numerous sources of delays. While the mobile platforms may have some performance counters, the available information is not sufficiently detailed as that given by the Intel SNB processor. Although the parameters of the distributions may change from platform to platform, the underlying distributions will not.

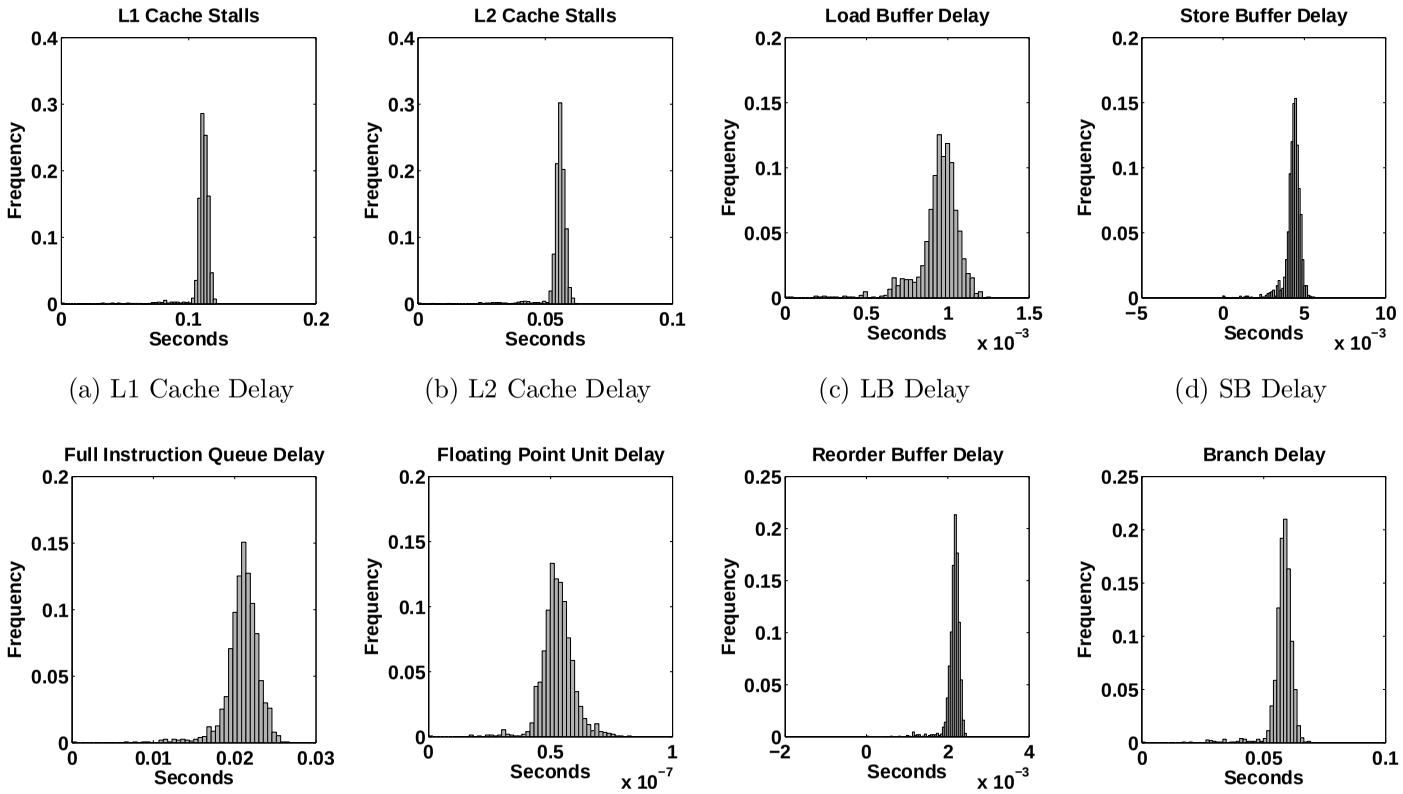
[2]It should be noted that network delay is not necessarily the dominating factor on mobile phones. A study from [44] showed that under a 2Mbps network connection halving the CPU frequency results

and the associated delays incurred in the processor and the memory subsystem were recorded. Figures 2.2(a-h) show the histograms of the durations of each monitored source of interruption. The data clearly demonstrates a substantial variation in the durations of different types of interruptions. In particular, the standard deviation of the total execution time (0.2117) is about 38% of the mean.

These interruptions to the execution flow can be viewed as intervals of idle periods, whose endpoints are random points in time, and whose lengths are random variables. Thus the total execution time may be viewed as the some of some minimum execution time (not random, but unknown) and the total duration of all the interruptions. Thus, QoS is said to follow sum distribution function $f_{\tau(s)}(x)$ which is, in part, parameterized by the operating frequency of the SoC. Chapter 3, proposes a standard form (the three-parameter gamma distribution) for single core applications and extends it to various multicore applications. Chapter 4 extends this further to accommodate any arbitrary distribution.

Since QoS should be considered a random quantity for mobile devices, a new perspective on energy management must be considered. For example, a common approach to balancing battery lifetime and performance is optimizing energy efficiency, also referred to as *performance-per-watt*, denoted by $\boldsymbol{E}\left(PPW(\boldsymbol{s})\right)$, where $\boldsymbol{E}(\cdot)$ denotes the expected value. PPW is the ratio of performance to power, and performance is simply the reciprocal of execution time. The measure of performance depends on a specific application, but is in general, a function of all the core frequencies $\boldsymbol{s}$. In the case of a video playback, $PPW(\boldsymbol{s})$ would denote the average number of frames rendered per second (fps) per Watt, and in the case of a web browser, it would be the number of webpages displayed per second per Watt. Hence, PPW denotes the

in a 50%-100% increase in the page load times, thus the bottleneck becomes the client CPU. To avoid the correlation between cores, all but one application processor core is disabled.



(a) L1 Cache Delay (b) L2 Cache Delay (c) LB Delay (d) SB Delay

(e) Full IQ Delay (f) FPU Delay (g) ROB Delay (h) Branch Delay

Figure 2.2: Distributions of the non-deterministic portions of sources of delay (Minimum delays subtracted).

number of items processed per joule of energy. Let Δ denote a deadline and $Q \in (0, 1)$ denote a lower bound on the likelihood of satisfying the given deadline.

2.2 Thermal Model

Due to circuit inefficiencies, executing any workload results in heat generation within the mobile SoC. In general, in order for the device to increase the performance output of an application, additional power consumption is required and by extension, a greater amount of heat is generated. Excessive thermal conditions may lead to decreased device reliability, or even permanent damage to the device. Additionally, mobile devices lack active cooling mechanisms and thus are far more susceptible to high thermal conditions. For this reason it is vital to understand the thermal characteristics of mobile devices.

One of the most classical methods for thermal models used for computer systems is based on the thermo-electrical analogy. This analogy suggests that the effects of heat storage and spreading can be equivalently represented as some circuit using resistors and capacitors. More precisely, consider a slab of thickness t and area A, which is heated on one side with uniform heat flux P as shown in Figure 2.3. Then the temperature difference across the sides of the slab due to heat conduction is given by $\Delta T = Pt/(kA)$ where k is the heat capacity of the slab. This equation is similar to Ohm's law ($V = IR$) where the thermal resistance is given by $R = t/(kA)$ and $I = P$. Additionally, according to the thermal energy equations developed by James Joule, the energy stored in the slab is equivalent to $E = \int Pdt = (\rho Atc)\Delta T$ where ρ and c denote the density and thermal capacity of the slab respectively. Once more, this relationship draws many parallels to the electrical charge storage equations for electrical capacitors ($Q = \int Idt = CV$) such that the thermal capacitance is given by $C = \rho Atc$.

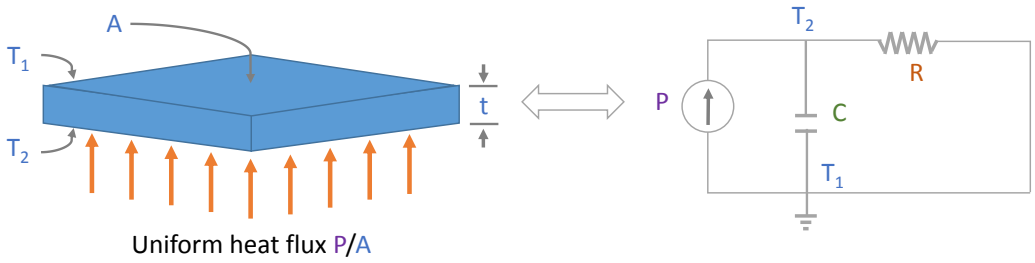


Figure 2.3: Electrical analogy for thermal conduction.

Perhaps the most well known example of thermo-electrical modeling is the HotSpot thermal model [45]. The granularity of the thermal model is defined by functional *blocks* which can be as fine-grain as a singular transistor or as course-grain as an entire chip or processor. Figure 2.4 shows the HotSpot thermal model for a typical four core desktop processor.

As discussed in more detail in the next section, the power dissipation of each block is dependent on the speed, voltage, and thermal state of the block, along with workload characteristics. Given this the vector of thermal profiles for each block, $\boldsymbol{T}(t)$, can be expressed using a state space model

$$\frac{d\boldsymbol{T}(t)}{dt} = -\boldsymbol{C}^{-1}\boldsymbol{G}\boldsymbol{T}(t) + \boldsymbol{C}^{-1}\boldsymbol{P}(\boldsymbol{s}, \boldsymbol{v}, \boldsymbol{T}, t) \tag{2.2}$$

where $\boldsymbol{P}$, $\boldsymbol{s}$, and $\boldsymbol{v}$ are vectors comprised of each block's total power dissipation, speed, and voltage at time t respectively. $\boldsymbol{G}$ and $\boldsymbol{C}$ are $N \times N$ matrices relating the thermal conductance and capacitance between any two pairs of the N blocks. This equation can be written more succinctly by substituting $\boldsymbol{B} = \boldsymbol{C}^{-1}$ and $\boldsymbol{A} = -\boldsymbol{B}\boldsymbol{G}$ thus

$$\frac{d\boldsymbol{T}(t)}{dt} = \boldsymbol{A}\boldsymbol{T}(t) + \boldsymbol{B}\boldsymbol{P}(\boldsymbol{s}, \boldsymbol{v}, \boldsymbol{T}, t). \tag{2.3}$$

It is important to note that $\boldsymbol{P}$ represents the total power of each block (i.e. the sum of the dynamic power $\boldsymbol{P}_{dyn}$ and the leakage power $\boldsymbol{P}_{lkg}$. The following section will detail a mechanistic method of determining these quantities.

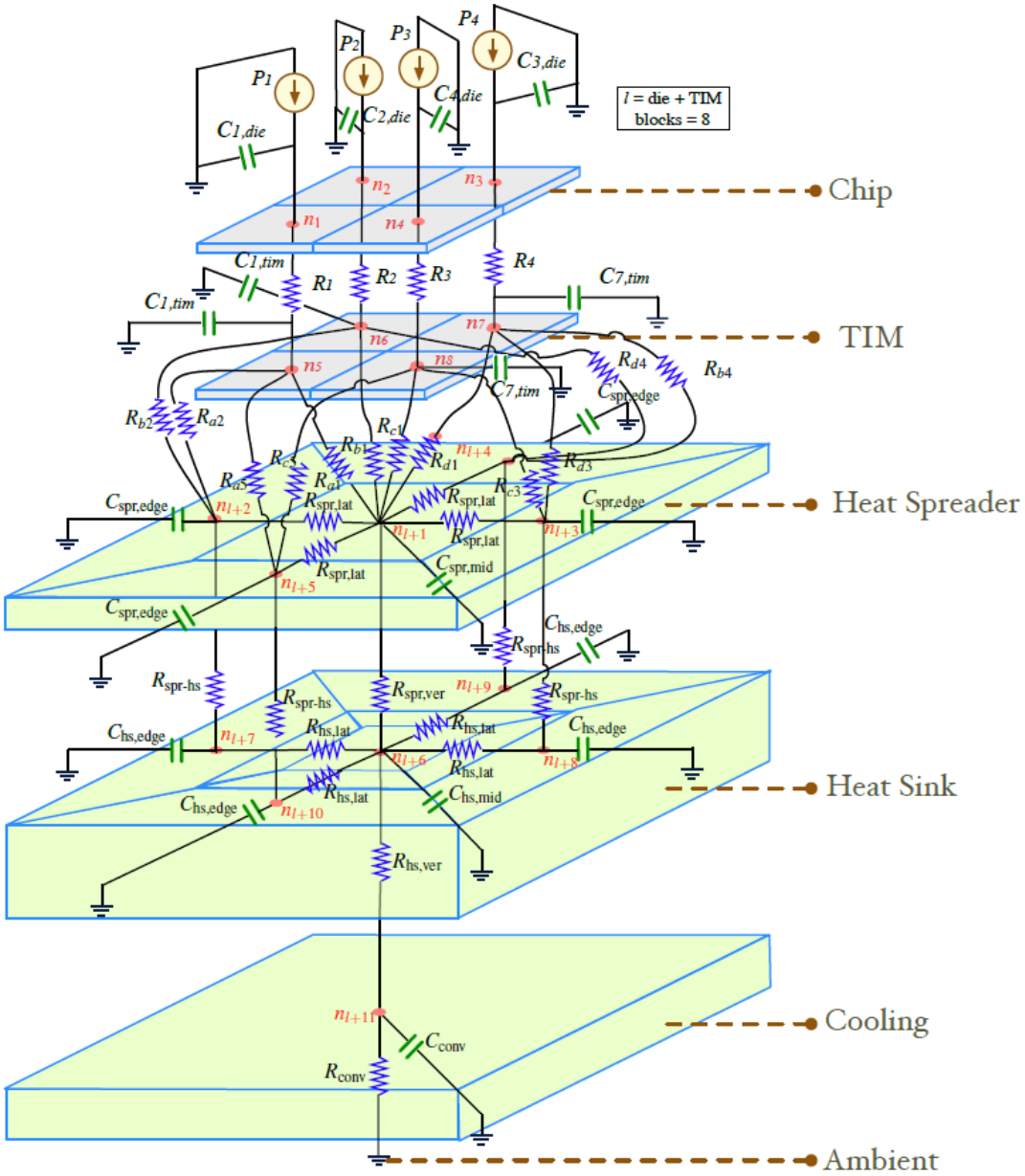


Figure 2.4: HotSpot thermal model for a four core processor. Figure copied from [46].

2.3 Power Model

Power modeling, simulation, and prediction of computing workloads have been studied extensively in the past [47, 19, 48, 49, 50, 51]. In these prior works, the power consumption of a singular block within the processor is typically model as

$$P_b(t) = P_{dyn,b}(t) + P_{lkg,b}(t) + P_{base,b}(t), \tag{2.4}$$

where $P_{dyn,b}(t)$, $P_{lkg,}(t)$, $P_{base,b}(t)$ are functions describing the dynamic, temperature-dependent leakage, and baseline or static power components respectively for a given block, b. The total system power consumption can then be represented as a sum of the power consumptions for all $\mathcal{B}$ blocks on the system.

$$P_{sys}(t) = \sum_{b \in \mathcal{B}} P_b(t). \tag{2.5}$$

2.3.1 Dynamic Power

Dynamic power is known to vary linearly with clock frequency, $s_b(t)$ and quadratically with operating voltage, $v_b(t)$. That is, the dynamic power of block b at time t can be modeled by the following:

$$P_{dyn,b}(t) = a_b(t) s_b(t) v_b(t)^2, \forall b, t \tag{2.6}$$

where $a_b(t)$ represents the activity factor of block b. In practice this value changes over time in relation to the workload characteristics. For example, a workload performing a large number of memory operations will exhibit a different power profile than a workload with purely ALU based operations. However, it is challenging to determine the form of $a_b(t)$ in practice, and more-so to predict the value of $a_b(t)$ in the future. Prior literature typically simplifies the issue by assuming a static form such as a

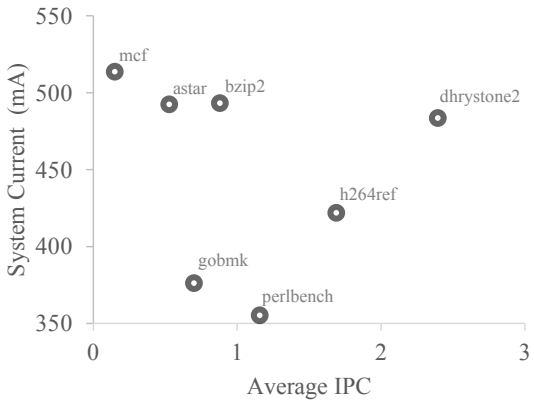


Figure 2.5: Dynamic CPU current draw vs. IPC for selected workloads. Battery voltage held constant at 4.33V.

constant value ($a_b(t) = A_b, \forall t$) or proportional to the instructions-per-cycle of the block ($a_b(t) = A_b IPC(t)$). However, such a simplistic form does not properly capture the variations between different workloads [48].

To illustrate this deficiency, the power and performance characteristics of several workloads were examined running on an actual mobile device – the Google Pixel housing the Qualcomm MSM8996 Pro system on chip. The frequency was fixed at 1.2864GHz and the operating temperatures of the cores where verified to be approximately the same (between 40°C and 45°C). The system current draw is monitored and recorded at a rate of 5000 samples per second using the Monsoon Power Monitor and averaged the samples over the course of the workloads execution. Additionally, the instructions per cycle were recorded using on-chip performance counters. Figure 2.5 shows the relationship between IPC and current draw. From this it can clearly be seen that dynamic power is not simply proportional to IPC alone and thus a more sophisticated prediction mechanism is needed to accurately capture the workload's power characteristics. In Chapter 4, a novel methodology is presented to represent $a(t)$ as a stochastic value based on workload characteristics visible via hardware and software performance counters and on chip-temperature sensors.

2.3.2 Leakage Power

The second component of power consumption, leakage power, is exponentially dependent on the temperature conditions of the SoC. The exact equation to represent leakage power is hard to derive analytically; however, many works have attempted to produce a surrogate approximation model that can be fitted using simulation. In the case of 65nm technology, the authors of [52] propose the following empirical model:

$$P_{lkg,b}(t) = k_b^1 v_b(t) T_b^2(t) e^{\frac{\alpha_b v_b(t)+\beta_b}{T_b(t)}} + k_b^2 e^{(\gamma_b v_b(t)+\delta_b)}, \forall b, t. \tag{2.7}$$

where k_b^1, k_b^2, α_b, β_b, γ_b, and δ_b are parameters that depend on circuit topology, size, technology and design.

Along with the non-linearity of this leakage model and the difficulty of fitting the model parameters, a major limitation of (5.5) is the cyclical relationship with (2.3) in regards to temperature. In order to solve (2.3) and (5.5) directly, numerical solutions for non-linear analysis are necessary therefore making temperature and power prediction computationally inefficient and difficult to use in practice. Alternatively, one may de-couple the thermal dependency by representing leakage power as a piecewise-linear model in temperature and voltage [12]

$$P_{b,lkg}(t) = P_{b,lkg}^0 + G_b^T T_b(t) + k_b^v v_b(t), \forall b, t \tag{2.8}$$

where G_b^T is the temperature coefficient associated with temperature T and k_b^v is the voltage coefficient associated with voltage v. $P_{b,lkg}^0$ represents the leakage power for block b such that $T_b = 0$ and $v_b = 0$ (i.e. the ambient temperature and the minimum voltage). Therefore, (2.3) can be rewritten as

$$\frac{d\boldsymbol{T}(t)}{dt} = \hat{\boldsymbol{A}}\boldsymbol{T}(t) + \boldsymbol{B}\hat{\boldsymbol{P}}(\boldsymbol{s}, \boldsymbol{v}, \boldsymbol{T}, t) \tag{2.9}$$

where

$$\hat{\boldsymbol{A}} = \boldsymbol{A} + \hat{\boldsymbol{B}}\boldsymbol{G_T}, \tag{2.10}$$

$$\hat{\boldsymbol{P}}(\boldsymbol{s}, \boldsymbol{v}, t) = \boldsymbol{P}(\boldsymbol{s}, \boldsymbol{v}, \boldsymbol{T}, t) - \boldsymbol{G_T}\boldsymbol{T}(t) \tag{2.11}$$

and $\boldsymbol{G_T}$ is a vector comprised of each block's G_b^T value. Thus the cyclical dependency between power and temperature is removed. Evaluation of the temperature and leakage power can be done efficiently as both are represented using a set of linear equations.

Chapter 3

AN OFFLINE APPROACH TO IMPROVING SMARTPHONE USER EXPERIENCE BY BALANCING PERFORMANCE AND ENERGY WITH PROBABILISTIC QOS GUARANTEE

As discussed in Chapter 1, user satisfaction is pivotal to the success of a mobile application. It has been shown that 49% of users would abandon a web-based application if it failed to load within 10 seconds. At the same time, it is imperative to maximize energy efficiency to ensure maximum usage of the limited energy source available to smartphones. In Chapter 2 the concept of non-deterministic execution times was introduced. This is an important factor to consider, that has been previously neglected by prior literature. When considering system non-determinisms, modeling frameworks must take special considerations to the various stochastic quantities. Additionally, this changes the nature of the objective function and the constraints of the underlying optimization problem. In this chapter, an approach is presented for the optimal energy control of mobile applications running on modern smartphone devices, focusing on the need to ensure a specified level of user satisfaction. The proposed approach uses a novel framework in which statistical models are used to address both single and multi-stage applications. These models are then used for offline system analysis and in the formulation of an optimization problem, the solution to which is a static, lightweight controller that optimizes energy efficiency of mobile applications, subject to constraints on the likelihood that the application execution time meets a given deadline. The proposed models and corresponding optimization method are validated using three common mobile applications running on a real Qualcomm Snapdragon 8074 mobile chipset. The results show that the proposed statistical estimates

of application execution times are within 99.34% of the measured values. Additionally, on the actual Qualcomm Snapdragon 8074 mobile chipset, the proposed control scheme achieves a 29% power savings over commonly-used Linux governors while maintaining an average web page load time of 2 seconds with a likelihood of 90%.

3.1 Problem Background: The Need For QoS Aware Control of Mobile Devices

Over the past decade, a large body of research has been published on optimizing energy efficiency [53, 54, 10, 8, 55, 56, 57, 58, 59, 60]. However, most, if not all of the work, have assumed that the underlying quantities (e.g., execution times) are deterministic. In [11], Isci et al. considered voltage and frequency (DVFS) as a means to dynamically tune a processor's power consumption based on an application's computation and memory characteristics. Others considered thermal constraints and leakage power when optimizing energy efficiency [61, 62, 63, 64, 14, 17].

While most of the prior works focus on optimizing system energy efficiency for the application domain of chip-multiprocessors (CMPs), a few more recent works have proposed energy efficiency optimization algorithms for user-centric, interactive smartphone applications. Egilmez et al. [15] and Singla et al. [16] proposed temperature-aware energy efficiency optimization while Zhu et al. [18] focused on meeting interactive application QoS. Egilmez et al. [15] specifically aimed at user satisfaction through DVFS-based control knobs to maintain a low and comfortable temperature. Singla et al. [16] developed power and thermal models for a modern mobile platform and proposed a closed loop thermal control to adjust processor frequencies. Zhu et al. [18] proposed eQoS to improve the energy efficiency of web browsers for smartphones while meeting certain QoS constraints.

Although the problems addressed in the existing literature are complex in and of themselves, they have assumed that the execution time of an application is *determin-*

istic. In reality, execution times vary substantially, depending on the continuously varying states of the system. This is due to factors such as operating system (OS) preemption events, page faults, system interrupts, processor pipeline stalls, branch prediction, cache misses, context switches, data input variations, network delays, etc. While considering only the average case may result in higher energy efficiency on average, not accounting for the variations in execution time can lead to situations where user satisfaction is poor. For these reasons, the execution time of an application must be modeled as a stochastic value, whose distribution ideally should depend on the control variables (voltage and frequency) and other parameters that capture the characteristics of the data and the computing devices. With this paradigm change, the constraints in the optimization problem can no longer be simply viewed as being satisfied or not being satisfied, but instead must be expressed as a *likelihood* of being satisfied.

3.2 Problem Statement and Novelty

In this dissertation, a novel approach is presented which aims to improve mobile user experience by balancing performance and energy with a probabilistic guarantee of QoS. The approach is based on modeling the execution time of applications as stochastic quantities. Each application consists of some number of computational segments or basic *units of computation* (UoC), with each UoC being mapped to a core. Variations in execution times are due to different sources of interruptions in the normal flow of computation, as well as intrinsic variations in the complexity of the data being processed. Note that the notion of a UoC varies among applications. First, a hypothesize is presented providing a closed form representation (distribution function) of the execution times of UoCs, and then experimentally validated. The hypothesized model parameters are functions of the control variables (e.g., voltage

and frequency), as well as characteristics of the data being processed (e.g., complexity of the web page component being processed); a necessary condition for application-aware dynamic energy management.

The model of the total execution time of an application depends on how its constituent UoCs are mapped to the cores. For example, some applications have independent UoCs executing in parallel on separate cores, while in others, the UoCs interact. This leads to different ways of modeling the total execution time for different applications. The proposed statistical models are used in the formulation of an optimization problem, the solution to which is a static, lightweight controller that optimizes energy efficiency of mobile applications, subject to constraints on the *likelihood* that the application execution time meets given deadlines.

The statistical models for single and multi-stage applications and the design of the controller are demonstrated on three commonly used smartphone applications: web browsing, image similarity search, and video playback. This is performed on two different real platforms: an Intel Quad-Core Sandy-Bridge (SNB) processor and a Qualcomm Snapdragon 8074 chipset-based mobile device. The results show that the proposed statistical estimates of application execution times are within 99.34% of the measured values. Next, an application case study is presented for which these estimates are applied to select the frequency setting when accounting for device energy efficiency and an execution time deadline. Our proposed optimization achieved a 29% power savings over commonly-used Linux governors while maintaining an average web page load time of 2 seconds with a likelihood of 90%—a quality that the Linux governors do not consider. This dissertation achieves similar findings for the other mobile applications: image similarity search and video playback.

In summary, this dissertation proposes an accurate execution time model that provides a quantative measure of the likelihood of meeting a deadline – an overlooked

dimension of execution time modeling. To demonstrate the importance of the developed model, a model is built for a modern, high-performance real smartphone device to improve user experience while providing a probabilistic guarantee on QoS. By doing so, on average the QoS is improved by 2.18**x** at a cost of a 25% degradation in power efficiency when compared to the optimal energy efficiency setting.

3.3 Problem Description

In this section, some basic terminologies are introduced and a precise statement of the optimization problem is made. Then, two different scenarios are described which fit into the general optimization framework, but with different models of execution time.

An application consists of a collection of basic UoCs, each of which is mapped to a core. For instance, a web browser receives an HTML document that consists of a number of URIs (Uniform Resource Index). The URIs are queued, processed and ultimately displayed on screen. Therefore, in this instance, a UoC would simply be a URI or a collection of URIs. In a video playback, a UoC refers to a single video frame that undergoes several stages of processing before it is displayed.

The execution time of a UoC on a single core is a random variable τ, with a distribution function $F_\tau(x \mid s) = \text{Prob}(\tau \leq x \mid s)$, where s represents the core frequency. In general, the distribution function can also include a set of *uncontrollable* parameters that affect the execution time, reflecting the complexity of the data being processed. Examples of such parameters would be number of URIs or the complexity of URIs in a webpage as well as the operating frequencies of each processor core. Note that in general the core voltage setting is determined by the frequency, and hence this dissertation considers only the core frequency as the main control variable.

This dissertation limits its study to the situation where only one foreground ap-

plication is being executed at a given time. Currently, this seems to be the most common application use case on smartphones; however, $F_\tau(x \mid s)$ can be extended to handle multiple applications with enough training to expose the variability caused by the interfering applications. For example in [65] (deterministic) the execution time is parameterized as a function of the memory interference due to concurrently running applications. Similarly, one could define the execution time pdf parameters to be functions of the memory interference.

A given dataset (e.g., a webpage or a video frame) is a collection of UoCs, $U_1, U_2, \cdots, U_n$, whose corresponding individual execution times are independent random variables $\tau_1, \tau_2 \cdots \tau_n$, with distribution function F_τ. In a CMP with n cores, let $Z_n(\boldsymbol{s})$ denote the total execution time of an application, where $\boldsymbol{s}$ denotes the vector of core frequencies. The relation between $Z_n(\boldsymbol{s})$, and the execution time of the UoCs that make up the application depends on the application and how they are mapped onto the cores. For instance, if all its UoCs execute concurrently and independently, then $Z_n(s)$ would be the maximum of the individual execution times, whereas in the case of a single core $Z_n(\boldsymbol{s})$ would be the sum of the individual execution times.

The objective function to be maximized is energy efficiency, also referred to as *performance-per-watt*, denoted by $\boldsymbol{E}\left(PPW(\boldsymbol{s})\right)$, where $\boldsymbol{E}(\cdot)$ denotes the expected value. PPW is the ratio of the performance to power, and performance is simply the reciprocal of the execution time. The measure of performance depends on the specific application, but is in general, a function of all the core frequencies $\boldsymbol{s}$. In the case of a video playback, $PPW(\boldsymbol{s})$ would denote the average number of fps per Watt, and in the case of a web browser, it would be the number of webpages displayed per second per Watt. Hence, PPW denotes the number of items processed per joule of energy. Let Δ denote a deadline and $Q \in (0, 1)$ denote a lower bound on the likelihood of satisfying the given deadline. Then, with these notations, the optimization problem

can be stated as follows:

$$\max_{\mathbf{s}} \quad \boldsymbol{E}\left(PPW(\mathbf{s})\right) \tag{3.1}$$

$$\text{s.t.} \quad F_{Z_n(s)}(\Delta|\mathbf{s}) = \text{Prob}(Z_n(\boldsymbol{s}) \leq \Delta) \geq Q. \tag{3.2}$$

A naive approach to solving the above optimization problem would be to experimentally evaluate the application program at all combinations of per-core frequencies. Given the large number of available frequency settings in modern processors and the increasing number of cores in the application processor, a brute force evaluation for identifying the optimal frequency combination for all cores is infeasible. For example, there are fourteen available frequency settings per core in the Qualcomm Snapdragon chipset (ranging from 300MHz to 2.15GHz). For an Octa-core application processor, it requires 14^8 experiments to determine the frequency combination that maximizes system energy efficiency for any application. For this reason, the more sophisticated modeling mechanisms presented herein, are needed. The proposed approach addresses two distinct scenarios, one where all the cores are operating concurrently and independently, and the other where there are interactions among the cores. These two scenarios require a different approach to modeling the execution times.

While this dissertation only examined core frequencies due to the lack of control over other SoC components (GPU, DSP, etc.), incorporating accelerators will require no changes to the models presented. It is only necessary to monitor the runtime of the code segment(s) offloaded and develop probability distributions as a function of their controls.

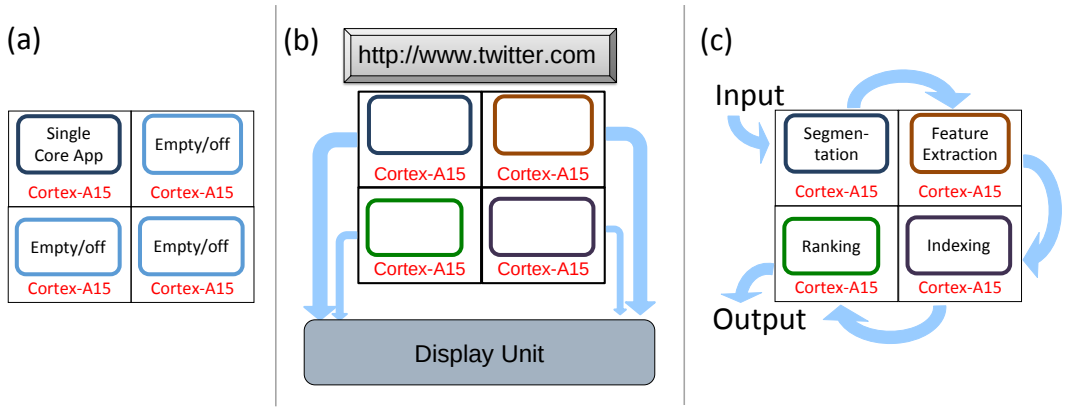


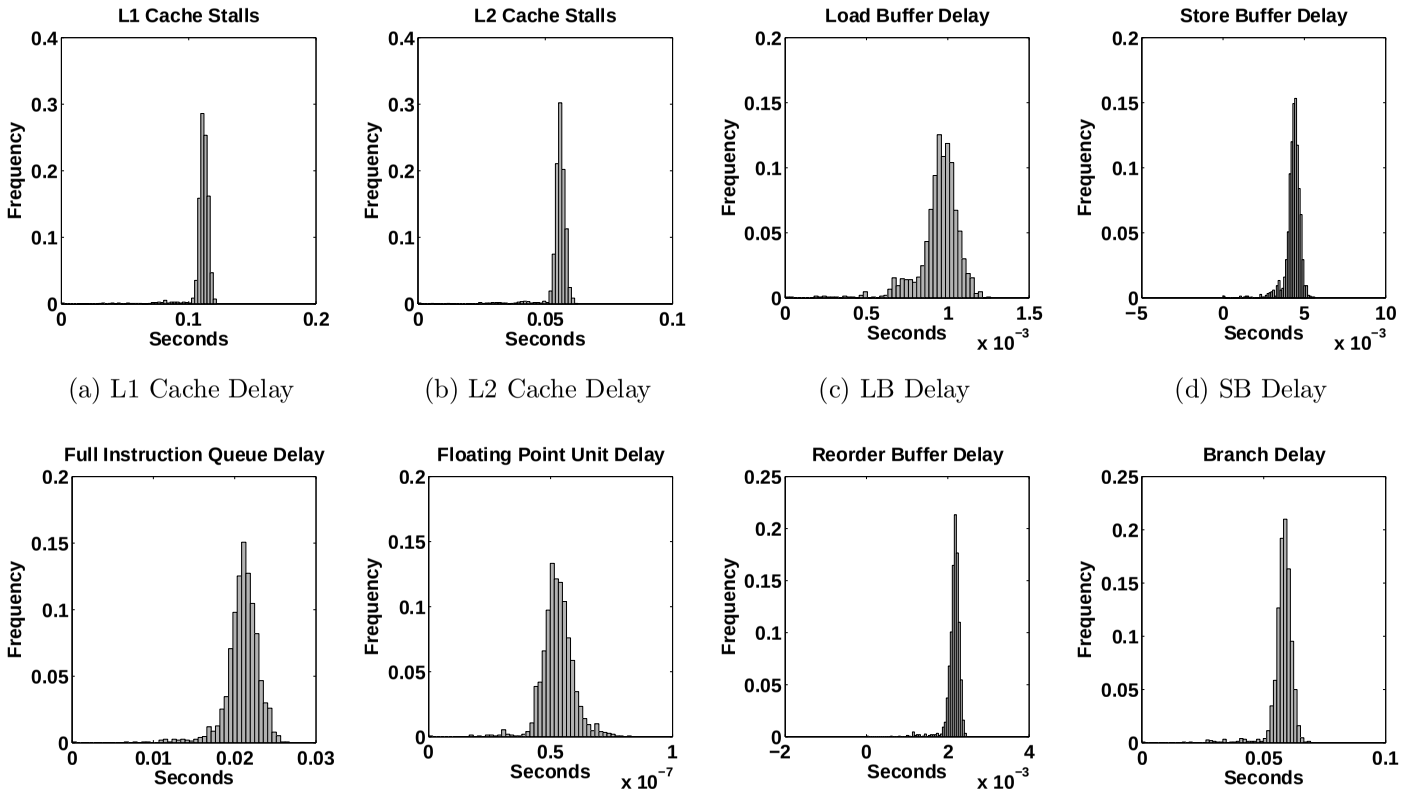
Figure 3.1: Illustrations of three types of application to core mappings: (a) single-thread, single core; (b) parallel computations, independent cores; (c) parallel computations, interacting cores.

3.4 Execution Time Models

3.4.1 Computations on a Single Core

To begin this section, evidence will be provided that supports the need to model execution times as random variables. As stated earlier, during the course of execution, flow of computation will be subjected to numerous interruptions, which contribute to its execution time. To better understand the sources of interruptions, consider the experiment briefly discussed in Chapter 2. That is, an experiment was conducted on the Intel SNB processor[1] in which the architectural sources of interrupts were monitored using the available performance counters on the processor [42] (the complete methodology is detailed in Section 3.6). It is important to note, that the severity of each source of delay can vary greatly. The focus of this dissertation is on the performance of web page rendering (web pages are loaded offline) and thus network delay is not

[1]The Intel SNB processor was selected over other platforms due to the availability of advanced performance counters to measure the numerous sources of delays. While the mobile platforms may have some performance counters, the available information is not sufficiently detailed as that given by the Intel SNB processor. Although the parameters of the distributions may change from platform to platform, the underlying distributions will not.



(a) L1 Cache Delay (b) L2 Cache Delay (c) LB Delay (d) SB Delay

(e) Full IQ Delay (f) FPU Delay (g) ROB Delay (h) Branch Delay

Figure 3.2: Distributions of the non-deterministic portions of sources of delay (Minimum delays subtracted).

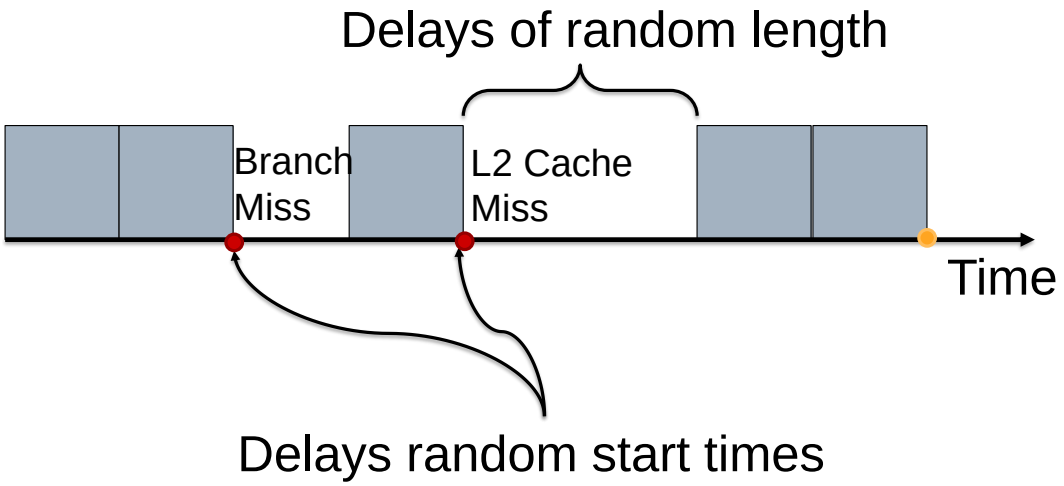


Figure 3.3: An illustration of the effect of interruptions to the execution flow of an application.

included. That said, network delay is not necessarily the dominating factor on mobile phones. A study from [44] showed that under a 2Mbps network connection halving the CPU frequency results in a 50%-100% increase in the page load times, thus the bottleneck becomes the client CPU. To avoid the correlation between cores, all but one application processor core is disabled. The Firefox web browser was executed, along with the eleven most visited web pages from the BBench suite [43], including Amazon, BBC, CNN, Craigslist, eBay, ESPN, Google, MSN, Slashdot, Twitter, and YouTube. The various sources of delays were monitored during the course of the eleven webpage loads. This was repeated 1,500 times, and the execution times and the associated delays incurred in the processor and the memory subsystem were recorded. Table 3.1 shows the statistical information gathered from the experiment, and Figures 3.2(a-h) show the histograms of the durations of each of the monitored sources of interruptions. The data clearly demonstrates a substantial variations in the durations of different types of interruptions. In particular, the standard deviation of the total execution time (0.2117) is about 38% of the mean.

The interruptions to the execution flow can be viewed as intervals of idle pe-

Table 3.1: Summary of Sources of Delays

	Mean	Std. Deviation	Min
Execution Time (s)	0.5532	0.2117	0.2862
Source of delay	**Mean**	**Std. Deviation**	**Min**
L2 Cache Stalls (s)	0.0828	0.0048	0.0276
L1 Cache Stalls (s)	0.1651	0.0095	0.0546
LB Stalls (s)	0.0012	0.0001	0.0003
SB Stalls (s)	0.0063	0.0005	0.0020
Full IQ Stalls (s)	0.0326	0.0021	0.0116
FPU Stalls (s)	8.80E-8	6.71E-9	3.51E-8
ROB Stalls (s)	0.0008	0.0002	0.0007
Branch Stalls (s)	0.0858	0.0049	0.0280

riods, whose endpoints are random points in time, and whose lengths are random variables. Thus the total execution time may be viewed as the sum of the minimum execution time (not random, but unknown) and the total duration of all the interruptions. A model for sum of random interval lengths often used in the literature is the Gamma distribution [66]. Figure 3.4 shows the histogram of the total execution of the web browser compared to a 3-parameter Gamma distribution. It shows that the Gamma distribution provides an adequate model to explain the variations in the total execution time for this application on a single core. Note that the dependence of the probability distribution function (pdf) on the core frequency will be modeled by relating it to the parameters of the Gamma distribution.

The pdf of the Gamma is given by

$$\begin{aligned} f_{X(s)}(x) &= \text{Gamma}(x; K(s), \theta(s), \lambda(s)) \\ &= \frac{1}{\theta(s)\Gamma(K(s))} \left(\frac{x - \lambda(s)}{\theta(s)} \right)^{K(s)-1}, \end{aligned} \tag{3.3}$$

s is the core frequency, $K(s)$ is the shape parameter, $\theta(s)$ is the scale parameter, and $\lambda(s)$ is the left endpoint or minimum value of $X(s)$. The mean $\mu(s)$ and variance $\sigma^2(s)$ of the distribution are given by: $\mu(s) = K(s)\theta(s)$ and $\sigma^2(s) = K(s)\theta^2(s)$.

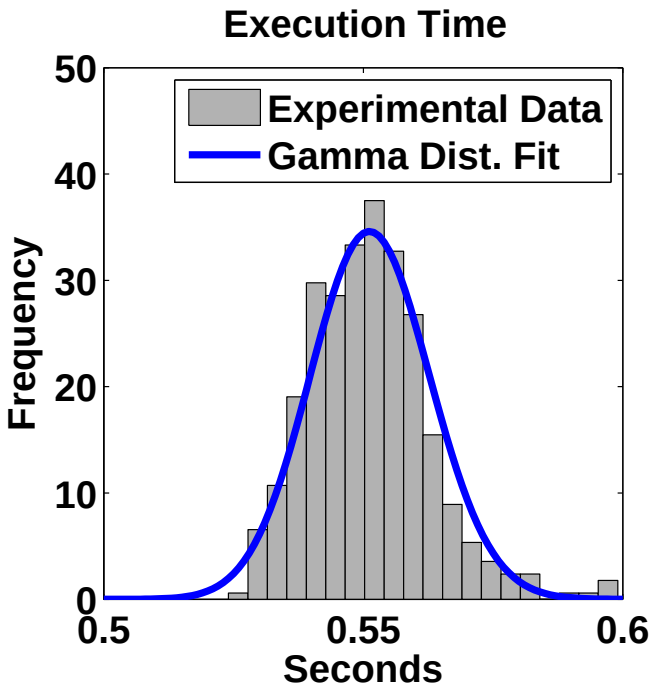


Figure 3.4: Histogram of execution times of an application running on a single core, compared to a pdf of a 3-parameter Gamma pdf.

It is important to note that the parameters $K(s)$, $\theta(s)$, and $\lambda(s)$ are modeled as functions of the core frequencies s. This provides a simple and effective means to relate the distribution function of the execution times to the core frequencies. In addition, it is also possible to make K and θ depend on application specific parameters. For instance, for web browsing we could include quantifiable characteristics of the web page such as the number of URIs and the types of objects in the URIs that affect the execution time. In this way, the execution time of a browser in processing a web page can be made sensitive to the complexity of the webpage.

3.4.2 Parallel Computations, Independent Cores

In this section, the execution time model of an application in which the UoCs are executed in parallel, and independently on multiple cores is described. Although the description is based on a specific multi-threaded web browser, it is easily made

applicable to other browsers and data parallel applications.

To load a web page, a browser must process a collection of URI elements. The web browser analyzes the collection of URI elements in a page and determines a subset of elements to be serviced by a secondary instance of the browser, assuming it deems the subset of elements to be "sufficiently complex". The servicing of this subset of URIs mapped to an application processor core denotes a unique UoC with its own execution time distribution. This process can be repeated until all URI elements are assigned a UoC. In practice the web pages of most web sites are partitioned to 1 or 3 UoCs.

With this view, the total time to load a page will be the latest completion time among all the UoCs. Thus if a page results in N UoCs, each mapped to a core, and the execution time of UoC n is a random variable τ_n, whose distribution function at a given core frequency s_n is given by (3.3), the total execution time will be $Z_N(\boldsymbol{s}) = max_{n=1}^{N}\tau_n(s_n)$, where s_n is the core frequency at which UoC n is processed. The distribution function of $Z_N(\boldsymbol{s})$ is given by

$$F_{Z(\boldsymbol{s})}(z) = \prod_{n=1}^{N} F_{\tau_n(s_n)}(z). \tag{3.4}$$

3.4.3 Parallel Computations, Interacting Cores

The previous sections described how the execution time of independent UoCs can be accurately modeled as simple functions of one or more independent random variables. While this is perfectly suitable to model many applications, there are many others in which UoCs possess a pipelined data flow structure such as *image similarity search* and *video playback*. In this section, it is described how to model the execution time of a collection of UoCs constituting an application, running in parallel on a network of interacting cores, where the network is a *cascade* of stages.

The naive approach is to model the total execution time of the application as

the sum of execution times of the stages. This leads to very pessimistic results (demonstrated in Section 4.5, see Figure 3.7) due to the fact that it doesn't accurately model the temporal dependencies among stages. A more appropriate model is the mathematical framework of *Network Calculus* [32] that can be used to describe the computation as flows of data through a network. With nodes exhibiting statistical variations in their execution times, the flows, which are represented by functions of time, are now *stochastic processes* (as opposed to simple random variables). Here it is described how the combination of network calculus and probability calculus can be used to predict the execution time of parallel interacting tasks.

In its original formulation, Network Calculus (NC) was aimed at modeling deterministic queuing systems for computer networks and is analogous to system theory used in circuit analysis and other domains. It captures the intricacies associated with buffering systems and pipelining in networked systems. The fundamental distinguishing feature of NC is the use of *min-plus* algebra as the basis for its calculations. As such, data flows are modeled as *non-decreasing* functions of time.

NC works on the notion of an observable node or *system*, $\mathcal{S}$, and derives properties (e.g., backlog, delay, etc.) based on the flow of data entering, exiting, and being serviced by the node. $\mathcal{S}$ in the present case is a cascade of one or more nodes. The flows are represented by the non-decreasing functions $I(t)$, $O(t)$, and $S(t)$ which are defined as follows.

- $I(t)$ – the total amount of data which has entered $\mathcal{S}$ during the interval $[0, t)$.
- $O(t)$ – the total amount of data which has exited $\mathcal{S}$ during the interval $[0, t)$.
- $S(t)$ – the *service curve* of system $\mathcal{S}$. This curve represents a lower bound on the total amount of data which could be serviced during the interval $[0, t)$.

The *goal* of this section is to provide a method to compute the pdf of $O(t)$, the throughput of a system, denoted by $f_{O(t)}(x)$. In the case of a pipelined computation

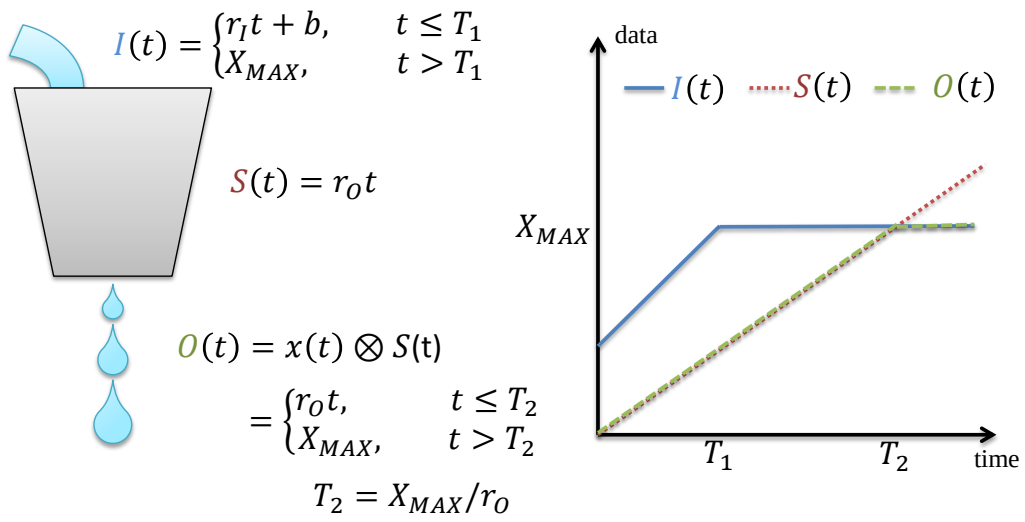


Figure 3.5: A classical network calculus example, the leaky bucket. The leaky bucket drains water via a hole in the bottom at a constant rate, r_O (i.e. the service characteristics of the system). The system receives an input flow of $r_I t + b$ until time T at which time no more flow is received. The output flow can therefore be calculated via min-plus convolution of the service curve and the input curve.

flow, this would be the pdf of the output curve of the last stage. The output flow of a system with a given service curve and input flow is given by

$$O(t) = I(t) \otimes S(t), \tag{3.5}$$

where $\otimes$ represents the min-plus convolution operation, defined by

$$O(t) = \inf_{0 \le \tau \le t} (I(s) + S(t - \tau)). \tag{3.6}$$

Figure 3.5 illustrates a classical example of this operation.

NC provides an algebraic representation of concatenation of simple systems to form a complex networked system. As in system theory, this greatly simplifies the calculation of flows between multiple connected systems. Consider the situation in which two systems $\mathcal{S}_1$ and $\mathcal{S}_2$ possess service curves $S_1(t)$ and $S_2(t)$ respectively. Additionally, $\mathcal{S}_1$ has input flow $I_1(t)$ and output flow $O_1(t)$ while $\mathcal{S}_2$ has input flow

$I_2(t)$ and output flow $O_2(t)$. The outflow of the cascade of $\mathcal{S}_1$ and $\mathcal{S}_2$ is given by

$$O_2(t) = I_1(t) \otimes S_1(t) \otimes S_2(t). \tag{3.7}$$

Distribution of the Service Process S(t)

In deterministic systems, a service process is a monotonic function which describes the potential cumulative output of the system given that the system is never starved for input. In other words, it is akin to the impulse response function in system theory. When extending into systems with non-determinism, a single service curve no longer represents the system. More precisely, it is a stochastic process, which is an infinite, non-denumerable set of service curves, as illustrated in Figure 3.6. At each fixed time t, the sampled space of outcomes are the points on all the curves at time t, representing the random variable $S(t)$. The objective now is to compute the distribution of the output variable $O(t)$ given by Equation 3.6, in terms of the input random variables $I(t)$ and the service random variable $S(t)$. One may view this as pdfs being propagated through the network to finally compute the pdf of the number of UoCs processed at the system output for each t.

Let $S(t \mid s)$ denote the service curve of some stage, which represents the number of UoCs processed in the interval $[0, t)$. It is assumed that a stage is synonymous with a core, whose frequency is s. Let $T_n(s)$ denote the time to process n UoCs by a stage. $T_n(s)$ is equal to the sum of the times to process each UoC. Then using the previous notation, let $X(s)$ be the random variable that represents the time to process one UoC by the given stage. The pdf of $X(s)$ is given by Equation (3.3). The the pdf of $T_n(s)$ is then given by

$$f_{T_n(s)}(t \mid n, s) = \underbrace{f_{X(s)} * f_{X(s)} * \ldots * f_{X(s)}}_{n\text{-fold convolution}}. \tag{3.8}$$

Distribution of the Output Process O(t)

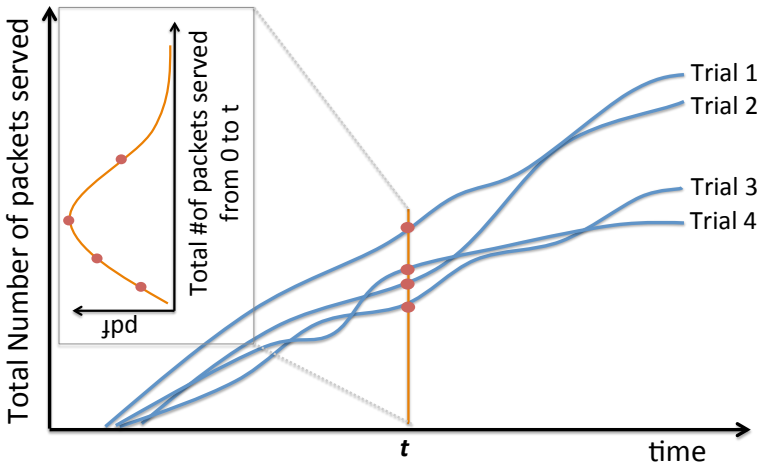


Figure 3.6: An illustration of Service curve $S(t)$ being a stochastic process. Each trial represents the algorithm being executed under the same input and system controls; however due to random system variations, the service times can vary, thus the observed output process can vary from trial to trial. At each t, the set of points on all the possible (infinite) curves represents the sample space of the random variable $S(t)$.

Consider the situation of a single UoC running on a core at frequency s, with an input flow, $I(t)$, service curve, $S(t \mid s)$, and output flow, $O(t \mid s)$. The computation of $O(t \mid s)$ given by Equation 3.6 is approximated by partitioning the time interval into a discrete set of time points $(0, t_1, t_2, \ldots t_r = t)$.

$$\begin{aligned} O(t \mid s) &= \min_{\tau \in (0, t_1, t_2, \ldots t_r)} I(\tau) + S(t - \tau \mid s) \\ &= \min \{ I(0) + S(t), I(t_1) + S(t - t_1 \mid s), \\ &\cdots, I(t_{i-1}) + S(t - t_{i-1} \mid s) \} \end{aligned} \tag{3.9}$$

The density of each sum within the min, i.e. $I(t_k) + S(t - t_k \mid s)$, is computed by the convolution of the individual densities. The density of the minimum of some n random variables, is approximated by iteratively computing the density of the minimum of

pairs of random variables, which is given by

$$f_{min(U,V)}(x) = f_U(x) + f_V(x) \\ - f_U(x)F_V(x) - F_U(x)f_V(x). \tag{3.10}$$

For an N stage pipeline, the distribution of the output process of the Nth stage, namely $O_N(t \mid \boldsymbol{s})$, is of interest. Equation 3.9 is expanded, with the output process of the stage i serving as the input process for stage $i+1$. Then the resulting expression will be the minimum among a potentially large number of service curves, along with the distribution of the initial $I(t)$, which is assumed to be known. The distribution function of such a minimum has to be computed numerically, using the approximation given in Equation 3.10.

In summary, estimation of execution times is done by numerically computing its distribution using Equation 3.3 or computing $f_{O_N(t)}(x \mid \boldsymbol{s})$ using the approximation given in Equation 3.10, depending on whether it is a collection of independent UoCs running on cores or a pipeline computation.

3.5 Power Model

A vital part of the optimization process is accurate power models. This is discussed in Chapter 2; however, is summarized below for completeness' sake. Power is represented as the sum of the dynamic power P_{dyn}, the leakage power P_{lkg} and static system power P_{stc}. While P_{stc} represents a constant offset, the dynamic power varies linearly with the clock frequency, as a circuit is operated only when the clock is high, while dynamic power varies quadratically with the voltage, as power of a transistor is the product of transistor current and voltage, and current of a transistor is also a function of voltage. The components of the dynamic power vector are expressed as

$$P_{dyn,c,b}(t) = P^{max}_{dyn,c,b}(t)s_c(t)v_c^2(t), \forall c, b, t \tag{3.11}$$

where $P^{max}_{dyn,c,b}$ is the dynamic power dissipated by block b of core c when the core is at the maximum speed and voltage. $P^{max}_{dyn,c,b}$ is obtained by profiling the time-varying power consumption of the task to be run on core c.

The leakage power is known to have exponential dependence on the die temperature and supply voltage. The exact equation is hard to derive analytically. Hence it is usually derived based on data fitting of the simulated power values for various components like adders, multipliers, memories, etc. An example empirical equation for leakage power in 65 nm is given [52].

$$P_{lkg,c,b}(t) = k^1_{c,b} v_c(t) T^2_{c,b}(t) e^{\frac{\alpha_{c,b} v_c(t) + \beta_{c,b}}{T_{c,b}(t)}} + k^2_{c,b} e^{(\gamma_{c,b} v_c(t) + \delta_{c,b})}, \forall c, b, t. \tag{3.12}$$

$k^1_{c,b}$, $k^2_{c,b}$, $\alpha_{c,b}$, $\beta_{c,b}$, $\gamma_{c,b}$, and $\delta_{c,b}$ are parameters that depend on circuit topology, size, technology and design. It should be noted that (3.12) can be simplified to a piece-wise linear approximation as shown in [55]. Thus, the following model is used to estimate leakage power for each core/block on the processor.

$$P_{lkg,c,b}(t) = P_{lkg0,c,b} + G^T_{c,b} T_{c,b}(t) + k^v_{c,b} v_c(t), \forall c, b, t. \tag{3.13}$$

where $G^T_{c,b}$ and $k^v_{c,b}$ represent the temperature and the voltage coefficients. $P_{lkg0,c,b}$ represents the leakage power for block b in core c corresponding to $T_{c,b} = 0$ and $v_c = 0$ (i.e. the ambient temperature and minimum voltage).

3.6 Experimental Setup

The proposed models and methodology were evaluated on real platforms. In this section, the setup is presented for the experiments and the validation results for the execution time and power models, evaluated on real platforms.

3.6.1 Real-Device Experimental Platform

The experiments were conducted on two different platforms: the Intel Quad-Core SNB processor and a DragonBoard development board based on the Qualcomm Snapdragon 8074 chipset. The Intel SNB processor includes four cores, each of which has a private L1 instruction cache (32KB), a private L1 data cache (32KB), and a private unified L2 cache (256KB). All four cores share the last-level cache of 6144KB. The frequency settings available in the Intel SNB processor range from 0.8GHz to 2.1GHz in steps of 100MHz. Since this processor is not a mobile chipset, it is only used in the event when detailed performance counters are needed (see Section 3.4). On the other hand, the Qualcomm Snapdragon 8074 chipset, which is in the US versions of Samsung Galaxy S5 smartphones and many other modern smartphone devices, is used for all results presented in the evaluation section (Section 3.7). It has four 2.15GHz cores with independent frequency control. Each core hosts a private L0-cache (4KB) and L1-cache (16KB). All cores share a 2MB L2 cache. Since the Dragonboard offers no onboard power sensors, an NI DAQ unit was used with a 1MHz sampling rate for current and voltage measurements. The readings were taken after AC to DC conversion.

Table 3.2: Parameters for the DragonBoard Experimental Platform.

	Cortex-A15
Architecture	ARM v7, Krait
Frequency	0.3-2.15GHz
L0 Cache Size	4KB I & 4KB D
L1 Cache Size	16KB I & 16KB D
L2 Cache Size	2MB

The experimental platforms run a rooted Android 4.4 KitKat OS. The applications of interest *web browsing*[67], *image similarity search* [68], *video playback*[69], were written in C and cross-compiled on the host machine with the ARM-Android NDK toolchains [70]. The binary is pushed to the device and is launched from the host machine via the adb terminal.

3.6.2 Benchmark Applications

Three applications common to the mobile domain were evaluated. The first is *web browsing*. Following [71], a web page is viewed as a collection of elements (or URIs) which are then serviced by the browser's threads. Each thread is treated as an independent server and acts in parallel with other threads. The specific web browser was Firefox. Minor modifications were made to the source code in order to observe the service time of each web page and each URI, to compute the empirical distributions. The BBench 2.0 benchmark [43] was used as it contains a collection of the eleven most widely viewed webpages. It should be noted that this benchmark is limited to offline browsing due to the limitations of implementing timing analysis code into the web pages. Although not explored in this dissertation, it is possible to capture the effects of network delay by either (1) decreasing the value of δ by the expected value of the network delay or (2) model the network as UoC cascading into the remainder of the stages. Effectively this creates an arrival process for the web browser. The second approach is explored in more detail in [72]. That said, network delay is not necessarily the dominating factor on mobile phones.

The second application was *image similarity search*, which is an image similarity ranking algorithm from the PARSEC benchmark suite [68]. The algorithm is used for content-based similarity search of feature-rich data, such as images or videos, and is an important building block for many image recognition and augmented reality

Apps running on modern smartphones or Google Glass-like devices. The algorithm consists of six stages of processing. Given an image, *image similarity search* searches a database of images to find the closest match. After the image is loaded, it is first fragmented into segments based on its contents. Then the feature extraction stage assigns each fragment a numerical feature vector. Finally this feature vector is compared against a central database and the most similar results are produced as the output. Minimal source code modifications were made to control binding of specific threads to cores in order to reduce variability in testing and to ensure that the proper cluster of cores were being tested. The load/input, feature extraction, and output stages were bound to the same core due to their relatively light computation, whereas the segmentation, indexing, and ranking stages were each mapped to an individual core. The application was run with the *sim-large* input set.

The third application, was a *video playback*, implemented as a data parallel streaming application. The open source VLC video player is used after making minor modifications to the source code in order to observe the per-frame service time and to construct the empirical distributions. The workload was selected from MobileBench [73].[2] Hardware acceleration was disabled so that all video decoding was done by the application processor. The choice was made for two reasons: (1) reaching 30 fps with GPU assisted acceleration would be trivial, and (2) the platforms provide a greater level of control over the application processor, including per-core DVFS in the case of the Snapdragon chipset.

[2]Although all input files were evaluated, only results for `big_buck_bunny_720p_stereo.avi` is reported as these are the few formats that require significant amount of rendering time (i.e. obtaining 60 FPS was not trivially done at the lowest speed).

3.6.3 Validation of Execution Time and Power Models

This section demonstrates the appropriateness and value of NC model for modeling the execution time of a pipeline that involves interacting cores. Figure 3.7 plots the expected execution time for *image similarity search* that was obtained using the proposed statistical models in combination with NC, and the actual measured values on the Intel SNB processor. It also shows the results of the naive approach, which is simply the sum of the execution times of each stage. The experiment demonstrates the value and accuracy of the proposed approach. The naive approach is a significant overestimate of the actual execution time, which would result in a suboptimal solution of core frequencies when attempting to maximize performance or energy efficiency. It is due to the fact that a simple summation does not account for the overlapping of computations involving different data sets. On average, difference between the model predicted execution times and the observed values were less than 0.66%. The average error using the naive approach was 26%.

The parameters of the model of power described in Section 3.5, were estimated using data gathered from synthetic benchmarks aimed to stress the various portions of the microarchitectural and memory components. 2000 samples of power data were collected while running *image similarity search*. The average difference between the model predicted power and measured values was 1.72%.

3.6.4 Optimal Selection of Core Frequencies

The key component of the model creation is that only the independent tasks are characterized. This avoids a full experimental exploration of the state space such that each core only needs to be evaluated at each frequency (rather than evaluating the system at every possible frequency-core combination). This was used to construct the

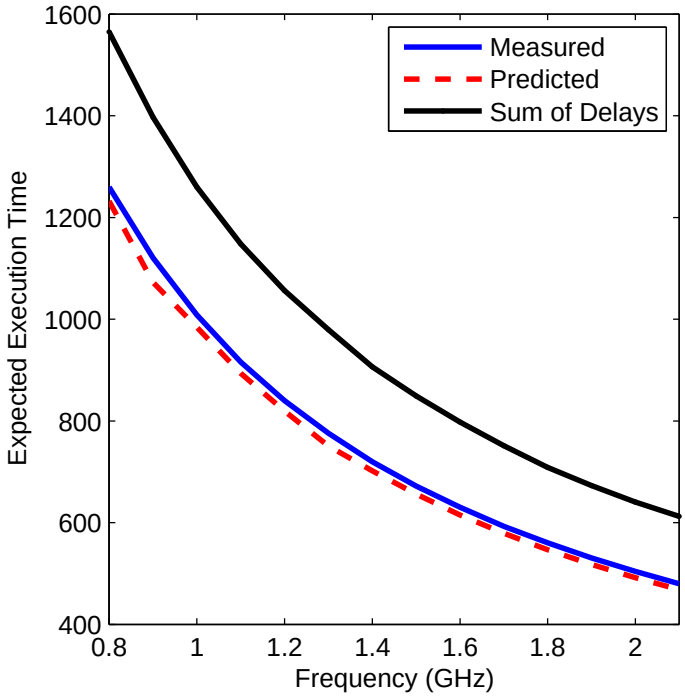


Figure 3.7: Predicted versus measured execution time of the image similarity search application on Intel SNB quad core processor.

individual densities $f_{\tau(s)}(x; K(s), \theta(s))$ (Equation 3.3) for each of the service threads.

With each $f_{\tau(s)}(x; K(s), \theta(s))$, the statistical model was used to compute the joint distribution of multiple random variables, where each corresponds to a core at an individual frequency. For the *web browser* and *video playback*, the joint distribution was computed using Equation 3.4. For *image similarity search* the joint distributions were computed using the NC approach described in Section 3.4.3.

To find the appropriate optimal frequency combination to solve Equations 3.1–3.2 an exhaustive enumeration of the operating frequencies was performed. The problem can then be resolved for various levels of QoS and stored in a table. This table is then used at run time to select the optimal frequency. While this a fairly heavy overhead (in our case 4 cores, 14 frequencies), it is a one-time static analysis which can bring significant improvements in energy efficiency (see Section 3.7).

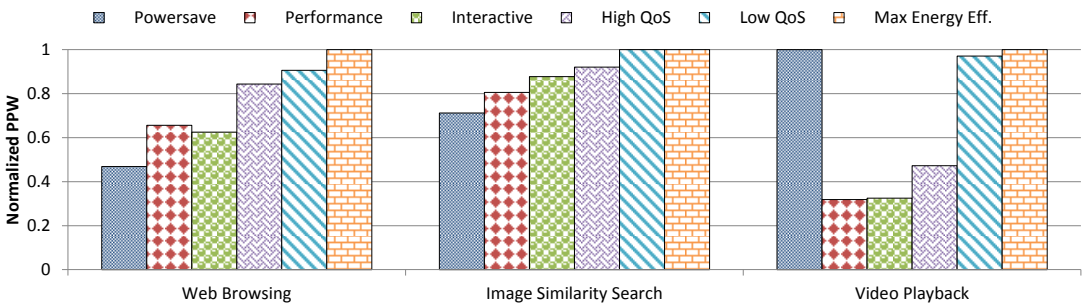


Figure 3.8: PPW scores of all frequency governors. Higher is better.

In terms of practicality, our current method is very similar to characterizing a standard cell for a VLSI library. For minor application updates or platform differences, the old models could be used. However, the exact loss of efficiency is difficult to quantify. Additionally, much like a standard cell needs to be re-characterized for different process technologies, our model needs to be constructed on a per-device basis or given significant application changes. While this is a limitation, the authors of this dissertation believe that the resulting increase in energy efficiency makes the process worthwhile. That said, one of our current efforts is to adapt the methodology to different platforms.

3.7 Case Study of Balancing Performance and Energy with Probabilistic Guarantee of QoS

In this section, the results of using the proposed methodology to maximize the PPW are compared against several standard schedulers available on Android OS: *powersaver*, *performance*, and *interactive* [74]. As stated earlier, the QoS $= (\Delta, Q)$, is a composite metric that includes both the deadline Δ and the likelihood Q (see Equation 3.2). For the following experiments, to ensure a high level of user satisfaction, Q is set to 0.9. That is, the selected frequencies should be such that the probability of meeting the deadline Δ is at least 0.9. For this set of experiments, since Q is fixed, QoS is effectively the same as the deadline Δ. For each application two levels of QoS

Table 3.3: Performance of several frequency governors on various mobile applications.

Governor	Web Browsing Req. QoS Lower is better	Freq. Vector	Obs. QoS Lower is better	Power	PPW
Powersave	—	300MHz, 300MHz, 300MHz, 300MHz	7.55 Sec	0.90 W	0.15
Performance	—	2.15GHz, 2.15GHz, 2.15GHz, 2.15GHz	0.97 Sec	4.99 W	0.21
Interactive	—	varies over time	1.52 Sec	3.56 W	0.20
High QoS	2.0 Sec	1.72GHz, 1.57GHz, 1.57GHz, off	1.36 Sec	2.76 W	0.27
Low QoS	4.0 Sec	1.26GHz, 960MHz, 960MHz, off	2.12 Sec	1.61 W	0.29
Max Energy Eff.	—	960MHz, off, off, off	4.24 Sec	0.75 W	0.32
Governor	Image Similarity Search Req. QoS Higher is better	Freq. Vector	Obs. QoS Higher is better	Power	PPW
Powersave	—	300MHz, 300MHz, 300MHz, 300MHz	0.84 Images/Second	0.85 W	0.99
Performance	—	2.15GHz, 2.15GHz, 2.15GHz, 2.15GHz	6.06 Images/Second	5.42 W	1.12
Interactive	—	varies over time	6.05 Images/Second	4.96 W	1.22
High QoS	4 Images/Second	652MHz, 422MHz, 422MHz, 1.49GHz	4.22 Images/Second	3.32 W	1.28
Low QoS	2 Images/Second	422MHz, 422MHz, 300Mhz, 960MHz	2.05 Images/Second	1.94 W	1.39
Max Energy Eff.	—	422MHz, 422MHz, 300MHz, 960MHz	2.05 Images/Second	1.94 W	1.39
Governor	Video Playback Req. QoS Higher is better	Freq. Vector	Obs. QoS Higher is better	Power	PPW
Powersave	—	300MHz, 300MHz, 300MHz, 300MHz	22.4 FPS	0.95 W	23.9
Performance	—	2.15GHz, 2.15GHz, 2.15GHz, 2.15GHz	39.2 FPS	5.15 W	7.61
Interactive	—	varies over time	39.1 FPS	5.03 W	7.77
High QoS	30 FPS	1.72GHz, 1.72GHz, 1.72GHz, 1.72GHz	31.7 FPS	2.81 W	11.3
Low QoS	20 FPS	652MHz, 652MHz, 652MHz, 652MHz	23.3 FPS	1.00 W	23.2
Max Energy Eff.	—	300MHz, 300MHz, 300MHz, 300MHz	22.4 FPS	0.95 W	23.9

were selected to represent a high and low state. Additionally, an unconstrained solution is explored as well—Maximum Energy Efficiency (Max Energy Eff.). Together these three points can represent different tradeoffs between energy efficiency and user satisfaction, as shown in Figure 3.8 and Table 3.3.

For *web browsing* and *image similarity search*, the selected levels of QoS (high, low, and unconstrainted) resulted in greater energy efficiency than the linux governors.

The frequency governor, *interactive*, tends to over-react to processor utilization and attempts to set the processor to the highest available frequency (i.e. assumes it is most energy efficient to finish the job as quickly as possible and then slow/disable the processor). As shown in Table 3.3, the proposed optimization method determines independent frequency levels for each application processor core. This is due to imbalances in the workloads between the cores. For example in the case of *web browsing* the first core receives a significant portion of the total number of URIs which constitute that core's UoC. Therefore its frequency is set to be the highest of 1.72GHz. In contrast, the other cores receive very few URIs which are typically not too computationally demanding. Thus the frequency of these latter cores are reduced. Because there was never a case when four UoCs were simultaneously being executed, the last application processor core's frequency was set to zero, i.e., automatically disabled, to minimize power consumption. This agrees with the observation made in a recent mobile device utilization study [75]—the processor utilization of web browsing has peaks and valleys between one to three cores of a four-core application processor with an average utilization to be below two cores. Similarly, for the *image similarity search* application, one see that the optimization determines frequencies based on the imbalances in the application pipeline. Specifically, it can be observed that the most computationally demanding stage is mapped to the fourth core and thus always requires a substantially larger frequency than the first three cores.

An interesting situation occurred in the case of video playback. As seen in Figure 3.8 and Table 3.3, the lowest frequency setting (i.e. Powersave Governor) results in the optimal PPW point for the system. The unconstrained optimal point of operation occurs at the same frequency setting. For this reason it is very tempting to choose a very slow speed to yield the highest energy efficiency; however, the data shows that the QoS would dramatically drop (given a Q value of 0.9). Thus the video

would be playing at 10 fps or lower with a likelihood of at least 90%. It is quite reasonable to assume this to be intolerable for a large user base. This issue highlights the importance of adding constraint (3.2) to the PPW optimization problem.

From these results, it is possible to see that there exists a strong non-linear relation between PPW and QoS (Δ). This is most prominent in *video playback*. Table 3.3 shows that increasing the required fps from unconstrained to 20 fps only degrades PPW by 2.9%, whereas increasing it from 20 to 30 fps degrades PPW by 51.3%. This must be considered when balancing user experience and energy savings.

To reiterate, the key points found in our analysis are:

- The unconstrained (no consideration to user satisfaction) optimization problem results in the highest possible energy efficiency.
- Including a QoS constraint reduces the possible optimal energy efficiency. Alternatively, reducing Δ (i.e. requiring the application to finish earlier) or increasing Q (i.e. requiring a larger spread of execution times to achieve the chosen deadline) requires a greater amount of power thus reducing energy efficiency.

Next, the accuracy or tightness of the likelihood constraint given in Equation 3.2 is examined. To do so, the observed or actual value of Q is plotted versus the required or specified value of Q. This is shown in Figure 3.9, using *web browsing* as an example. The Firefox web browser was repeatedly executed 100 times with Q set to 0.1, 0.5, and 0.9, and $\Delta = 2.0s$. Then the number of page loads that exceeded Δ (2 second load time) was recorded. Since the specified value of Q is a lower bound on the likelihood, and the observed Q is simply a sample estimate of the likelihood, all the points in the plot should be at or above the $y = x$ line. The observed results are very consistent with the expected, and show that the constraint on the likelihood does indeed restrict the set of optimal solutions.

To highlight the strengths of the proposed approach, the tradeoff analysis is pre-

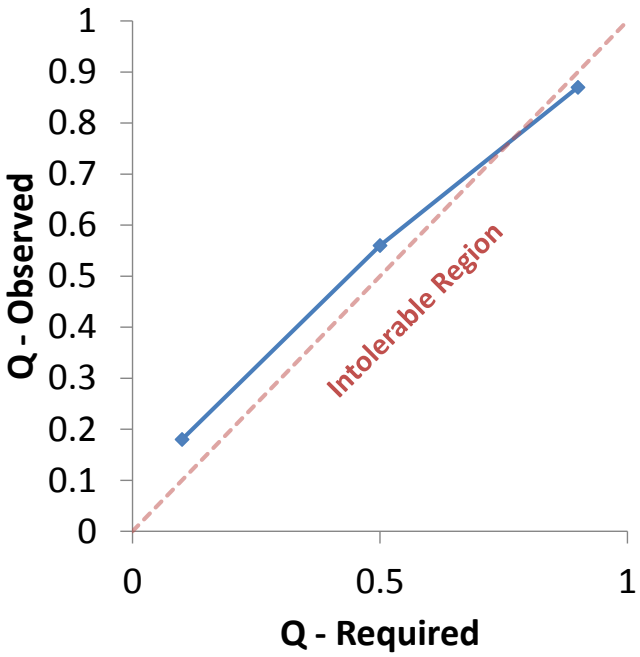


Figure 3.9: Observed Q versus specified Q (lower bound on likelihood), showing tightness of constraint in Equation 3.2.

sented. Specifically, *web browsing* is explored in greater detail to better demonstrate the benefits of modeling execution time as a non-deterministic value. Figure 3.10(b) presents the energy efficiency of *web browser* at the corresponding frequency settings. The green circles in Figure 3.10(a) represent the average execution time needed to load the web page. The range for each point represents the statistical distribution of the execution time for *web browsing* in the various frequency settings.

In addition to the expected load times of the *web browser*, the proposed framework also provides a *quantitative measure* for the likelihood of performance guarantee. This is shown in Figure 3.10(c) where a deadline on the web page load times is set to 2 seconds. The likelihood of reaching this deadline at a frequency setting of 1.728GHz is represented by the grey shaded area under the distribution, and the likelihood of meeting that deadline is 95.48%. The data also presents tradeoffs between likelihood and PPW. While a frequency of 1.728GHz provides a high likelihood of satisfying the

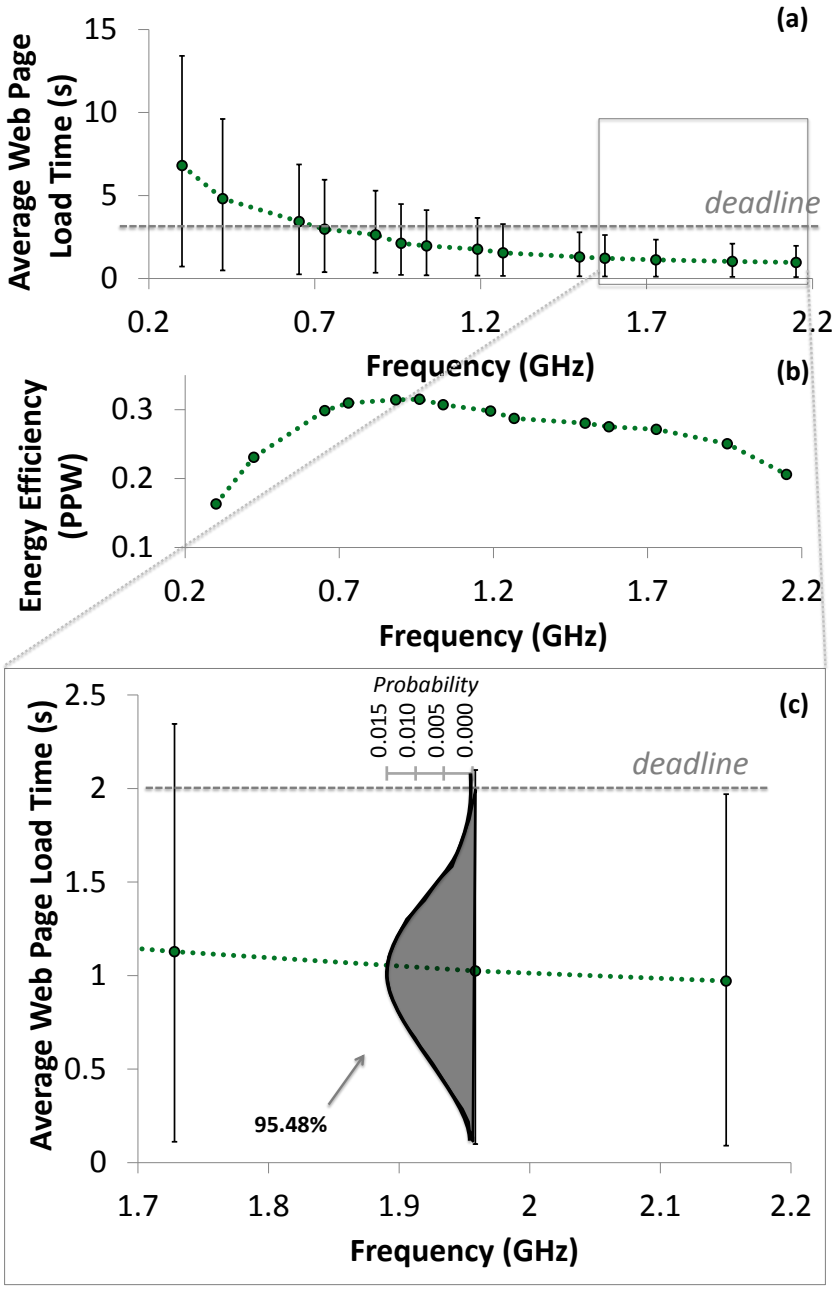


Figure 3.10: (a) Execution time, (b) energy efficiency results at all available frequencies for *web browsing* on the Dragonboard. (c) Example calculation of the probability of meeting the specified deadline.

deadline, it is suboptimal in terms of PPW (see figure 3.10(b)). If the a frequency of 960 MHz was selected, the likelihood is much less (55%) but the PPW is close to the optimal (0.315). The reduced likelihood implies the possibility of the image quality being degraded.

Figure 3.11 shows this result. It is clear that as the frequency increases, the variance in web page load times decreases, and the difference in mean execution time reduces. There are diminishing returns on average web page load time; however, the likelihood of reaching any given deadline increases as well. When coupled with the non-linear power model, the data shows that the optimal point may not necessarily be one which gives a necessary level of QoS.

3.8 Chapter Summary

This dissertation provides a new and necessary step toward developing a methodology for optimizing energy efficiency subject to user satisfaction. A statistical framework is developed for characterizing, profiling, and predicating the execution time of parallel applications running on mobile platforms.

The proposed statistical framework provides additional and valuable information that can be used to guide the control of voltage/frequency settings available on modern processors. This is particularly helpful for today's energy-constrained smartphones that execute real-time apps with execution time constraints. With the detailed performance and energy efficiency characterization on web browsing, it was shown how the proposed statistical framework is used to consider the energy efficiency for smartphones while accounting for a probabilistic guarantee on QoS.

Specifically, the optimization method creates a set of static tables containing frequencies corresponding to optimal energy efficiency, subject to specified QoS levels. Once the application is started, the processor state is altered based on the desired

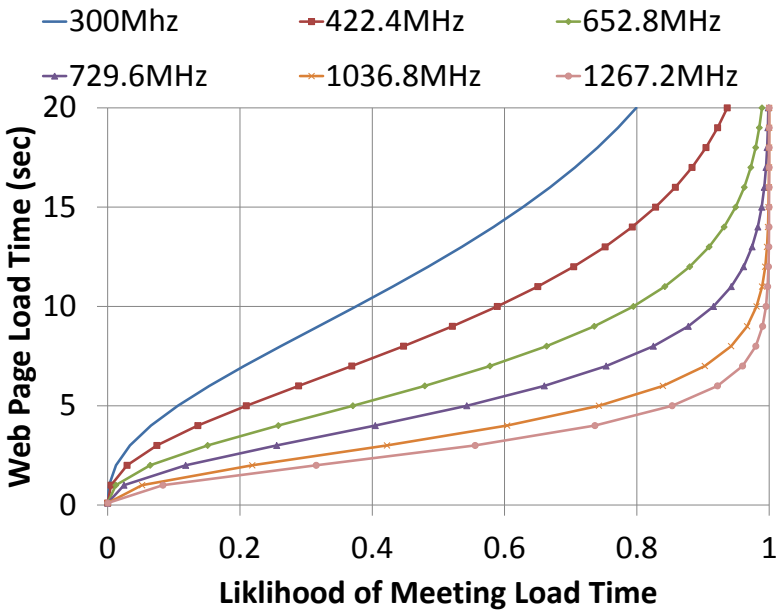


Figure 3.11: Web page load time versus the likelihood of reaching those load times for various frequency settings.

QoS level. This is an important first step towards ensuring user experience in non-deterministic workloads. However, this is limited to offline analysis of stochastic workloads while the most desirable use case would be to dynamically tune the processor based on the changing characteristics of the user, input, background task noise, and other system state. In the following chapter, an efficient and accurate modeling technique is presented to accomplish exactly that use case.

Chapter 4

A DATA-DRIVEN, MOMENT BASED FRAMEWORK FOR PREDICATING PERFORMANCE AND POWER IN REAL TIME

The success of mobile devices and applications is directly linked to a user's satisfaction of the quality of service – a metric used to denote the user's perception of the quality of an application. The first and necessary building block to manage user satisfaction is to establish accurate performance and power models which are sensitive to the mobile device's controllable features such as scalable voltage and frequency. Traditionally, performance and power models have been developed with server and desktop workloads in mind; assuming long term, stable operating conditions. However, this is insufficient for mobile workloads, which are subject to many sources of variability (e.g. user interactions, network delay, architectural stalls, etc.) leading to unpredictable phases of computation. As such, modern energy management algorithms tend to use overly simplistic modeling techniques which provide little to no insight into the level of confidence of the model estimations, potentially resulting in large errors, wasted energy, missed user satisfaction targets, and sub-optimal system control. This work establishes the importance and value of modeling the many sources of variations in mobile workloads. A completely data-driven approach is presented which is capable of accurately predicting the workload's statistical characteristics which may follow any arbitrary distribution. The method is light-weight allowing for real-time model evaluation and update.

4.1 Problem Background

Mobile computing devices have become ubiquitous in everyday life. These devices serve as the platforms in which services and applications may interact directly with users. Paramount to the success of any given application is the application's *quality of service* (QoS) – a quantifiable metric used to capture a user's satisfaction with the mobile device and application. For example, QoS could represent the load time of a web page, the frame rate of video playback, or the response time of an application after receiving a command from the user. Logically, one would assume, to produce the highest level of user satisfaction, it is best to operate the device at its maximum frequency. However, this comes at the cost of high power consumption: something not sustainable for mobile devices with limited energy sources such as rechargeable Li-Ion batteries. A recent survey of smartphone consumers demonstrates the importance of long battery life, ranking it as the most important factor in deciding which phone to purchase [76]. Thus, in order to successfully control a mobile device, one must find a balance between maximizing QoS and minimizing energy consumption.

To maintain high user satisfaction, mobile devices employ various energy management techniques. One such method is online management of controllable parameters such as voltage and frequency, in order to trade-off between power consumption and system performance. Critical to the success of these management techniques is to properly predict future power and performance values. However, quantifying QoS is a non-trivial task as QoS is application specific and thus cannot be generalized to all workloads. As a proxy to QoS, many prior works utilize processor performance or instructions per second (IPS) since it provides a loose measure of the rate of progress towards completing a task – the completion time reflecting the user's satisfaction [3, 4, 5]. IPS has the benefit of being application independent, requiring no

information from the application itself. However, with additional context provided by the application, QoS can be defined as a more relevant parameter such as the response time of the workload or the rate at which requests are processed [77, 78]. As a matter of convenience, this paper will use the term *performance* to represent both application performance metrics (i.e. QoS) and processor performance metrics (i.e. IPS) since both values can be related to user satisfaction.

In regards to performance and power prediction, a major limitation of prior work is the assumption of steady state workload characteristics - leading to deterministic values of performance and power. While this might be applicable to many server and desktop applications, mobile workloads are subject to numerous sources of uncertainty. One such source of uncertainty is due to the interactive nature of mobile workloads. User actions induce interruptions into the system's steady state behavior at random times by making various workload requests. Other causes of system variations include interference by co-scheduled and background applications [65] or processor stalls [41], etc.

Ultimately, these sources of uncertainty cause performance and power to be stochastic quantities. Traditionally, prior works have treated power and performance as deterministic quantities, providing no insight into the statistical distribution of possible values [2, 3, 4, 5, 77, 78]. This is not sufficient for the mobile domain due to the high level of system uncertainty present. Estimation of this likelihood is extremely valuable as it provides a measure of uncertainty when making performance prediction. For example, prior studies have demonstrated that for highly interactive workloads such as interactive games, latencies of about 100 milliseconds are required to maintain satisfactory user experience. Additionally, latencies of less than 50 milliseconds are undetectable by users, thus providing no benefit to the overall user experience [79, 80, 81]. With knowledge of the distribution of possible performance

and power values, energy management techniques may tune the system to maximize the likelihood of achieving a response time between 50 and 100 milliseconds while minimizing energy consumption

In this chapter, a prediction framework is proposed to determine non-deterministic workload power and performance characteristics. The proposed framework is built upon arbitrary polynomial chaos – representing performance and power as functions of statistically varying system parameters such as architectural delays, network conditions, application input variations, and so-on. Unlike regression based modeling techniques which require a known distribution of input and output parameters, the proposed framework is entirely data-driven, thus capable of optimally capturing the effects of any arbitrary distribution. This is a necessary feature to accurately capture the statistical characteristics of mobile workloads as shown in Section 4.2. The models are both computationally efficient and accurate, reducing performance prediction error by **2.91X** and **1.70X** respectively over prediction methods used by the Android frequency governors at an increase in overhead of 0.0016%.

4.2 Problem Definition and Related Works

The problem of predictive modeling is one of determining a function $y = f(X, s)$ where y is the response variable, X is a set of observable system variables, and s is a set of controllable variables.

To make predictive modeling tractable, some restrictions must be placed on $f(X, s)$. One approach is to construct a functional form (usually a polynomial) for $f(X, s)$, where X is heuristically selected based on detailed knowledge of processor architectures [2, 3, 4, 5, 82]. For instance, in [82], an analytical performance model is built using insights into the ARM Cortex-A8 processor architecture. The model calculates the total number of execution cycles needed to evaluate any given application based on

a linear combination of (1) the number of instructions comprising an application and (2) the penalty to execution time due to pipeline stalls caused by various architectural events such as miss events, inter-instruction dependencies, and functional unit limitations. These penalty functions take as input the counts of each architectural event and are parameterized by the architectural characteristics of the processor (pipeline width and depth, instruction latency, etc.). By breaking the performance into these components, the contribution of performance from each architectural event can be determined and performance bottlenecks can be identified. However, this method requires extensive knowledge of both the platform's architecture, as well as workload characteristics such as inter-instruction dependencies. Acquiring such information can be costly, requiring detailed analysis of the application and system. Furthermore, any modifications to the assumed system state (e.g. adding background tasks which cause resource contention) would require complete re-analysis of the system.

An alternative is to take a parametric approach [83, 84, 85, 86, 77, 65, 87] which restricts $f(X, s)$ to a known class of functions $f(X, s, p)$, whre p is a set of unknown parameters. For instance, f could be a multivariate polynomial in X and s, and p would be the coefficients. The parameters p are estimated by collecting sample data (y, X) and solving $\text{argmin}_p E(y, f(X, s, p))$, where E denotes some error functions (e.g. quadratic) and argmin returns the parameters, p, which correspond to the minimum point of E. Among these prior works, [77, 65, 87] are most closely related to our approach. In [77], Zhu and Reddi proposed the use of input characteristics along with performance counters to estimate the application specific performance metrics of the web browser. The study assumed a linear model for the web browser's performance, but the accuracy was low (about 66%). Shingari et al. [65] extended these models to account for variations due to co-scheduled tasks and shared memory contention. The reduced order model was constructed via offline experimentation of

the web browser loading the 50 most frequently visited web pages along with numerous combinations of interfering applications. One method to increase accuracy of web page load time is to annotate the HTML documents with additional information reflecting the page's complexity [87]. However, this requires significant modification to the HTML code of every webpage, making this impractical for energy management.

A limitation of the previously discussed prior works is the assumption of deterministic systems. This is rarely the case in computing systems, especially mobile systems in which both X and y exhibit statistical variations. Reference [41] was one of the first to demonstrate the stochastic nature of execution times in mobile applications, and identified the numerous sources of variations, including branch mispredictions, cache misses, various types of interrupts, and co-scheduled background applications competing for resources. Based on heuristic arguments, the execution time was modeled as a gamma distribution, whose parameters had to be estimated from sample data. As is often the case in modeling, distributional assumptions are made for convenience, and cannot be physically justified. For these reasons there has been a shift towards data driven methods of uncertainty quantification [88].

4.3 Demonstrating the Non-deterministic Nature of Mobile Workloads

To substantiate the need for a more robust selection of execution time distribution, we executed 10 mobile applications[1] 100 times each and recorded the associated performance power values. To eliminate the variation caused by the control parameters, frequency selection and input complexity, each trial was conducted at a fixed frequency under the same input set. Figures 4.1 and 4.2 depict the distributions of performance and power respectively.

It is evident that no single, common model of a parametric distribution would

[1]See Section 5.4.2 for full details of the experimental platform and workloads.

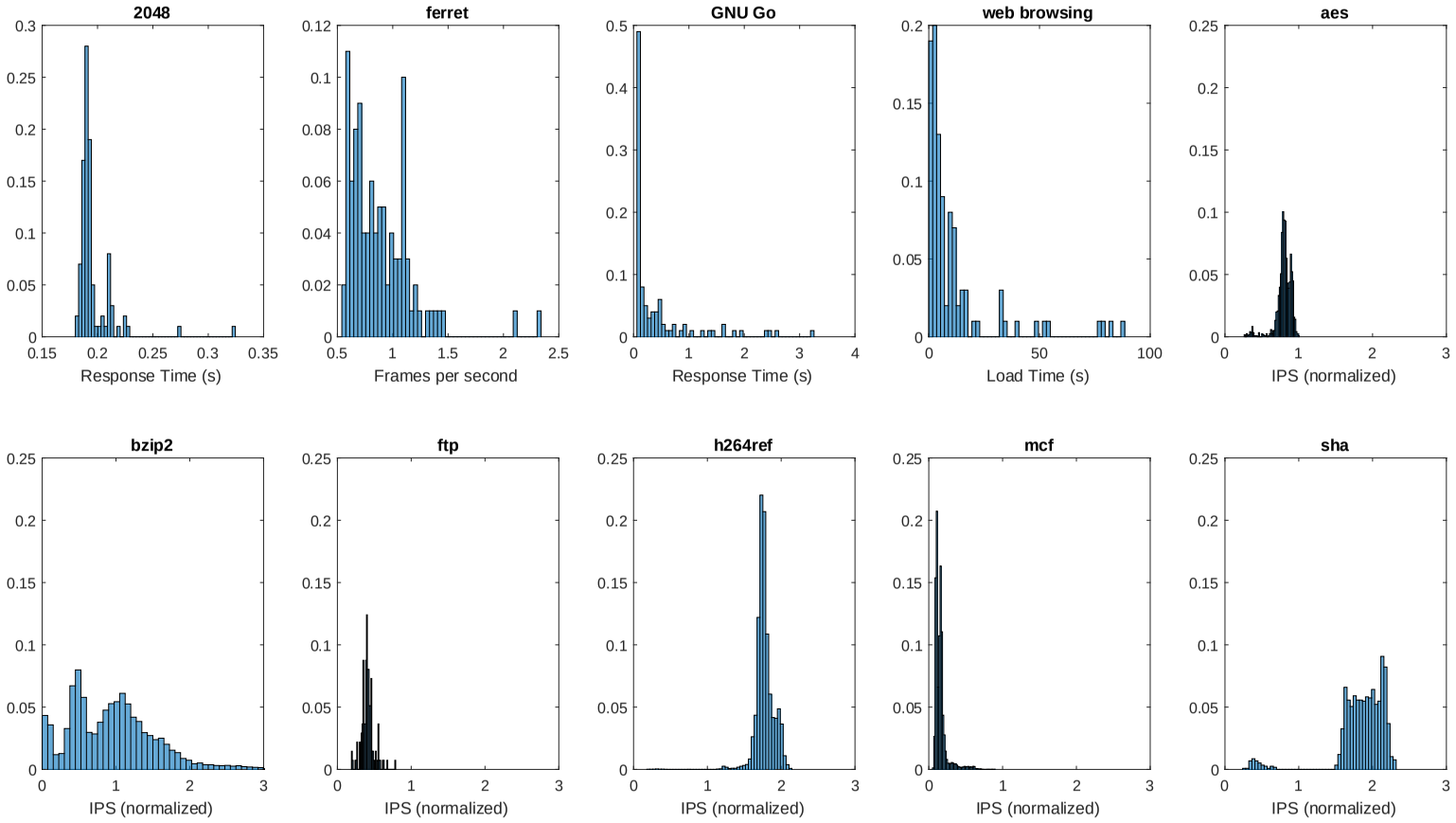


Figure 4.1: The distributions of performance for the 10 mobile workloads.

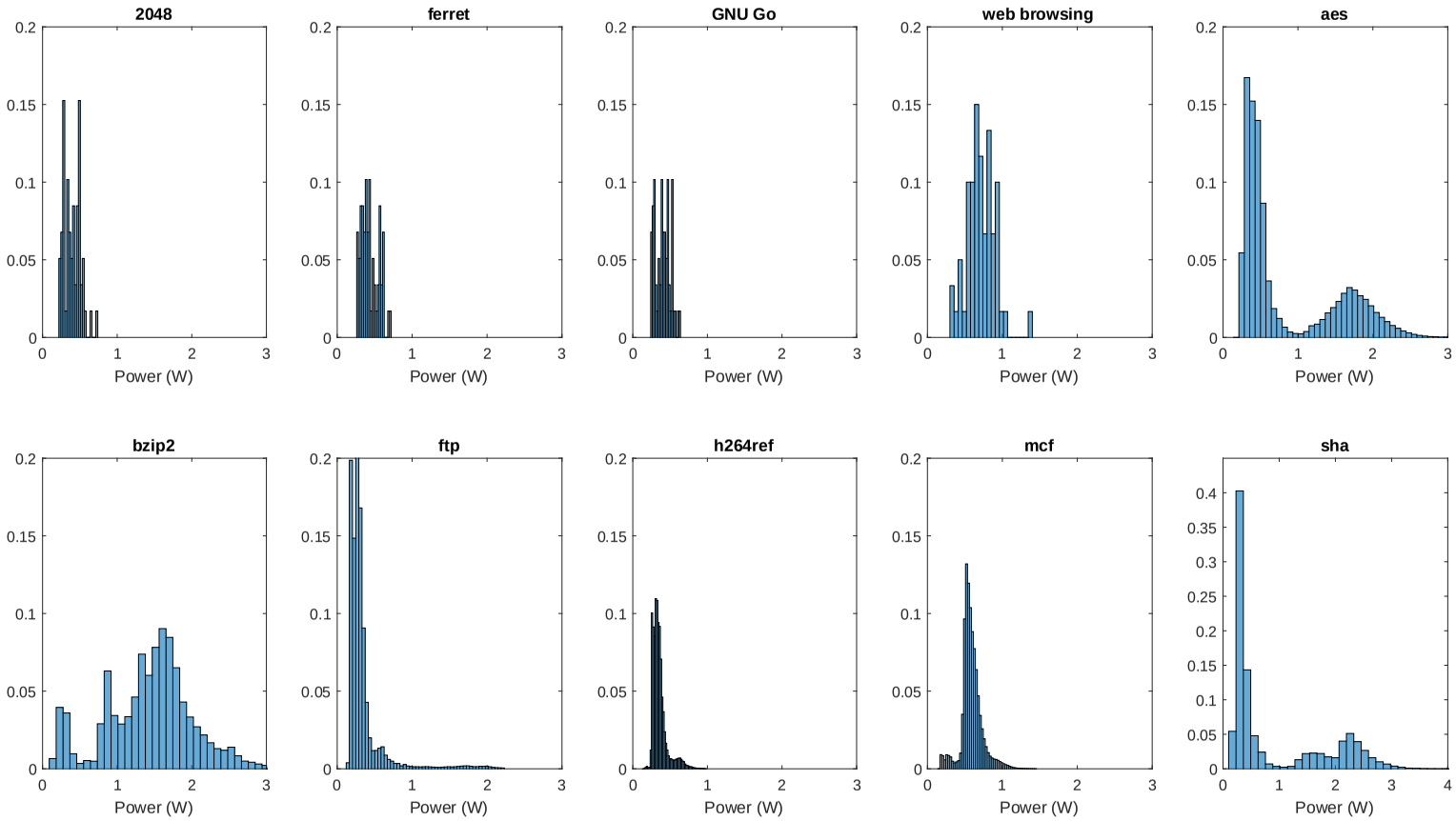


Figure 4.2: The distributions of power consumption for the 10 mobile workloads.

explain all the data sets. Examining the power and performance distributions of *aes*, *bzip2*, and *sha* in particular show multi-modal distributions. Additionally, certain workloads such as *2048*, *ferret*, *GNU Go*, and *mcf* appear to have quite varied distributions. As such we conclude that a more robust modeling mechanism is needed to accurately capture performance and power distributions.

Modeling uncertainties is necessary for effective control of user-centric workloads. To see how this might be useful: consider the gaming AI application, GNU Go. After a user input, this application must evaluate a set of possible "moves" and determine the best possible move to make. This process is repeated until the game's conclusion. As stated earlier, it is important to maintain a response time between 50 and 100 milliseconds to ensure user satisfaction. Therefore, we need to understand the performance and power characteristics of this application not only to ensure satisfactory user experience, but also to maximize the performance-per-watt (i.e. the energy efficiency) of the system – extending the system's battery life.

We begin by analyzing the system at a fixed operating frequency of 1.4GHz and record the number of cycles needed to process the response along with the associated power values. In the interest of space we do not display the results for each frequency; however, this analysis would be necessary to understand the relationship between the control variable, frequency, and the resulting power and performance metrics. Figure 4.3 shows the frequency histograms of the response time, the power consumption, and the energy efficiency of this application at a fixed processor frequency. Given such distributions for each operating frequency, it is possible to evaluate (1) the frequency corresponding to the highest energy efficiency on average and (2) the likelihood of each frequency producing a response time between 50 and 100 milliseconds. In the case of *GNU Go*, there is approximately a 27% likelihood that the input will be serviced within 100 milliseconds. Therefore, the mobile device should increase the operating

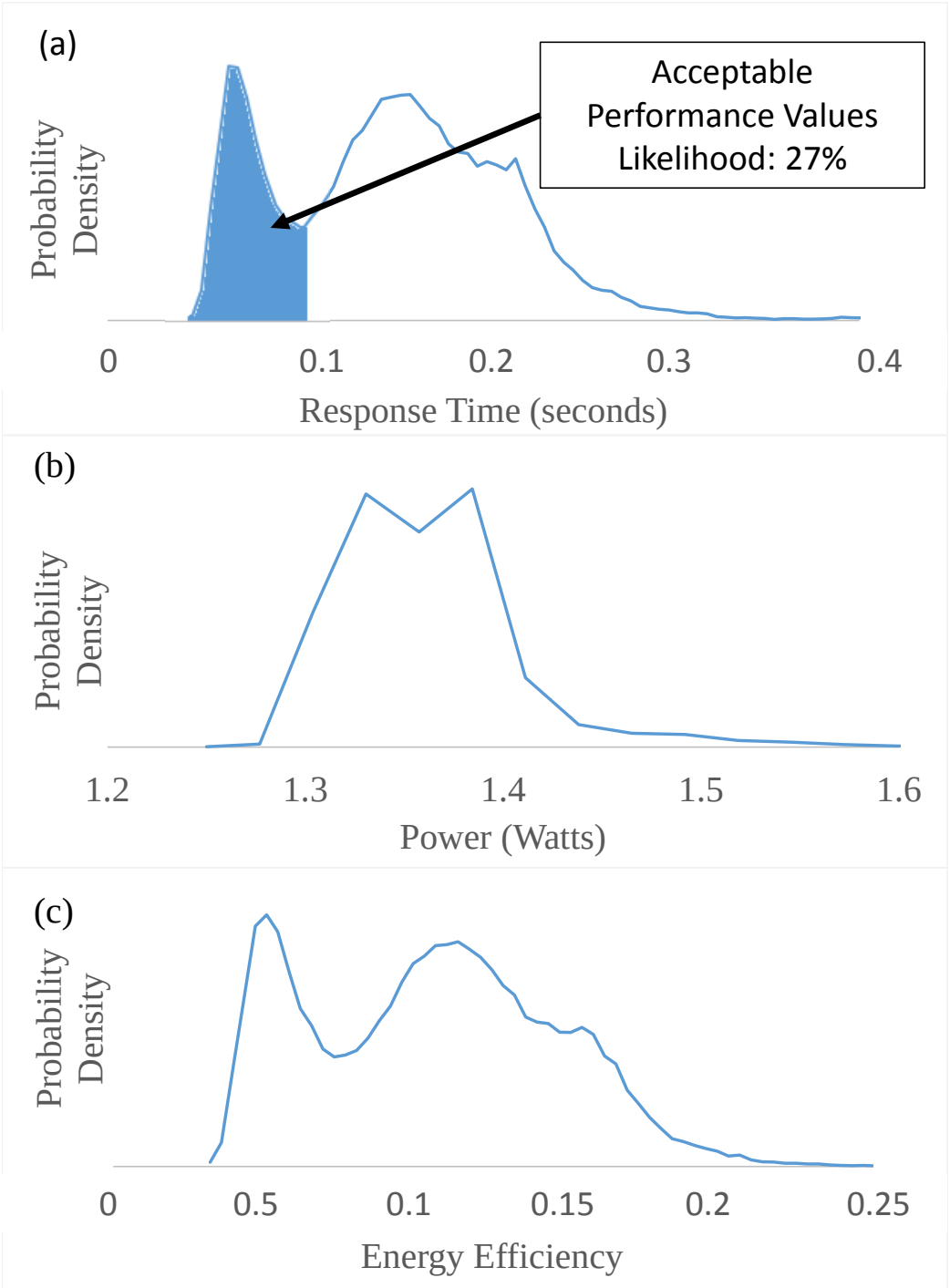


Figure 4.3: (a) The measured distribution of performance values for the GNU Go workload – a gaming AI which responds to user inputs – at a fixed CPU frequency of 1.4GHz. (b) Measured distribution of power values seen when running GNU Go. (c) The resulting distribution for energy efficiency.

Table 4.1: Summary of Advantages and Disadvantages of Various Prediction Methods

Prediction Method	Accuracy	Convergence	Complexity	Random
Last Observed [78]	Low	Low	Low	—
Static Value [41]	Low	Low	Low	—
Regression [77, 83, 84, 85, 86]	High	Moderate	Moderate	Gaussian
Proposed	High	High	Moderate	Arbitrary
Offline Simulation [89, 90]	Very High	—	Very High	Arbitrary

frequency to improve the likelihood of meeting quality of service expectations.

Based on the real system's performance and power characterization results, we propose a framework to construct a stochastic model to represent performance and power of a mobile device. The framework is able to learn its parameters directly via data collection, making no assumptions of the underlying distributions. This technique has been shown to provide faster convergence over traditional Gaussian approaches [88], thus requiring fewer data samples and lower model order to accurately approximate the system response. To summarize, this paper proposes an efficient and accurate means of estimating power and performance subject to numerous sources of uncertainty. The approach treats the values as stochastic responses such that randomness is caused by variations in the underlying system state observables (e.g. cache hit ratio, context switches, input characteristics). The benefits of this data-driven, learning-based algorithm are summarized in Table 4.1. With marginal increase in complexity, our prediction framework achieves higher accuracy and convergence rate than traditional regression modeling techniques by considering the stochastic nature of operating a mobile system.

Table 4.2: Input Variables for the IPS Prediction

System Events	
x_1	L1 data cache misses per cycle
x_2	Branch misses per cycle
x_3	Shared L2 cache misses per cycle
x_4	Bus accesses per cycle
x_5	Page faults per cycle
x_6	CPU Temperature
Controls	
x_6	Power Oriented CPU cluster frequency
x_7	Performance Oriented CPU cluster frequency
x_8	GPU frequency
x_9	Memory bus frequency

4.4 A QoS Prediction Framework using Polynomial Chaos Expansion

4.4.1 Overview

In order to maintain coherency, this section provides a brief explination of polynomial chaos expansion (PCE); however, readers should view Appendix A for a more complete tutorial of PCE. PCE provides a mechanism to relate a stochastic characteristic of the application, $y(t)$, to a collection of randomly varying system observables, $X(t)$. To model QoS, this work elects to use an X defined by Table 4.2 as these have shown to have strong correlation to IPS and power in prior works [41, 2]; however, the proposed methodology is general enough to accommodate any composition of X and merely use Table 4.2 as an example.

As mentioned earlier, mobile applications and their QoS are subject to numerous sources of non-determinisms ranging from architectural variations (e.g. memory access rates and delays), system level variation (e.g. co-scheduled applications utilizing the same shared resources), and application level variation (e.g. input complexity such as the composition of a webpage for the web browser or the number of objects in an image for image recognition algorithms). As such, to accurately and efficiently model the relationship between the response and the sources of variation, statistical information of these values must be known. For example, classical model regression methods such as least squares error minimization rely on deterministic data samples or data samples with Gaussian noise [91]. These methods lose their optimality and convergence rates when data samples follow different random distributions. Instead, this work proposes to utilize *polynomial chaos expansion* (PCE) to represent QoS. PCE presents an effective means to correct these deficiencies and improve convergence rates by decomposing $X(t)$ to a set of orthogonal polynomials with little to no effect on the computation time needed to evaluate the model.

4.4.2 A Brief Background to Polynomial Chaos Expansion

Consider a system with inputs $X(t) = \{x_1(t), x_2(t), \cdots, x_M(t)\}$, and output $y = f(X(t))$. Often, as in this present situation, f is unknown and must be estimated from data. One common approach is to assume a specific form for f. For example, in the case of deterministic values of X, linear regression may be used which assumes $\hat{y} = f(X(t)) = \sum c_i x_i(t))$ such that the unknown coefficients, c_i, must be estimated using samples of input-output pairs obtained by some means, either by measurements on a real system, or by simulation of a model. The optimal coefficients are obtained by minimizing an error norm $|||\hat{y} - y||$.

How can one proceed in the case that $X(t)$ is a stochastic process? The answer is

very similar to the deterministic case outlined above, and is referred to as *polynomial chaos expansion* (PCE) [92]. The PCE expansion, originally referred to as Homogenous Chaos [93], expresses a Gaussian process $y(t)$ in terms of an infinite collection of independently varying random variables:

$$y(t) = \sum_{i=0}^{\infty} c_i(t)\phi_i(x_1, x_2, \cdots), \tag{4.1}$$

where $(x_1, x_2, \cdots)$ denotes an *infinite* collection of independent, standard Gaussian random variables, and the *basis functions* $\{\phi_i(\cdot), i = 0, 1, \cdots \infty\}$ are orthogonal Hermite polynomials for all (i.e. infinite) orders. The equality expressed in equation (4.1) is in the limit, and in the mean-square. Intuitively what all this says is that for each t, the right hand side of (4.1) can be used (in theory) to generate the random variables $y(t)$ for each t. In practice, the maximum order of the polynomials will be fixed, as well the number of random variables X.

PCE is not restricted to Gaussian processes and has been generalized for other standard distributions [92]. Table 4.3 outlines the optimal orthonormal basis functions for some common distributions. This method of creating PCE models under the assumption of known distributions for $X(t)$ is commonly referred to as **generalized polynomial chaos expansion** or gPCE for short.

4.4.3 Polynomial Chaos Expansion for Arbitrary Distributions

A limitation of gPCE is the reliance on knowledge of the underlying distributions of the input variables. To address this issue, Oladyshkin and Nowak ([88]) proposed a data-driven alternative to gPCE called **arbitrary polynomial chaos expansion** (aPCE). Much like gPCE, the result of aPCE is a truncated series of orthogonal basis functions,

$$\hat{y}(t) = c_0(t) + \sum_{j=1}^{R} c_j(t)\phi_j(X(t)), \tag{4.2}$$

Table 4.3: Optimal Polynomials for Various Probability Spaces.

	Variable Distribution	**Polynomial Class**
Continuous	Gaussian	Hermite
	Log-normal	Hermite
	Gamma	Laguerre
	Beta	Jacobi
	Uniform	Legendre
Discrete	Poisson	Charlier
	Binomial	Krawtchouk
	Negative Binomial	Meixner
	Hypergeometric	Hahn

representing the estimated response surface, $\hat{y}(t)$, as a weighted summation of R multi-variate orthogonal polynomials. The derivation of the aPCE basis functions is briefly discussed here; however, readers should consult ([88]) for the full derivation.

To determine the form of the multi-variate basis functions one must utilizing the work of Ghanem and Spannos ([94]) which states: given the M-dimensional input vector $X(t)$ such that all parameters within $X(t)$ are independent, the multi-dimensional basis functions, ϕ_i, $\forall i = 1, \ldots, R$, can be constructed as the product of the corresponding univariate polynomials, $P_j^{(k)}(x_j)$. That is,

$$\phi_i(X) = \prod_{j=1}^{M} P_j^{(\alpha_j^i)}(x_j), \quad \sum_{j=1}^{M} \alpha_j^i \leq K, \quad i = 1 \ldots R, \tag{4.3}$$

where K is the order of the aPCE function, $R = (K + M)!/(K!M!)$ is the total number of multivariate polynomials of degree less than or equal to K and α_j^i is a multivariate index which is capable of enumerating all possible products of individual univariate basis functions.

Therefore, the problem reduces to determining the M sets of univariate polynomial basis functions

$$P_j^{(k)}(x_j) = \sum_{i=0}^{k} p_{i,j}^{(k)} x_j, \quad k = 0 \dots K, \quad j = 0 \dots M. \tag{4.4}$$

where all polynomial coefficients, $p_{i,j}^{(k)}$, must be derived. To do so, a collection of k equations is constructed using the definition of orthogonality:

$$\int_{x_j} P_j^{(k)}(x_j), P_j^{(l)}(x_j) d\Gamma(x_j(t)) = 0, \quad \forall k \neq l \tag{4.5}$$

where $\Gamma(x_j(t))$ is the probability density of $x_j(t)$. Given that the i^{th} moment of $x_j(t)$ is $\mu_i = \int_{x_j(t)} x_j(t)^k d\Gamma(x_j(t))$, these equations can be simplified to the following closed system of equations:

$$\begin{aligned} \sum_{i=0}^{k} p_i^{(k)} \mu_i &= 0 \\ \sum_{i=0}^{k} p_i^{(k)} \mu_{i+1} &= 0 \\ &\vdots \\ \sum_{i=0}^{k} p_i^{(k)} \mu_{i+k-1} &= 0 \\ p_k^{(k)} &= 1. \end{aligned} \tag{4.6}$$

Thus, it is possible to determine the optimal orthogonal univariate polynomials as a function of the 0^{th} to $2k^{\text{th}}$ moments of $x_j(t)$. As an example, the univariate polynomials of order 0, 1, 2, 3 are given below

order 0: $P^{(0)}(x_j(t)) = 1$

order 1: $P^{(1)}(x_j(t)) = x_j(t) - \mu_1$

order 2: $P^{(2)}(x_j(t)) = x_j(t)^2 + \dfrac{\mu_3 - \mu_1\mu_2}{\mu_1^2 - \mu_2} x_j(t) + \dfrac{\mu_2^2 - \mu_1\mu_3}{\mu_1^2 - \mu_2}$ order 3: $P^{(3)}(\xi) = \xi^3$

$$+\frac{-\mu_5\mu_1^2+\mu_4\mu_1\mu_2+\mu_1\mu_3^2-\mu_2^2\mu_3+\mu_5\mu_2-\mu_4\mu_3}{\mu_4\mu_1^2-2\mu_1\mu_2\mu_3+\mu_2^3-\mu_4\mu_2+\mu_3^2}x^2$$
$$+\frac{-\mu_2^2\mu_4+\mu_2\mu_3^2-\mu_1\mu_5\mu_2-\mu_1\mu_3\mu_4-\mu_5\mu_3+\mu_4^2}{\mu_4\mu_1^2-2\mu_1\mu_2\mu_3+\mu_2^3-\mu_4\mu_2+\mu_3^2}x$$
$$+\frac{-\mu_5\mu_2^2+2\mu_2\mu_3\mu_4-\mu_3^3+\mu_1\mu_5\mu_3-\mu_1\mu_4^2}{\mu_4\mu_1^2-2\mu_1\mu_2\mu_3+\mu_2^3-\mu_4\mu_2+\mu_3^2}.$$

A critical result of [88] is the convergence rate of aPCE in relation to regression techniques and gPCE modeling. The results indicate that exponential reduction of model error is achieved utilizing aPCE with increasing model order and data sample size when compared to gPCE methods.

4.4.4 Evaluating Expected Value and Probability Constraints

Thus far, a mathematical function has been derived which relates a collection of random variables, $X(t)$, to some stochastic response variable, $y(t)$, via aPCE. However, it is desirable to determine the probability distribution function of $y(t)$ in order to evaluate the expected value, or the likelihood of achieving some range of values of $y(t)$. For example, if $y(t)$ represents the response time of a user-centric application it is desirable to know what the likelihood of $y(t)$ being less than 1/30 seconds[2] is under any given control policy, $\boldsymbol{s}(t)$.

To do so, two requirements are needed: first the aPCE representation of $y(t)$ must be derived and second, the (empirical) distribution of each $X(t)$ must be known. In order to determine the polynomial basis functions of the aPCE model, the statistical moments of $X(t)$ must be determined. One method of doing so is via data collection and estimating the moments via empirical methods:

$$\mu_i \approx \frac{1}{N}\sum_{n=1}^{N} x_j^{(n)}(t)^i \tag{4.7}$$

[2] 1/30 or 1/60 of a second represents a minimum response time before which users can conceptualize poor quality of service.

where N is the number of data samples and $x_j^{(n)}(t)$ is the n^{th} data sample of input $x_j(t)$. Assuming N is sufficiently high, a frequency histogram can be collected at the same time to represent the distribution of each element within $X(t)$.

With this empirical distributions of $X(t)$, both requirements are meet to determine the distribution of $y(t)$. First, an artificial sampling of $X(t)$ must be created, this is a simple matter with the empirical distributions of $X(t)$ already collected. A uniform random variable can be utilized to index into the "buckets" of the histogram. These artificial samples are then used to evaluate the aPCE model assuming a fixed control policy, $\boldsymbol{s}(t)$. The result is a collection of histograms representing the distributions of power, performance, and energy efficiency as shown in Figure 4.4. From this, it is a simple mater of determining any statistical quantity needed for the DEM controller.

4.4.5 Offline QoS Model Learning

A common approach in computing $\hat{y}$ is to perform a random sampling of $y(t)$ and $X(t)$ through simulation of $f(\cdot)$. These approaches generate N data samples and response labels, $\{X(t_1), y(t_1)\}$, $\{X(t_2), y(t_2)\}$, ... , $\{X(t_N), y(t_N)\}$. Therefore, there exists an N-by-M matrix, $\boldsymbol{X}$, which contains a vector of N samples of X. Additionally, the corresponding N samples of the QoS responses are stored in a vector, $\boldsymbol{y}$, of length N. It is possible to determine the model coefficients of (4.2) by solving for the unkown vector $\boldsymbol{c}$ in the following linear equation

$$\boldsymbol{y} = \boldsymbol{c}\phi(\boldsymbol{X}), \tag{4.8}$$

where $\phi(\boldsymbol{X})$ is a N-by-R matrix containing the transform for each sample of X into R orthogonal polynomials (see the Appendix A for details), and $\boldsymbol{c}$ is a vector containing the aPCE model coefficients.

Given (4.8) it is possible to estimate the performance (response) for any input

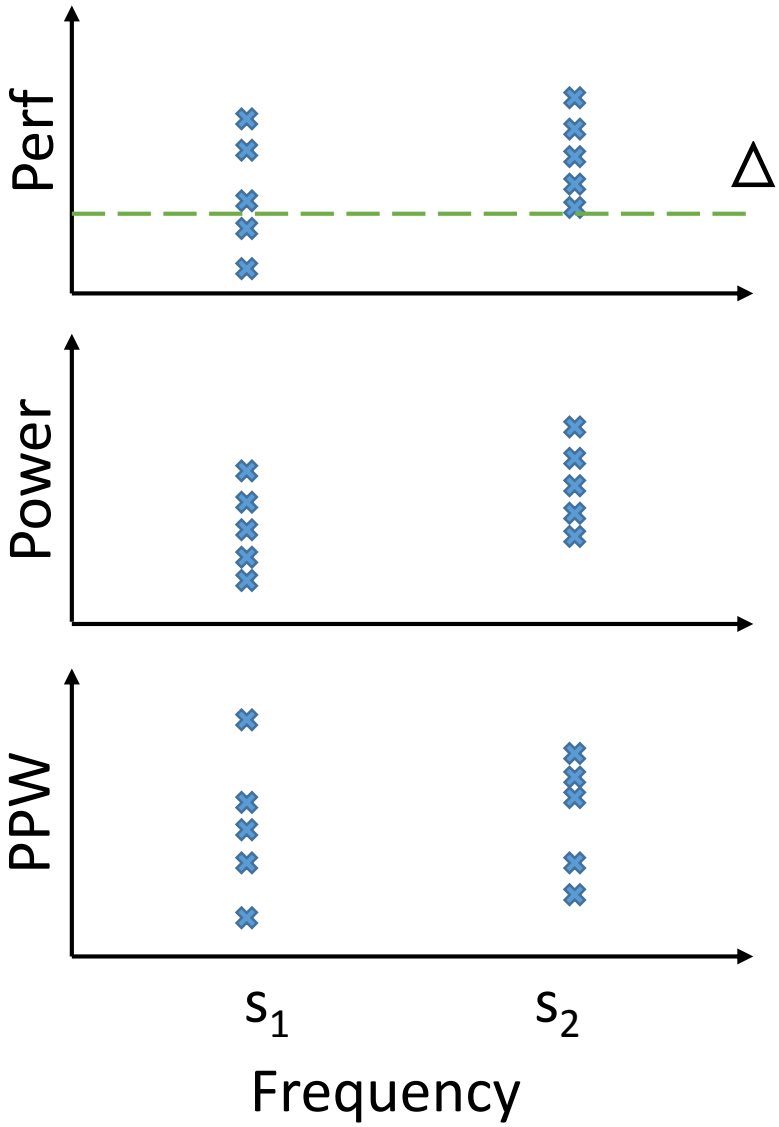


Figure 4.4: Illustrations of the empirical distributions of power, performance, and energy-efficiency used to represent the actual probability distributions.

Algorithm 1: Online Performance Prediction

$\boldsymbol{c} = \mathbf{1}$;

$n = 1$;

while *Application is still running* **do**

Estimate: $\hat{X}(t_{n+1})$;

Predict: $\hat{y}(t_{n+1}) = \boldsymbol{c}\boldsymbol{\phi}(\hat{X}(t_{n+1}))$;

Wait: δ time until the actual IPS value is ready;

Read: $y(t_{n+1})$ and $X(t_{n+1})$ from the PMUs;

Update: $\boldsymbol{c} = \boldsymbol{c} - 2\eta\frac{\left(\boldsymbol{c}^T\boldsymbol{x}(t_{n+1}) - y(t_{n+1})\right)}{\boldsymbol{c}^T\boldsymbol{c}}$;

$n = n + 1$;

combination of X. Equivalently, one can determine the coefficients c_i for each aPCE basis function $\phi_i(X)$ using least squares error minimization over a training set. That is, the argument of

$$\min_{\boldsymbol{c}} \quad \sum_{n=1}^{N} \left(y(t_n) - \hat{y}(t_n)\right)^2 \tag{4.9}$$

$$\text{s.t.} \quad \hat{y}(t_n) = c_0 + \sum_{i=1}^{R} c_i \phi_i(X(t_n)), \tag{4.10}$$

must be solved

In the event that higher order models are needed, regularization techniques will be required to prevent overfitting. For example, the error minimization problem (4.9) can be replaced by the LASSO optimization problem [95] combined with validation methodologies (e.g. N-fold cross validation [96]).

4.4.6 Online QoS Model Learning

The model learning technique presented in the previous section serves as an optimal means of determining the coefficients to the QoS prediction model. However, this method requires a sampling of X such that all sources of variability are exposed. In practice this may result in a large number of experiments which may require an excessive amount of time. This work therefore proposes an online prediction algorithm which tunes the prediction model for the current system status. The proposed online prediction algorithm is based on stochastic gradient descent techniques such as the Wiener filter which is commonly used in system response modeling [97]. Algorithm 1 outlines this procedure.

The algorithm begins by assuming the initial value of the coefficients, $\boldsymbol{c}$, are set to 1; however, any vector of real values is permitted. Next, while the application is running, the algorithm produces a sequence of predictions at times t_1, t_2, ..., t_n, ... such that the time between any two consecutive time samples is δ. However, the aPCE model requires knowledge of $X(t_n)$, the model inputs at time t_n, in order to predict the performance response $y(t_n)$ for the same period. Since this value will not be available until time t_{n+1}, the algorithm must compute an estimate $\hat{X}(t_n)$ of $X(t_n)$. This work examines four choices to perform this estimate. The simplest of which is ***Last Observed*** in which $\hat{\boldsymbol{x}}(t_{n+1}) = X(t_n)$. Next, this work evaluates the ***Linear*** method which uses the previous two samples of X to estimate the rate at which $X(t_n)$ is changing. That is $\hat{X}(t_{n+1}) = 2X(t_n) - X(t_{n-1})$. The final two methods, ***Moving Average*** and ***LMS Filter*** utilize a window of the past 5 data samples in order to predict the next sample. As the name ***Moving Average*** suggests, $\hat{X}(t_{n+1})$ is estimated by averaging the last 5 samples of X. Similarly, LMS Filter also averages the past 5 samples; however, it assigns a weight to each of these samples using steepest

descent method [98].

With $\hat{X}$ estimated, the algorithm derives estimates of performance value, $\hat{y}$, for the next time interval using the aPCE model with the current estimate of $\boldsymbol{c}$. Once the time interval has passed, the algorithm reads the actual values of $X(t_{n+1})$ and $y(t_{n+1})$ and use this information to update the estimate of $\boldsymbol{c}$, using the normalized gradient of the least squares minimization objective in (4.9) along with a learning rate parameter $\eta \in (0, 1]$. This learning rate parameter offers a mechanism of how responsive the model update is to recent samples. For the experiments discussed in this work, a learning rate parameter of 0.9 is used. This was determined as a good choice for the system through trial and error; however, methods exist to determine the optimal learning rate [95]. This process is repeated until the application terminates. Note that, for all experiments, the online learning method is used unless otherwise noted.

4.5 Experimental Setup

In this section, we describe the setup of the experiments and the validation results for the execution time and power models, evaluated on real platforms.

4.5.1 Real-Device Experimental Platform

Platform Specifications

The experiments were conducted on a Google Pixel Smartphone [99] housing Qualcomm's MSM8996pro chipset [100]. The phone features 6 on-chip performance monitoring units (PMU). The 6 PMU's can be programmed to observe various hardware events such as L2 cache misses, branch mispredictions, and main memory page faults. We sample these PMU's at a rate of 10 samples per second. The power sensors and PMU's are sampled every 10ms. The MSM8996pro offers a heterogeneous architec-

Table 4.4: Parameters for the Pixel Smartphone and Qualcomm MSM8996pro chipset.

	Power Cluster	Performance Cluster
Number of Cores	2	2
Architecture	Kyro	Kyro
Instruction Set	ARM v8a	ARM v8a
Frequency	0.3-1.59GHz	0.3-2.15GHz
L1 Cache Size	32KB I & 32KB D	32KB I & 32KB D
L2 Cache Size	512KB	1.5MB
L3 Cache Size	4MB	

ture with 2 cores tuned for high performance and 2 cores tuned to low power. The two cores within a cluster share a common L2 cache and both clusters share a common L3 cache. Table 5.1 lists the architectural specification of the device.

The Google Pixel's battery was removed and replaced with a constant voltage source via the Monsoon power monitoring unit [101]. The unit measures the total system's voltage and current at a rate of 5000 samples per second.

The experimental platform runs a rooted Android 7.2 OS. The applications of interest were written in C and cross-compiled on the host machine with the ARM-Android NDK toolchains [70]. The binary is pushed to the device and is launched from the host machine via a wireless connection using the android debug bridge terminal.

Simulating Random User and Network Requests

Typically, mobile applications are responsive in nature–providing a quick burst of computation after a service request is received from either the user or the network. To implement the bursty nature of mobile applications, we constructed a testing

Table 4.5: Parameters for Various Distributions Used to Determine Inter-Arrival Time of Workload Requests in Section 4.8.

Distribution	3G	4G
Constant	2250ms	250ms
Uniform	[1000ms, 3500ms]	[0ms, 500ms]
Exponential	mean=2250ms	mean=250ms

framework in which a client program runs on an external desktop and, via a wireless network connection, sends workload requests to the mobile platform – the response time of which is a random quantity determined by the selected workload. The testing framework allows us to manually adjust the delay between workload requests. We generate delay values according to three distributions, the parameters of which are determined by typical network round trip time's seen in 3G and 4G networks [7]. These parameters are outlined in Table 4.5.

4.5.2 Benchmark Applications

In order to provide a good breadth of mobile applications, we evaluate a number of applications which (1) have been utilized in prior works for predicting the performance of mobile and interactive workloads [81, 41, 18, 79, 65] and (2) represent workloads which may be present on future smartphones.

Interactive Workloads for QoS Prediction

User satisfaction is the driving force of how successful a mobile application will be; however, to estimate the applications QoS, additional information or hints can be propagated by the application itself to determine data input complexity and the complexity of servicing this data.

Web Browser and HTML Viewer – The web browser serves as one of the most utilized applications on modern smartphones. A user's satisfaction with a web page is directly related to the time needed to load and render the page [1]. We therefore explicitly denote QoS as the time required to load and render the page after the request has been made. Additionally, many modern applications are constructed using HTML as it allows applications to be updated with more ease. We utilize the Google Webview [102] framework to load offline webpages. These webpages are selected from the 10 most viewed webpages according to the Alexia service [103]. The composition of a web page is determined by its DOM tree – a structure whose nodes are labeled by a unique tag describing the node's function (e.g. java scripts, images, links, etc.) along with multiple attribute fields providing additional information for the node (e.g. the dimensions of an image to display). We modify the browser's source code to record the load time of each web page as well as record the count of each tag type and attribute which represents the page's complexity of the web page according to [18].

Workloads for IPS and Power Prediction

In order to properly predict the QoS of an application, additional effort by the application designer is needed to propagate important information to the QoS prediction algorithm. Instead we demonstrate that our performance prediction framework can be used to estimate the instructions-per-second of the processor with only information from the on-chip performance monitoring units, requiring no interaction with the fore-ground application. We examine the following mobile workloads.

Security/Communication Benchmarks – A common task in smartphones is the transmission of data via a wireless/radio network. This data is encrypted to mitigate packet snooping. We examine two common algorithms used in secured data

transmission: *sha* and *aes* [104]. These workloads also serve as a comparison point with prior works in mobile performance monitoring [81].

Interactive Gaming – On modern mobile processors, gaming applications are common place. In this work we examine the CPU portion of these gaming applications – the AI which is responsible for responding to user's request and interactions. The first is *2048* [105], a puzzle game with extremely limited computational overhead and who's response time is largely dictated by IO requests. The second is *gnu go* [106] which incorporates computationally heavy AI techniques such as breadth first search and decision trees.

Compression – Due to limited storage and expensive data transmission it is common for applications to first compress data. We examine the *bzip2* [107] benchmark under several different inputs such as text files, images, videos, programs, and file systems.

4.6 Analyzing the Load Time of Mobile Web Browsing

Web browsers and HTML viewers serve as some of the most significant applications on mobile devices. In fact, since 2014, the number of mobile users who access web pages have surpassed the number of desktop users [108]. These applications are highly interactive and thus user satisfaction is vital to maintain a website's user base [1]. Specifically, the QoS of the web browser is defined as the time needed to load and render a webpage. It is therefore necessary to specify some deadline Δ such that the webpage QoS will meet the deadline and ensure user satisfaction. Although smartphones have limited control over network conditions, mobile devices can adjust the web page load and render times by dynamically scaling the CPU frequency [41, 18]. While increasing the CPU frequency will reduce load times, doing so can increase the total energy consumption resulting in reduced battery lifetimes.

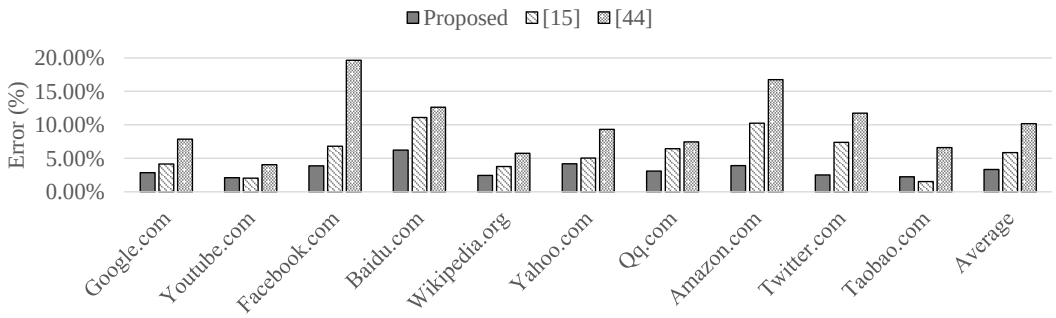


Figure 4.5: The prediction error (%) when using the proposed aPCE modeling framework and two state-of-the-art prediction methods. Lower is better.

The HTML representation of any given website can be defined by its document object model, often called the DOM tree. The DOM tree is a hierarchy of the various objects within the website (e.g. links, images, scripts, etc.) while each object is categorized by a *tag* and given a list of *attributes*, both of which are defined by the HTML language. The total website load time is thus a function of the computational complexity of each of these objects. Zhu and Reddi [18] proposed utilizing a linear model, based on counts of each specific tag and attribute within a given webpage. However, this methodology fails to consider the effect of run-to-run variations, which can significantly contribute to variations on the performance of mobile applications. In [41], to model the performance of each webpage, a sample distribution of the load times were observed offline for each CPU frequency. These distributions are then used to determine the likelihood of loading a webpage within a specified deadline for each given frequency. While this method captures the statistics of run-to-run variations, it fails to predict the load time for any specific execution of the web browser.

This work proposes to combine these two approaches, by constructing an aPCE model such that the input parameters are a composite of the architectural events of the mobile device along with webpage characteristics. This will provide an estimate of the load time sensitive to run-to-run variations as well as input complexity. The proposed

QoS prediction framework is compared against two prior works which represent the state-of-the-art.

- *Proposed* – The aPCE based QoS prediction framework proposed in this manuscript. This work proposes a collection of system features as shown in Table 4.2 along with application specific characteristics to represent a set of stochastically varying system parameters which dictate the load time of the web page. In addition, for web browsing, the compositions of the DOM tree representing a given webpage is considered.

- Gaudette et al. *[41]* – A statistical distribution parametrized by the CPU frequency is modeled for each application/input combination. These distributions are then leveraged to determine the likelihood of producing a specific QoS at a given frequency.

- Zhu et al. *[18]* – A QoS prediction framework specifically for the web browser. The authors propose utilizing a linear model where the inputs are only the counts of each specific tag and attribute within a given webpage.

A series of experiments was conducted which loaded the top 10 most visited webpages 10 times each and at each possible CPU frequency (total of 900 experiments). The pages were stored offline in order to remove the non-measurable effects of network delay. Each of the considered models are constructed using the method of least squares to determine the model coefficients using half of the experiments chosen randomly, and then verified the model's accuracy using the other half. Figure 4.5 shows the the proposed model's prediction error and the comparison with the methods from [41] and [18]. It can be seen that for 8 out of the 10 webpages (except youtube and taobao), the proposed prediction model outperforms both of the state-of-the-art alternatives–in part due to the additional information which was gathered about

the system state via performance counters. On average, the proposed methodology reduces prediction error by 42.7%.

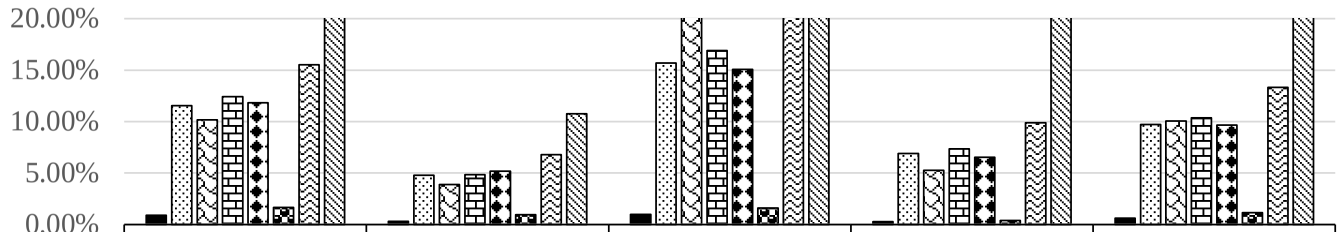
4.7 IPS Performance Prediction

In the previous section, an exploration of modeling QoS specific to the web browser was given. In that scenario, the application was able to define its own QoS metric and also provide hints (additional model parameters) to improve overall user satisfaction. However, this places an extra burden on the application designer and may not always be available. Instead, QoS can be generalized to be instructions per second (IPS) of the processor – a measure of the rate of progress the processor is making toward completing some fixed number of instructions.

4.7.1 Offline Construction of Performance Modeling Using aPCE

This section demonstrates the appropriateness and value of utilizing aPCE models to predict system performance with changing system conditions. For this experiment, the set of observable inputs is limited to non-intrusive values only. That is, this input set can be constructed with no modification of the applications or even knowledge of the application running (i.e. the values in Table 4.2). For each application, several time traces of the actual performance value along with the observable inputs are recorded. The model was then trained on half of the samples chosen at random while the other half was used to determine the testing error.

Both a first and second order aPCE model was constructed using this data; however, it was observed that a second order aPCE model produces no benefit in terms of accuracy. Thus, this work only presents the first order aPCE model.



		2048	aes	sha	bzip2	Average
	Proposed - Offline	0.89%	0.30%	0.97%	0.28%	0.61%
Proposed - Online	Last Observed	11.54%	4.78%	15.70%	6.90%	9.73%
	Linear	10.18%	3.90%	20.87%	5.26%	10.05%
	Moving Average	12.43%	4.84%	16.90%	7.35%	10.38%
	LMS Filter	11.82%	5.17%	15.06%	6.53%	9.65%
	Oracle	1.65%	0.94%	1.61%	0.38%	1.15%
	Ondemand	15.53%	6.78%	21.12%	9.88%	13.33%
	Static	83.29%	10.75%	60.78%	31.01%	46.46%

Figure 4.6: Comparison of prediction error between common prediction methods used by the Android Governors, Ondemand and Interactive, against the proposed aPCE modeling framework under numerous mobile workloads. Lower is better.

This work compares the proposed aPCE based prediction method against two commonly used prediction methods. The first is simply assuming the last observed performance value as the current value. Such methods are used in processor frequency control mechanisms such as the *Ondemand* and *Interactive* frequency governors – the default governors used by the Android OS. Additionally, the proposed method of prediction is compared against an offline analysis technique which would provide a single IPS value for the workload based on the average IPS seen in the training set (i.e. pre-characterization of the workload such as suggested by [41]). This is denoted as *Static*. The prediction errors are presented in Figure 4.6. By incorporating additional information into the model (i.e. the performance counters), the aPCE model is able to better adapt to changing and uncertain system conditions. Additionally, since aPCE outperforms *Ondemand* by a factor of over 25X, one can conclude that utilizing the last observed value as a means of prediction is insufficient for mobile workloads. Overall, the aPCE prediction framework performs far better than either *Ondemand* or *Static*.

4.7.2 Online IPS Prediction

A common application of IPS prediction models is use in dynamic energy management techniques. These techniques aim to minimize the energy used by the system or equivalently, to maximize the systems energy efficiency. These techniques must be agnostic to the applications which are running and as such must learn the current system characteristics quickly in order to make effective control decisions. In this environment, it is impractical to assume each application is analyzed offline beforehand to construct the prediction models.

In this section, the effectiveness of the purposed online learning algorithm (Section 4.4.6) is presented. Like the offline analysis, the online learning utilizes a first

order aPCE model and the same set of performance monitoring events as in Table 4.2. This work evaluates the effectiveness of the performance prediction technique subject to various methods of determining $\hat{X}(t_n)$ – described in detail in Section 4.4.6. Additionally, this work compares against the prediction method used by the *Ondemand* frequency governor. Finally, this work observes the case in which a static IPS value is assigned to each application by averaging all observed values from prior executions of the program. Figure 4.6 shows the prediction error for each of these schemes.

For all applications, every aPCE modeling technique outperforms the *Ondemand* and *Static* prediction methods. Additionally, one can see that even utilizing naive methods of modeling $\hat{X}(t_n)$ result in better prediction accuracy than Ondemand and static.

4.8 Power Prediction using aPCE

Thus far, it has been demonstrated that the proposed framework is effective predicting performance metrics; now it will be shown that the proposed modeling methodology is capable of capturing power variations as well. The defacto power prediction method for computing systems is the last observed value. This method has nearly zero overhead and has proven to be sufficient for a wide variety of server and desktop workloads where power fluctuations are relatively small to the mean power value [109, 110, 111]. However, mobile applications exhibit much larger power fluctuation due to bursty computation phases, high levels of system non-determinism, and are much more sensitive to the external environment such as ambient temperature, network conditions, and so on. Thus, it is expected that the last observed value will be inaccurate for the mobile platform.

To implement the bursty nature of mobile applications, a testing framework was constructed in which a client program runs on an external desktop and, via a wireless

network connection, sends workload requests to the mobile platform – the response time of which is a random quantity determined by the selected workload. The testing framework allows us to manually adjust the delay between workload requests. Delay values were generated according to three distributions, the parameters of which are determined by typical network round-trip time's seen in 3G and 4G networks [7]. These parameters are outlined in Table **??**.

A second order aPCE model using the same input parameters as for the IPS prediction was determined to be sufficient to accurately predict the power every 100ms. Figure 4.7 shows the results for 5 workloads which are likely to be repetitively executed in a bursty manner. First, the stable case is analyzed. That is each workload is continuously executed with no interruptions. One can see that while the aPCE modeling framework produces lower prediction error (2.11% on average), last observed still maintains a relatively small prediction error of 7.47% on average.

Once interruptions are introduced into the workload, last observed value begins to exhibit much greater errors – up to 49.8% depending on the workload and interruption rate. This is largely due to the fact that last observed value does not take into account any of the system conditions for the next cycle. In contrast, the aPCE prediction framework maintains a relatively constant error despite the workload and the interruption rate due to the fact that the model is constructed such that system conditions are utilized. Overall, the aPCE prediction framework reduces the average prediction error by a factor of **9.78X** and maintains an error of less than 6%.

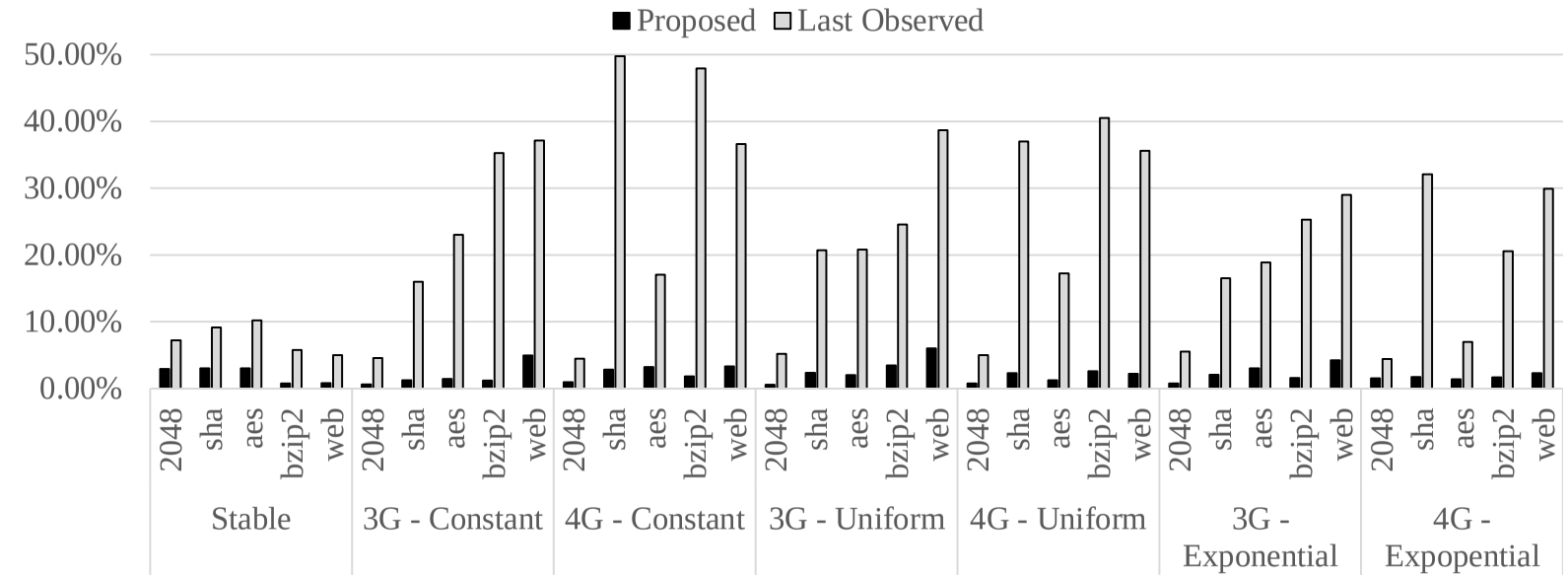


Figure 4.7: Comparison of the error of power prediction methods, last observed and the proposed aPCE framework. Workload inter-arrival time was varied based on the parameters of common network delays – 3G and 4G. Lower is better.

4.9 Energy Efficiency Analysis of Mobile Devices

Using the aPCE models created in Sections 4.7 and 4.8, now an in-depth analysis of the performance, power, and energy efficiency of a mobile workload can be performed. A major benefit these models provide is the additional information regarding the distribution of performance and power values as a function of the processor frequency. This section limits itself to a single example application, GNU Go, to highlight the importance and usefulness of having the knowledge of the ***performance and power distributions***.

Once the aPCE models have been constructed for performance and power (Section 4.4), the associated distribution functions can be recreated using knowledge of the distributions of the input parameters – these distributions are easy to estimate via the same moments used to construct the polynomials. From these distributions, samples of $\boldsymbol{X}$ are generated and then feed to the aPCE model inorder to determine the corresponding y values. With this information, it is possible to evaluate the effect of any given set of operation conditions. For example, given a soft-deadline, energy management algorithms can utilize the distribution of application run-time to determine the likelihood of reaching a given deadline target for any given control policy. If the deadline is relatively unimportant, then the algorithm can relax the likelihood requirement to potentially save energy [41]. It should be noted that while [41] provided the first step to model the QoS distributions, it relies on an assumed form of the distribution (gamma) which is not accurate.

Alternatively, it is possible to analyze the PPW also known as the energy efficiency of the system – the ratio between performance and power which equivalently determines the number of instructions which can be evaluated per joule of energy. This value is useful when trying to optimize both battery life and performance.

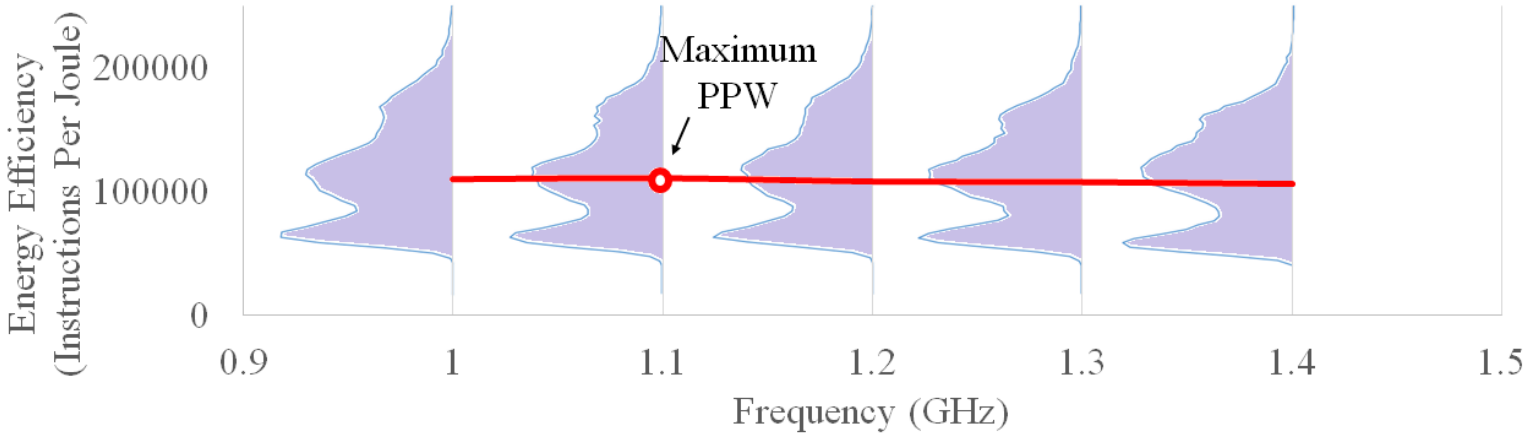


Figure 4.8: The distributions of energy efficiency as a function of CPU frequency. The solid trend line shows the expected energy efficiency of each frequency.

The distributions of energy efficiency for five operating frequencies are constructed using the aPCE models. Approximately a 4% error exists between the measured and the calculated distributions. The relationship between frequency and energy efficiency is plotted. Each of the distributions is overlaid with their associated frequency. This plot will help us understand not only the expected energy efficiency as a function of frequency, it will also allow us to determine the range of possible energy efficiency values at each frequency. Figure 4.8 shows this plot. For this workload, the optimal operating frequency turns out to be 1.1GHz on average. However, at lower frequency, variation in energy efficiency is higher, implying that any given level of energy efficiency is more likely to be achieved at higher frequencies.

The system can leverage this information in various ways. For example, consider the case in which a mobile device's battery is critically low. While on average it is most energy efficient to operate at 1.1GHz the likelihood of reaching this energy efficiency is lower than at higher frequencies. Moreover, there is much greater uncertainty at 1.1GHz and below. Therefore, operating at a higher frequency may result in greater efficiency more often. The controller could leverage this statistical information to use a higher operating frequency such that there is a greater assurance the application will complete within the remaining battery lifetime.

4.10 Summary of Contributions

In this chapter a new prediction framework is presented which is used to determine the quality of service and power consumption for mobile applications. The key contributions of this framework are summarized below:

- A QoS and power prediction framework is demonstrated which is able to capture the fast changing, dynamic workloads with sources of uncertainty – a common environment to mobile devices.

- This framework can be customized based on hardware specific features such as those visible from on-chip performance counters, as well as application specific features such as image complexity for the image similarity search algorithm.

- The proposed prediction method is independent of input distributions. Instead the framework determines the optimal form for any arbitrary distribution. This provides improved convergence properties over prior works which assume a standard distribution [41].

- An important property of the framework is the orthogonality of the polynomial basis functions used to represent a QoS response surface. This allows for models to increase model order without re-computing lower order coefficients.

4.11 Conclusion

In this chapter, a framework is presented to predict QoS (performance and power) for mobile devices. The framework captures the effects of user-induced interruptions along with run-to-run variations within these quantities, and constructed accurate models for these quantities. The models are constructed using various system events such as cache misses, page faults, and TLB refills which can easily be recorded on modern mobile processors. The models where constructed using arbitrary polynomial chaos expansions which approximates stochastic systems by a set of orthogonal polynomial bases. These polynomials are constructed using only sample moments of the system events, requiring significantly less computation than with traditional techniques. To demonstrate the proposed approach, performance and power models were developed for numerous mobile workloads running on actual hardware. The proposed approach was shown to reduce the error by a factor of **6.1X** and **9.4X** for performance and power estimates respectively, over the prediction method used by

Android frequency governors. Finally, a use case of how to utilize the additional information concerning uncertainty is provided. This chapter showed how to construct the probability density functions corresponding to performance, power, and energy efficiency and leverage this additional information to better select the optimal CPU frequency operating point for mobile workloads.

Chapter 5

A ROBUST CONTROLLER OF MOBILE DEVICES: MAXIMIZING ENERGY-EFFICIENCY

In recent years, mobile devices have become more and more ingrained in everyday life. As a result, users place higher and higher expectations on these devices, requiring a large number of different types of applications to be serviced in a relatively quick period of time. These workloads are often bursty in nature and exhibit a large amount of variability in performance and power from run to run. To better service these demands, mobile SoC's have introduced a slew of heterogeneous processing elements such as GPU, DSPs, and additional CPU cores. However, due to high importance of user satisfaction, the limited energy resources available to mobile devices, and the lack of active cooling mechanisms, there is a strong need to develop practical dynamic energy management techniques.

Existing DEM techniques have primarily been developed with desktop and server workloads in mind. Such workloads often have very stable power and performance profiles, and thus do not capture the highly variable and user centric nature of mobile workloads. This chapter proposes a closed loop controller aimed at improving the energy efficiency of mobile devices. The controller is unique compared to all others as it takes into consideration the distributions of QoS and power – a vital and necessary feature to accurately predict the non-deterministic mobile workloads. The proposed controller is implemented on an actual smartphone device running the Android operating system. Able to effectively acclimate to the non-deterministic conditions of the system, the proposed controller improved the energy efficiency of the mobile device by **19%** over the existing governor implemented on Android devices.

5.1 Introduction

As mobile devices become more and more ubiquitous to daily life, greater demands and expectations are placed on these devices – multitasking, higher screen resolutions, and faster frame rates just to name a few. To service these demands, SoC's have introduced a wide array of complex, heterogeneous interacting components; however, this strategy comes at a price of increased system complexity and power demand. Unfortunately, despite these improvements the potential performance of these devices is often limited by several factors. For example, the maximum possible temperature the SoC or the user can tolerate. High temperatures lead to significantly reduced operational lifetimes and increased power dissipation, thus, it is vital to properly manage the thermal state of the mobile device. Additionally, shared resource contention between the SoC components can degrade overall system performance while potentially over exerting the finite energy available to the mobile device. Therefore, to ensure high performance levels (and by extension high user satisfaction) it is vital to understand the complex interactions between the various system components as well as system characteristics such as power, performance, and temperature.

In order to overcome the thermal and shared resource limitations of the mobile device it is necessary to improve the energy efficiency of the SoC. While this can be addressed at various hardware design stages, the need for system level, dynamic control of the heterogeneous components will always be needed. Dynamic control is able to provide further energy efficiency improvements by optimally tuning the system components to adapt to the changing usage conditions of the device. Such runtime techniques are referred to as *Dynamic Energy Management* (DEM) and is the focus of this work.

A common metric for gauging energy efficiency is *performance per watt* (PPW)

which equivalently measures the the number of instructions which can be executed per Joule of energy. Alternatively, with additional information provided by the application layer, we can determine energy efficiency by *QoS per watt* (QPW). For example, in video decoding and rendering, one can measure the frames per second per watt – a more direct measure relating to user satisfaction.

DEM for heterogeneous SoCs is an extremely challenging issue which involves solving a complex multidimensional, non-linear optimization problem of a multi-input, multi-output (MIMO) system. Often this is tackled by reducing the problem space in order to simplify the problem. For example, dynamic voltage and frequency scaling (DVFS) is one of the most common mechanisms to perform DEM. Rather than consider every controllable parameter on the SoC, DVFS simply modifies the operational voltage and frequency of the various system components in order to increase performance at the cost of increased power consumption. Despite the pervasiveness of mobile systems, most DEM mechanisms are still based on methods developed for desktop and server platforms – relaying on deterministic and stable workload patterns. However, this is not the case for mobile systems which are often subjected to numerous sources of non-determinisms as well as bursty computation phases. As a result, the previously used techniques for desktops and servers are highly sub-optimal on mobile devices. A DEM controller for mobile devices must determine globally optimal frequency settings for each processing element while accounting for cross-PE performance, power, and temperature interactions. It must be flexible enough to accommodate the changing demands placed on the system by the user while guaranteeing QoS under a finite energy budget. Being a non-deterministic MIMO system, the design of a DEM controller for mobile devices with the aforementioned characteristics while assuring stability and robustness is a much more challenging task than its desktop counterpart.

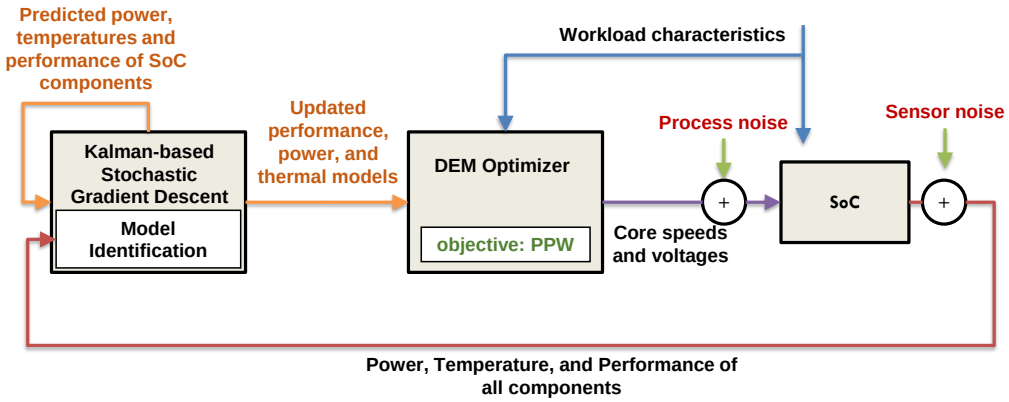


Figure 5.1: Structure of the closed-loop controller with its various components.

In this chapter we describe the design of a closed-loop controller for DEM of mobile systems. The controller is smart enough to predict upcoming performance, and thermal conditions in the presence of numerous sources of system non-determinisms as to make optimal voltage and frequency decisions in an informed manor. The controller is intelligent as it is capable of tuning the prediction models using a Kalman-based stochastic learning technique. Figure 5.1 illustrates the controller's structure. The DEM optimizer determines the optimal DVFS states for the next time interval based on the computed power and thermal models, and workload characteristics. A Kalman-based stochastic gradient descent filter (kSGD) is used to reduce the prediction error caused by model inaccuracies or by sensor noise.

To predict the optimal voltage/frequency setting in real time, the controller incorporates detailed power and thermal models which consider inter-device interference and numerous system non-determinisms such as memory contention and other architectural delays. These models are "black box" in nature, tunable to most any system regardless of the system complexity. Additionally, the controller utilizes temperature aware power models to accurately determine the leakage power component and adapt to poor thermal conditions.

The controller is implemented and evaluated on an actual Android-based system– the Google Pixel. This device houses Qualcomm's MSM8996pro SoC along with numerous thermal and performance sensors. Our experimental results on maximizing energy efficiency demonstrates a **19%** over the existing governor implemented on Android devices.

Overall, the proposed controller described in this article advances DEM for mobile processors in the following ways:

- it achieves better QoS, power, and temperature prediction over existing controller models by taking into account dynamic, stochastic workload characteristics;
- the prediction method is computationally efficient as it employs simple yet accurate polynomial chaos models;
- the controller is able to estimates per-component power consumption without the aid of power sensors – something that many SoCs lack.

5.2 Model Identification and Learning

In this section a discussion on the formulation of the various thermal, power, performance, and QoS models is given. This is done to ensure that this chapter is self-contained; however, readers should address Chapters 2 and 4 for more detailed information.

5.2.1 Identification of Thermal Model

Compact thermal models are among the simplest mechanism to characterize a processor's power-temperature relationship with sufficient accuracy. The compact thermal models use the electro-thermal analogy relating heat generation, spreading

and storing to the electrical circuits concepts of current sources, resistors and capacitors, respectively. This work employs a simplified version of the HotSpot compact thermal model [36] which assumes each processing element (i.e. cpu core, gpu, dsp, etc.) and the package temperature is represented by one thermal node. Additionally, it is assumed that the thermal capacitance associated with the die thermal nodes are ignored, as the die thermal constants are usually within a few milliseconds and are not noticeable when sensors have very low sampling rate (30 ms or more). We summarize these assumptions along with the model as shown in Figure 5.2.

Each processing element of the SoC has a power source P_i. This power varies over time based on the workload characteristics along with the thermal conditions of the processing element. Each core is connected to the package through a vertical resistance and to every adjacent core through a horizontal/lateral resistance. Combined these form a symmetric resistance matrix $\boldsymbol{R}$ where R_{ij} denotes the resistance between processing elements i and j. The package is lumped into a single thermal node connected to the external skin temperature of the device and finally to the ambient. It is assumed that each processing element poses a thermal sensor such that T_i is observable along with the package, T_p, and skin, T_s.

As discussed in more detail in Section 5.2.2, the power dissipation of each block is dependent on the speed, voltage, and thermal state of the block, along with workload characteristics. Given this the vector of thermal profiles, $\boldsymbol{T}(t)$, can be expressed using a state space model

$$\frac{d\boldsymbol{T}(t)}{dt} = -\boldsymbol{C}^{-1}\boldsymbol{G}\boldsymbol{T}(t) + \boldsymbol{C}^{-1}\boldsymbol{P}(\boldsymbol{s}, \boldsymbol{v}, \boldsymbol{T}, t) \tag{5.1}$$

where $\boldsymbol{P}$, $\boldsymbol{s}$, and $\boldsymbol{v}$ are a vector comprised of each block's total power dissipation, speed, and voltage at time t respectively. $\boldsymbol{G}$ and $\boldsymbol{C}$ are $n \times n$ matrices relating the thermal conductance and capacitance between any two pairs of the n blocks. We can

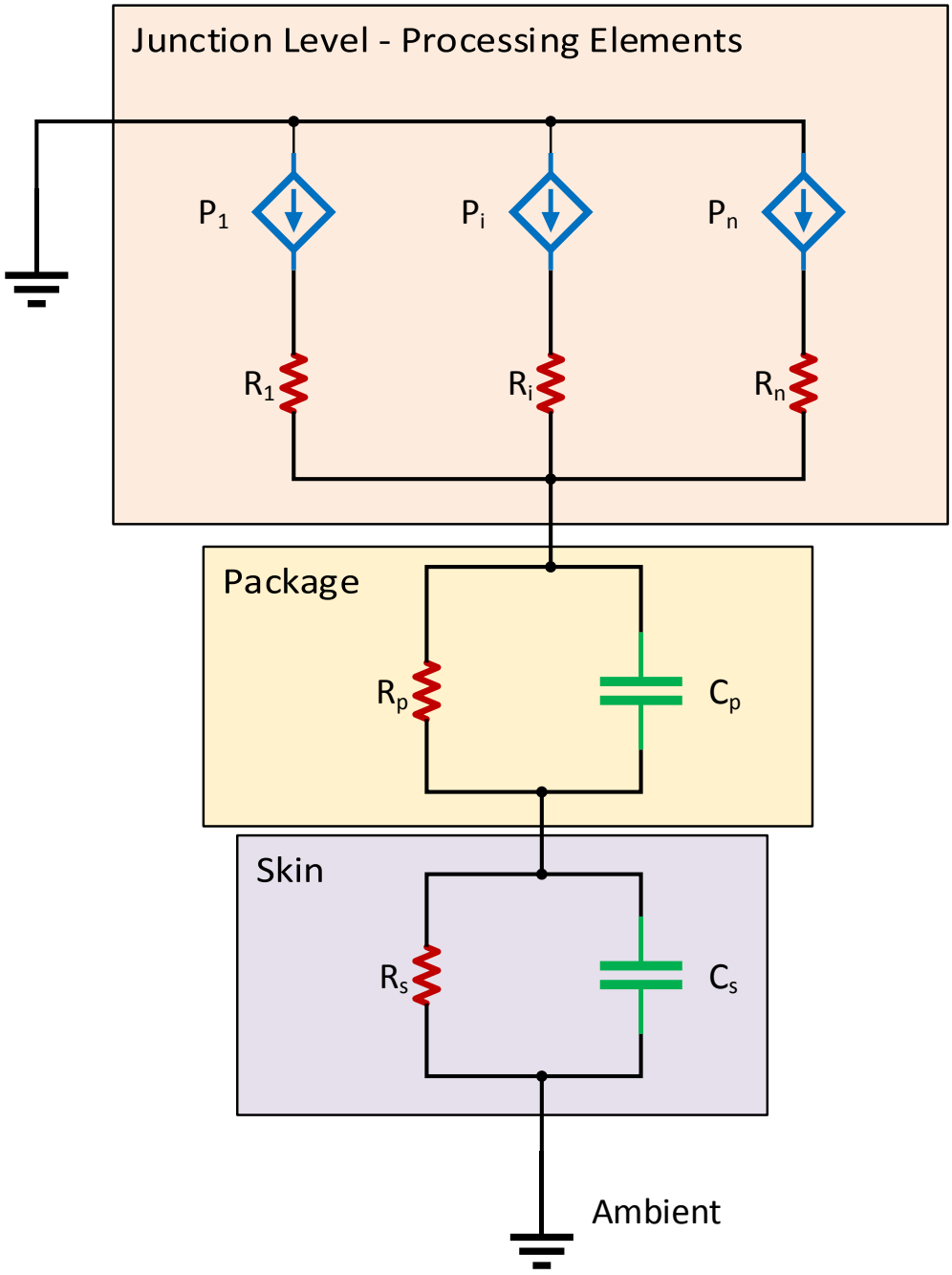


Figure 5.2: Compact thermal model used in this work.

write this equation more succinctly by substituting $\boldsymbol{B} = \boldsymbol{C}^{-1}$ and $\boldsymbol{A} = -\boldsymbol{BG}$ thus

$$\frac{d\boldsymbol{T}(t)}{dt} = \boldsymbol{AT}(t) + \boldsymbol{BP}(\boldsymbol{s}, \boldsymbol{v}, \boldsymbol{T}, t). \tag{5.2}$$

The coefficient matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ can be derived mechanistically given sufficient information of the thermal capacitance and resistance of the SoC layout and fabrication material. Alternatively, these values can be determined analytically by observing a time series of $\boldsymbol{T}$ and $\boldsymbol{P}$ values and utilizing least mean squares fitting techniques.

5.2.2 Identification of Power Model

As discussed in the prior section, this work assumes that the SoC processor can be segmented into seperate logical blocks (e.g. arithmetic units, floating point units, caches, etc). In this section, we will show how well established power models for CMOS logic can be translated into the aPCE framework discussed in Chapter 4. It has been well studied that the power consumption of a singular block within the processor is typically modeled as

$$P_i(t) = P_{dyn,i}(t) + P_{lkg,i}(t) + P_{base,i}(t), \tag{5.3}$$

where $P_{dyn,i}(t)$, $P_{lkg,i}(t)$, $P_{base,i}(t)$ are functions describing the dynamic, temperature-dependent leakage, and baseline or static power components respectively for a given block b. The sum of power consumption of each of the $\mathcal{B}$ blocks within the system constitutes the total system power consumption:

$$P_{sys}(t) = \sum_{i \in \mathcal{B}} P_i(t). \tag{5.4}$$

Leakage Power

For any given block, Leakage power is known to have an exponential dependence on the block's temperature and supply voltage. The exact equation is difficult to derive

analytically, hence it is usually derived via simulations and data fitting. The authors of [52] provide the leakage power for 65 nm.

$$P_{lkg,b}(t) = k_{b,1} v_b(t) T_b^2(t) e^{\frac{\alpha_b v_b(t) + \beta_b}{T_b(t)}} + k_{b,2} e^{(\gamma_b v_b(t) + \delta_b)}, \forall b, t \tag{5.5}$$

such that $k_{b,1}$, $k_{b,2}$, α_b, β_b, γ_b, and δ_b are parameters that depend on circuit topology, size, technology, and design.

Due to the non-linearity of this leakage model, paramterizing $k_{b,1}$, $k_{b,2}$, α_b, β_b, γ_b, and δ_b becomes a challenging task. Additionally, a major limitation of (5.5) is the cyclical relationship with (2.3) in regards to temperature. In order to solve (2.3) and (5.5) directly, numerical solutions for non-linear analysis are necessary therefore making temperature and power prediction computationally inefficient and difficult to use in practice. Alternatively, one may de-couple the thermal dependency by representing leakage power as a piecewise-linear model in temperature and voltage [12]

$$P_{lkg,b}(t) = P_{lkg,b}^0 + G_b^T T_b(t) + k_b^v v_b(t), \forall b, t \tag{5.6}$$

where G_b^T is the temperature coefficient associated with temperature T and k_b^v is the voltage coefficient associated with voltage v. $P_{lkg,b}^0$ represents the leakage power for block b such that $T_b = 0$ and $v_b = 0$ (i.e. the ambient temperature and the minimum voltage). We can therefore rewrite (2.3) as

$$\frac{d\boldsymbol{T}(t)}{dt} = \hat{\boldsymbol{A}}\boldsymbol{T}(t) + \boldsymbol{B}\hat{\boldsymbol{P}}(\boldsymbol{s}, \boldsymbol{v}, \boldsymbol{T}, t) \tag{5.7}$$

where

$$\hat{\boldsymbol{A}} = \boldsymbol{A} + \hat{\boldsymbol{B}}\boldsymbol{G_T}, \tag{5.8}$$

$$\hat{\boldsymbol{P}}(\boldsymbol{s}, \boldsymbol{v}, t) = \boldsymbol{P}(\boldsymbol{s}, \boldsymbol{v}, \boldsymbol{T}, t) - \boldsymbol{G_T}\boldsymbol{T}(t) \tag{5.9}$$

and $\boldsymbol{G_T}$ is a vector comprised of each block's G_b^T value. Thus the cyclical dependency between power and temperature is removed. Evaluation of the temperature and

leakage power can be done efficiently as both are represented using a set of linear equations.

Dynamic Power

Dynamic power captures the active power of a given block. It is well established that dynamic power of traditional CMOS based technology varies quadratically with the operating voltage of the block. Furthermore, since the dynamic power is only consumed while the clock is high, dynamic power varies linearly with the clock frequency. Therefore, dynamic power of block b can be expressed as

$$P_{dyn,b}(t) = a_b(t)s_b(t)v_b^2(t), \forall b, t \tag{5.10}$$

where $s_b(t)$ and $v_b(t)$ are the time varying speed (frequency) and voltage profiles of block b while $a_b(t)$ represents the activity factor of block b at time t. In practice this value changes over time in relation to the workload characteristics. For example, a workload performing a large number of memory operations will exhibit a different power profile than a workload with purely ALU based operations. However, it is challenging to determine the form of $a_b(t)$ in practice, and more-so to predict the value of $a_b(t)$ in the future. Prior literature typically simplifies the issue by assuming a static form such as a constant value ($a_b(t) = A_b, \forall t$) or proportional to the instructions-per-cycle of the block ($a_b(t) = A_b IPC(t)$). However, such a simplistic form does not properly capture the variations between different workloads [48].

To illustrate this deficiency, we executed several workloads on an actual mobile device – the Google Pixel housing the Qualcomm MSM8996 Pro system on chip. The frequency was fixed at 1.2864GHz and the operating temperatures of the cores where verified to be approximately the same (between 40degC and 45degC). We recorded the system current draw at a rate of 5000 samples per second using the Monsoon Power

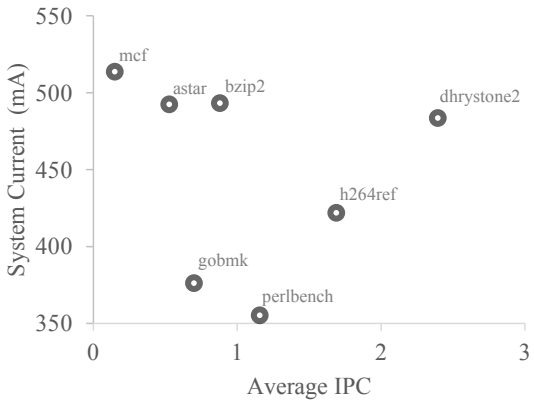


Figure 5.3: Dynamic CPU current draw vs. IPC for selected workloads.

Monitor and averaged the samples over the course of the workloads execution. Additionally, the instructions per cycle were recorded using on-chip performance counters. Figure 5.3 shows the relationship between IPC and current draw. From this we can clearly see that dynamic power is not simply proportional to IPC alone and thus a more sophisticated prediction mechanism is needed to accurately capture the workload's power characteristics. In the following section we present a novel methodology to represent $a(t)$ as a stochastic value based on workload characteristics visible via hardware and software performance counters and on chip-temperature sensors.

Representing P_{sys} with aPCE Models

Thus far, we have provided known mathematical form for system power; however, these formulas where derived using heuristic methods. They also rely on the assumption of known fixed values for many parameters such as the leakage parameters (G_b, k_b, and $P^0_{lkg,b}$) as well as the activity factor (a_b). However, these parameters are often difficult to derive and may vary over the course of the mobile device's lifetime. Therefore, this work proposes the use of arbitrary polynomial chaos expansion to represent system power as a function of observable system state parameters such as architectural events, device temperature, and device voltage/frequency (see Chapter

4). The aPCE representation will be tuned over time to reflect the current workload and system characteristics. This method will be discussed in detail with the controller design in Section 5.3.

5.2.3 Definition of Performance Model

In this work, we assume that performance is some observable and quantifiable quantity dependent on controllable system parameters such as operating frequency. For example, the time needed to render and display a web page or the frames per second decoded by a video player can represent QoS in the context of specific applications. More generally and without the need of application visibility, performance can be related to the IPS of the processor. That is an application or task can be considered as a finite sequence of instructions which must be executed, the speed at which these instructions are executed dictate the level response time of the application and by extension some function of performance. However, it has been shown in prior literature [41] that the flow of execution is halted due to numerous sources – the start time and length of these delays being stochastically random variables. As such, we can utilize aPC methods to model QoS in much the same manner as power. Please refer to Chapter 4 and Appendix A for more information.

5.2.4 Definition of QoS for Mobile Devices

Quality of service is a metric used to relate the user's satisfaction to the system's performance metric. This work assumes that performance and user satisfaction are positively correlated values. However, the issue of evaluating QoS is non-trivial in the mobile domain as performance becomes a non-deterministic value. Thus, QoS must be represented as a probability:

$$\text{QoS}(s, X, \Delta) = Prob\{Perf(s, X) \geq \Delta\} \tag{5.11}$$

where Δ is some performance target set to ensure user satisfaction. Therefore, the best we can do is to ensure that system performance is at least some specified value Δ, with likelihood $Q \in [0, 1]$, where Δ and Q are design parameters that must be choose to ensure desired levels of user satisfaction.

5.3 A Dynamic Energy Management Controller for Mobile Devices

One of the most practical applications of power and performance prediction is in the design of dynamic control algorithms for computer systems. These control algorithms can typically be generalized by a straightforward execution flow in which the algorithm (1) **observes** the current workload and system characteristics, (2) **learns** from this observation by updating the prediction model and reducing its error, and (3) **adapts** to the current system condition by determining the optimal control parameters which optimize some given objective. These three steps are repeated periodically as the system executes its workload. In the following section, we describe an implementation of a QoS-aware DEM controller, starting with the formulation of the optimization problem needed for the adapt stage.

5.3.1 Formulation of the Optimization Problem

The general role of any DEM optimizer is to determine the optimal trajectory of control parameters such that some given objective is optimized while ensuring that all specified constraints are satisfied. In this work, the objective function to be optimized is energy efficiency, which is defined as performance per watt (PPW). Additionally, it is desirable to ensure high levels of user satisfaction. This can be accomplished by constraining the optimization problem such that some minimum level of performance, Δ, is always produced. For example, for the use case of video playback, users expect a minimum playback rate of 30 frames-per-second, anything less would result in sub-

optimal user experience. However, as shown in this work, performance should not be considered as a deterministic value for mobile devices. Therefore, the minimum performance constraint must be reformulated into a quality of service constraint by including a likelihood parameter, Q, such that the probability that performance is at least Δ is greater than Q. Therefore, we propose the following optimization problem to maximize energy efficiency of mobile devices subject to user satisfaction conditions:

$$\max_{\boldsymbol{s}(t)} \quad PPW(\boldsymbol{s}(t), X(t)) = \frac{E\left[\text{Perf}(\boldsymbol{s}(t), X(t))\right]}{E\left[P_{sys}(\boldsymbol{s}(t), X(t))\right]} \tag{5.12}$$

$$\text{such that} \quad \text{Perf}(\boldsymbol{s}(t), X(t)) = \sum_{i=0}^{\infty} c_{Perf,i}(t)\phi_i(\boldsymbol{s}(t), X(t)), \tag{5.13}$$

$$P_{sys}(\boldsymbol{s}(t), \boldsymbol{v}(t), X(t)) = \sum_{i=0}^{\infty} c_{P,i}(t)\phi_i(\boldsymbol{s}(t), \boldsymbol{v}(t), X(t)), \tag{5.14}$$

$$\boldsymbol{s}_{min} \leq \boldsymbol{s}(t) \leq \boldsymbol{s}_{max}, \tag{5.15}$$

$$\frac{d\boldsymbol{T}(t)}{dt} = \boldsymbol{A}\boldsymbol{T}(t) + \boldsymbol{B}E\left(\boldsymbol{P}(\boldsymbol{s}, \boldsymbol{v}, \boldsymbol{T}, t)\right), \tag{5.16}$$

$$\boldsymbol{T}(t) \leq \boldsymbol{T_{max}} \tag{5.17}$$

$$\text{Prob}\left\{Perf(\boldsymbol{s}(t), X(t)) \geq \Delta\right\} > Q \tag{5.18}$$

where $\text{Perf}(\boldsymbol{s}(t), X(t))$ and $P_{sys}(\boldsymbol{s}(t), X(t))$ represents the prediction models of performance and total system power respectively. These are aPCE functions, parameterized by the coefficients, $c_{Perf,i}(t)$ and $c_{Perf,i}(t)$. As described in Section 5.4.3, the optimal basis functions ϕ_i are determined beforehand by randomly sampling $X(t)$ to generate the necessary sample moments.

The objective function (5.12), represents the expected value of the systems energy-efficiency as a function of operating frequency $\boldsymbol{s}(t)$. For any given frequency, this expected value can be determined by evaluating (5.13) and (5.14) with the expected

value of $X(t)$.

Constraint (5.18) provides the quality of service constraint to ensure that system performance is at least some specified value Δ, with likelihood Q.[1] To evaluate this constraint, the distribution of $X(t)$ must be known. In the event that $X(t)$ follows an arbitrary distribution, this distribution can be estimated by randomly sampling $X(t)$ and constructing a histogram to estimate the distribution of $X(t)$. With the distribution of $X(t)$ known, the distribution of $\text{Perf}(\boldsymbol{s}(t), X(t))$ for any given value of $\boldsymbol{s}(t)$ can be estimated using the aPCE model. First, randomly generate N samples of $X(t)$ from its distribution where N is some large number. Next, evaluate $\text{Perf}(\boldsymbol{s}(t), X(t))$ for each of these samples and construct the associated cumulative mass function (CMF) for performance. Let $F_{\boldsymbol{s}(t)}(\delta)$ represent this CMF where δ is some performance value. Constraint (5.18) can therefore be determined by evaluating $1 - F_{\boldsymbol{s}(t)}(\Delta)$.

Furthermore, mobile systems are often thermally constrained due to the lack of active cooling. Thus, we consider the thermal properties of the mobile device by creating a Hotspot model [36] in the form of the state space equation defined in (5.16). The coefficient matrices, $\boldsymbol{A}$ and $\boldsymbol{B}$ are determined beforehand by experimentally stressing the system (see Section 5.3.3 for details of how to experimentally derive $\boldsymbol{A}$ and $\boldsymbol{B}$). Constraint (5.17) ensures that the thermal condition of all components remains under a specified maximum, T_{max} often set at 100°C.

The controller's tunable parameters include per-component frequency, $\boldsymbol{s}(t) \in [\boldsymbol{s}_{min}, \boldsymbol{s}_{max}]$; however, most mobile systems only allow the frequency to be selected from a discrete set of options.. It should be noted that while this work limits the examined constraints to (5.13–5.18); other additional constraints can also be examined

[1]Note that although not considered in this work, it is possible to add a similar constraint for power consumption.

such as deadline and quality management requirements.

5.3.2 Error Correction Mechanisms

The model parameters, $\boldsymbol{c}_{Perf,i}(t)$ and $\boldsymbol{c}_{P,i}(t)$, need not be static and can be **learned**/updated periodically based on the changing operating conditions of the mobile device. Moreover, the necessary order truncation of the polynomial chaos model induces errors and servers only as an approximation of the actual power and performance values. As such, the predictions can be inaccurate under certain untrained scenarios and thus a feedback of past measurements can be used to correct for future predictions. This work proposes the use of an intelligent learning technique based on the Kalman-based Stochastic Gradient Descent algorithm [112] to minimize the impact of model error.[2]

Kalman-based Stochastic Gradient Descent (kSGD) – Kalman-based stochastic gradient descent (kSGD) [112] offers a mechanism for learning model parameters of statistically varying systems. For online learning, kSGD acts as an adaptive filter which determines the model coefficients via a minimization of mean square error of a sequence of data points such that each data-point is weighted according to the variance of the system. The advantage of kGSD over traditional filtering techniques such as recursive least squares and traditional gradient descent is the fast convergence of the model parameters along with its robustness – it is not sensitive to the condition number of the problem.

Algorithm 2 summarizes one iteration of the kSGD procedure which is performed independently for the two models, Perf$(\cdot)$ and $P_{sys}(\cdot)$. Let $\boldsymbol{c}(k)$ represent the estimated model parameters at time k (either for the performance or power model). To improve the estimate of $\boldsymbol{c}$, the kSGD algorithm determines two factors: the direction

[2]Although any online technique to reduce model error could be used.

Algorithm 2: kSGD($\boldsymbol{c}(k), M(k), \boldsymbol{\phi}, X(k), y(k)$)

Input: $\boldsymbol{c}(k)$ – current estimate of model parameters;

$M(k)$ – current estimate of the input covariance matrix;

$\boldsymbol{\phi}$ – the set of aPCE basis functions;

$X(k)$ – measured values of model input;

$y(k)$ – measured values of model output;

1. Compute: $v = M(k)\boldsymbol{\phi}(X(k))$;
2. Compute: $s = \gamma^2 + \boldsymbol{\phi}(X(k))^T v$;
3. Update: $\boldsymbol{c}(k+1) = \boldsymbol{c}(k) + v(y - \boldsymbol{\phi}(X(k))^T \boldsymbol{c}(k))/s$;
4. Update: $M(k+1) = (I - \frac{v\boldsymbol{\phi}(X(k))}{s})M(k)$;

of the improvement and the magnitude of this improvement. First, as shown in Line 1 of Algorithm 2, a vector v is calculated based on the estimated covariance matrix, $M(k)$, along with the basis functions, $\boldsymbol{\phi}$, evaluated with a observed input vector $X(k)$. This is similar to the a-posteriori update stage of Kalman-filter which minimizes the effects of model inaccuracies via the inner-product between the sample data point and its covariance matrix. Next, a scaling factor, s, is determined in Line 2. This factor is determined using a hyperparameter, γ^2, and is designed to mitigate issues involving the condition number of the problem. Finally, the model update occurs in Lines 3 and 4. Much like other Gauss-Newton methods, the direction of the model improvement is determined by the gradient of the error function (minimize the L2-norm of model error). This process is repeated at every point in time in which a new data point is available.

5.3.3 Power Prediction via Thermal Workload Characterization

To learn the performance and power prediction coefficients in (5.13) and (5.14), a time trace of the workload's power and performance characteristics must be either derived or measured. This is a non-trivial task due to limitations on-chip power sensors. Most modern mobile SoCs lack the fine grain power monitors needed to view the power of each processing element which can then be used to calculate the platform power consumption. Instead, these devices have at best a single monitor to measure the battery current. Instead, a collection of thermal sensors at different locations within the chip are offered. For the Qualcomm MSM8996 pro, over 25 on-chip sensors are available including temperature sensors for the per-core alu, the per-core L1 cache, the per-cluster L2 cache, the shared L3 cache, the GPU, and the package. Instead of reading a power sensor directly, we propose to utilize the knowledge of the mobile device's thermal characteristics to infer the per-component power consumption of the device. In other words, the collection of thermal sensors will act as a surrogate for the power sensors needed to characterize Equation (5.14).

Prior work has demonstrated that the temperature sensors can be used to obtain information about the power dissipation at various locations in the processor. One such work proposes to use blind identification techniques [113]. Alternatively, the thermal model can be constructed via experimental evaluation and model fitting. This procedure is outlined below.

1. Establish a baseline condition such that no workload is running and all power sources are either disabled or set to the lowest power settings (e.g. minimum frequency and voltage). Record the steady state temperature vector and total power consumption.

2. For each processing element/power source one at a time, assign a workload to

the element and vary the power setting over time. Record the time trace of temperature and total power. Assume that the difference between the total power and the baseline power is the power dissipated by the specific element under stress.

3. Use state space fitting methods such as LMS to determine $\boldsymbol{A}$ and $\boldsymbol{B}$.

With $\boldsymbol{A}$ and $\boldsymbol{B}$ determined offline, we can now estimate the per-component power consumption, $\hat{\boldsymbol{P}}(t)$, using thermal readings only.

$$\hat{\boldsymbol{P}}(\boldsymbol{s}, \boldsymbol{v}, \boldsymbol{T}, t) = \boldsymbol{B}^{-1}\left(\frac{d\boldsymbol{T}(t)}{dt} - \boldsymbol{A}\right) \tag{5.19}$$

$$\frac{d\boldsymbol{T}(t)}{dt} \approx \frac{\boldsymbol{T}(t) - \boldsymbol{T}(t-\Delta)}{t_s} \tag{5.20}$$

where, t_s is the time between reading two concurrent thermal readings are obtained. It is then assumed that P_{sys} is approximately the sum of the components of $\hat{\boldsymbol{P}}$.

5.3.4 Controller Implementation

A summary of the procedure to maximize energy efficiency using the proposed closed loop controller is described in Algorithm 3 and implemented on the experimental platform using C. The controller design is simplistic enough to be portable to most mobile platforms. As we will show, it is also robust enough to achieve significant energy-efficiency improvements. The inputs of this controller are the aPCE performance and power models, specifically, the set of basis functions, $\boldsymbol{\phi}$, as well as the initial estimates of the model coefficients, $\boldsymbol{c}_P Perf(0)$ and $\boldsymbol{c}_P P(0)$. Additionally, the state-space matrices, $\boldsymbol{A}$ and $\boldsymbol{B}$, for the thermal model defined in (5.19) are needed.

The flow of the controller follows a simple procedure of **observe**, **learn**, and **adapt**.

Observe: First, the current values of performance, $y(k)$, temperature, $T(k)$, and sources variation, $X(k)$ are measured using on-chip sensors (Line 2 of Algorithm 3).

Following this, the power consumption during the previous update window, $P(k)$, is estimated using (5.19–5.20) as described in Line 3.

Learn: With the measured system information available, the performance (Line 4) and power (Line 5) models are updated using the kSGD method described in Section 5.3.2.

Adapt: Finally, in Line 6, the optimization problem detailed in (5.12–5.17) is solved using convex optimization techniques. This optimization represents adaptation portion of the controller, leveraging the estimated power and performance prediction models to determine the resulting system characteristics at various voltage and frequency settings, $\boldsymbol{v}$ and $\boldsymbol{s}$.

The controller, then waits a specified amount of time before making the next iteration. In order to be compatible with most mobile operating systems, it is assumed that control decisions occur at a fixed frequency where the time between updates is $t_s \in [10\ ms, 100\ ms]$. For this work, a value of 100 ms is used.

5.4 Experimental Methodology

In this section, a description of the experimental setup is provided. An essential condition for the experimental methodology developed for this work is to accurately represent real world use of smartphone devices. A commercially available smartphone was selected as the experimental platform along with realistic mobile workloads.

5.4.1 Real-Device Experimental Platform

The experiments were conducted on a Google Pixel Smartphone [99] housing Qualcomm's MSM8996pro chipset [100]. The phone features on-chip performance monitoring units (PMU). The six PMU's can be programmed to observe various hardware events such as L2 cache misses, branch mispredictions, and main memory page

Algorithm 3: DEM-Controller

Input: t_s – controller update period;

$\boldsymbol{c}_{Perf}(0), \phi, M_{Perf}(0)$ – initial estimate of performance model parameters and covariance matrix;

$\boldsymbol{c}_P(0), \phi, M_P(0)$ – initial estimate of power model parameters and covariance matrix;

$\boldsymbol{A}$, $\boldsymbol{B}$ – coefficients to the power-temperature state space model (5.16);

1. Initialize: $k = 0$;

while *Device is running* **do**

2. Read: $y(k)$, $T(k)$, and $(\boldsymbol{X}(k))$;
3. Calculate: $\boldsymbol{P}(k)$ using (5.19–5.20);
4. Update: $\boldsymbol{c}_{perf}(k+1) = \text{kSGD}(\boldsymbol{c}_{perf}(k), M_{perf}(k), \boldsymbol{\phi}, X(k), y(k))$ via Algorithm 2;
5. Update: $\boldsymbol{c}_P(k+1) = \text{kSGD}(\boldsymbol{c}_P(k), M_P(k), \boldsymbol{\phi}, X(k), y(k))$ via Algorithm 2;
6. Solve: $\boldsymbol{s}(k)$ and $\boldsymbol{v}(k)$ using convex or non-linear methods on (5.12–5.17);
7. Increment: $k = k + 1$;
8. Wait: t_s time.

faults. We sample these PMU's at a rate of 10 samples per second. Additionally, The MSM8996pro offers a heterogeneous architecture with two CPU cores tuned for high performance and two CPU cores tuned for low power. The two cores within a cluster share a common L2 cache and both clusters share a common L3 cache. Unless otherwise noted, all experiments were conducted on the performance oriented cluster since at lower operating frequency settings, performance and power oriented clusters exhibit similar power and performance characteristics (i.e. the operational character-

Table 5.1: Parameters for the Pixel Smartphone and Qualcomm MSM8996pro Chip-Set.

	Low Power Cluster	High Performance Cluster
Number of Cores	2	2
Architecture	Kyro	Kyro
Instruction Set	ARM v8a	ARM v8a
Frequency	0.3-1.59GHz	0.3-2.15GHz
L1 Cache Size	32KB I & 32KB D	32KB I & 32KB D
L2 Cache Size	512KB	1.5MB
L3 Cache Size	4MB	

istics of the power oriented cluster is a subset of the performance oriented cluster). Table 5.1 lists the important architectural specification of the device.

A common limitation of commercially available mobile devices is the lack of real time power measuring sensors and should they exist, typically provide coarse measurements at a very low sample rate. To address these issues, the device's battery was removed and replaced with a constant voltage source via the Monsoon power monitoring unit [101]. The unit can measure the total system's voltage and current at a rate of 5000 samples per second.

The experimental platform runs a rooted Android 7.2 OS. The applications of interest were cross-compiled on a host machine with the latest ARM-Android NDK toolchains [70]. The binary is pushed to the device and is launched from the host machine via a wireless connection using the android debug bridge terminal.

5.4.2 Benchmark Applications

In order to provide a good breadth of mobile applications, a number of applications were evaluated which (1) have been utilized in prior works for predicting the performance of mobile and interactive workloads [81, 41, 18, 79, 65] and (2) represent workloads which may be present on current or future smartphones.

Interactive Workloads for Application Specific Performance Prediction

User satisfaction is the driving force of how successful a mobile application will be; however, to estimate an application's performance metric, additional information or hints can be propagated by the application itself to determine data input complexity. Below are several interactive workloads considered in this work along with the additional information passed to the prediction framework.

Web Browser and HTML Viewer – The *web browser* serves as one of the most utilized applications on modern smartphones. A user's satisfaction with a web page is directly related to the time needed to load and render the page [1] and therefore, performance denotes the webpage load time. We utilize the Google Webview [102] framework to load various webpages. These webpages are selected from the 100 most viewed webpages according to the Alexia service [103]. The composition and complexity of a web page is determined by its DOM tree – a structure whose nodes are labeled by a unique tag describing the node's function (e.g. java scripts, images, links, etc.) along with multiple attribute fields providing additional information for the node (e.g. the dimensions of an image to display). This information can be determined quickly after the webpage request is made and is provided to the prediction framework as a means to capture per-webpage complexity [18].

Image Recognition and Similarity Search – *Ferret* [114], is an image simi-

larity ranking algorithm from the PARSEC benchmark suite [68]. The algorithm is used for content-based similarity search of feature-rich data, such as images or videos, and is an important building block for many image recognition and augmented reality Apps running on modern mobile devices. The performance metric of this algorithm is measured by the response time from when the image is given to the algorithm to when the algorithm produces the similarity results at the output. We observe that Ferret's algorithmic complexity is related to three factors, the database size, the image size, and the number of unique fragments in the image. However, since we assume that Ferret is being applied to an augmented reality application, it is assumed that all pictures have the same image size and the database size is constant; therefore, input complexity is limited to the number of fragments per image.

Interactive Gaming – On modern mobile processors, gaming applications are common place. In this work, we examine the CPU portion of these gaming applications which is responsible for responding to user's request and interactions. The first is *2048* [105], a puzzle game with extremely limited computational overhead and who's response time is largely dictated by IO requests. The second is *GNU Go* [106] which incorporates computationally heavy search techniques such as breadth first search and decision trees. The program flow for both of these applications is to wait for a user's input, respond to this input, and finally return control back to the user, repeating the process. The performance metric of this application is the response time from when the user inputs the request to when the application finishes the serving the request. In the case of *GNU Go* this requires processing the user's move, determining the application's move, and then render this information to the smartphone's display. In contrast, *2048* simply needs to process the user's input.

Workloads for General IPS Performance Prediction

While additional effort by the application designer provides a more direct measure to improve user satisfaction it is also preferable to increase user satisfaction to all types of applications–even those which are not specifically designed with QoS-aware energy management in mind. To accomplish this we demonstrate that our performance prediction framework can be used to estimate the instructions-per-second of a processor with only information from the on-chip performance monitoring units, requiring no interaction with the foreground application. We examine the following mobile workloads.

Security/Communication Benchmarks – A common task in smartphones is the transmission of data via a wireless/radio network. This data is encrypted to mitigate packet snooping. We examine two common algorithms used in secured data transmission: *sha* and *aes* [104]. These workloads also serve as a comparison point with prior works in mobile performance monitoring [81]. Finally, we examine an *FTP* client used to transfer files between the host machine and the mobile client. This workload utilizes a wireless network as the communication medium between the devices.

Office Benchmarks – Due to limited storage and expensive data transmission it is common for applications to compress data. We examine the *bzip2* [107] benchmark under several different inputs such as text files, images, videos, programs, and file systems. Additionally, we examine an implementation of the min-cost flow algorithm, *mcf*, as applied to resource scheduling.

Video Decoding and Playback – The final application of interest is *h264ref*, an algorithm allowing for the encoding or decoding of video files to the h264 specifications.

Table 5.2: Input Variables for the Power and Performance Prediction

System Events	
x_1	L1 data cache misses per cycle
x_2	Branch misses per cycle
x_3	Shared L2 cache misses per cycle
x_4	Bus accesses per cycle
x_5	Page faults per cycle
x_6	CPU Temperature
Controls	
x_7	Power Oriented CPU cluster frequency
x_8	Performance Oriented CPU cluster frequency
x_9	GPU frequency
x_{10}	Memory bus frequency

5.4.3 Prediction Models for Mobile Workloads

To demonstrate the benefit of utilizing the proposed aPCE framework, two alternative prediction schemes are examined: *regression* and *last observed*. These two schemes are commonly implemented in literature and in industry; and serve as a baseline for predictive model analysis. In the following section, the implementation of all three prediction models is discussed.

aPCE Prediction Framework

As introduced in Chapter 4 and more thoroughly in Appendix A, the proposed aPCE prediction models represent the response variable (i.e. performance or power) as the weighted sum of a collection of orthogonal basis functions which take, as inputs, a

collection of random variables denoted $X = \{x_1, x_2, \ldots\}$. This has two major benefits over alternative techniques such as regression modeling which directly uses the value of X.

First is the orthogonality of the basis function. While not covered in this paper, the orthogonality of the basis functions allows for incrementally modifying the model complexity over time without the need to re-evaluate all coefficients [115]. For example, consider the scenario in which a second order aPCE model is constructed. If a designer feels that the model error is inadequate, they may increase the order of the model or add additional model parameters. Without orthogonality, this would require completely re-evaluating the model coefficients including those previously derived from the second order model. However, the orthogonality principle allows the new coefficients to be evaluated, independently from the prior ones. This can greatly reduce model training overhead and allow for dynamic model construction.

Second, aPCE and PCE in general decompose the response variable into a set of optimal[3] orthogonal polynomials [92]. In other words, there exists no better representation of the response variable as a function of the given input parameters X. This ultimately results in faster convergence rates over any other modeling technique when determining model parameters thus requiring less experimental design work.

The first step to developing the aPCE model is identifying the sources of variation within the system. In this work, the effects of architectural events are examined. While it is theoretically possible to model all architectural events, in practice this is not feasible. An excessive amount of model parameters may lead to over-determined models. Additionally, the specification of the system may limit the amount of variables which can be observed at any given time. In the case of the Google Pixel smartphone, only 6 architectural events may be observed at any given time. Thus,

[3] in the mean-square sense

we limit our input to the most highly correlated architectural events as described in Table 5.2

With the model input defined, the next step is to construct the optimal orthogonal basis functions. To do so, sample data must be available for the sources of variation, X, as well as the corresponding system response, y. As an example, this work constructs a second order aPCE model. As with regression modeling, this involves determining the 0, 1^{st}, and 2^{nd} order terms for each input variable x_i as well as the cross-terms for each combination of $(x_i, x_j), i \neq j$. We begin by constructing the univariate basis functions.

$$\begin{aligned}
&\text{order 0:} \quad P^{(0)}(x_i) = 1 \\
&\text{order 1:} \quad P^{(1)}(x_i) = x_i - \mu_1(x_i) \\
&\text{order 2:} \quad P^{(2)}(x_i) = x_i^2 + \frac{\mu_3(x_i) - \mu_1(x_i)\mu_2(x_i)}{\mu_1^2(x_i) - \mu_2(x_i)} x_i \\
&\qquad\qquad\qquad + \frac{\mu_2(x_i)^2 - \mu_1(x_i)\mu_3(x_i)}{\mu_1(x_i)^2 - \mu_2(x_i)}
\end{aligned}$$

where $\mu_j(x_i)$ is the j^{th} moment of input x_i. We are able to calculate the sample moments of x_i by taking a random sampling of N data points of each performance counter.

$$\mu_k(x_j) \approx \hat{\mu}_k(x_j) = \frac{1}{N} \sum_{i=1}^{N} x_j(i)^k, \quad \forall k = 1 \ldots K, \tag{5.21}$$

where $x_i(k)$ is the k^{th} realization of the random variable, x_i. Finally, we must determine the basis functions for the cross-terms $(x_i, x_j), i \neq j$. As long as x_i and x_j are independent, these can be easily found by taking the product of the two univariate basis functions [94]. That is,

$$P^{(2)}(x_i, x_j) = P^{(1)}(x_i) P^{(1)}(x_j), \forall i \neq j.$$

With the polynomial basis functions defined, it is now possible to define the aPCE representation of the system response, y. For example a second order model for two

input terms, x_1 and x_2 would be the following:

$$\begin{aligned} y =& c_0 + \\ & c_1 P^{(1)}(x_1) + c_2 P^{(1)}(x_2) \\ & + c_3 P^{(2)}(x_1) + c_4 P^{(2)}(x_2) + c_5 P^{(1)}(x_1) P^{(1)}(x_2) \end{aligned}$$

where, c_i are the model coefficients which must be determined empirically.

Regression-based Prediction

Regression based modeling is among the most widely used modeling frameworks in literature [77, 83, 84, 85, 86]. As with the aPCE method, a second-order model is constructed for each of the ten workloads using the variables described in Table 5.2. The difference, however, is that the variables are directly applied to the model rather than being transformed via a set of basis functions. Equivalently, the regression model requires that all coefficients, c_0, c_is and c_js be determined for the second order model:

$$y = c_0 + \sum_{i=1}^{10} c_i x_i + \sum_{i=1}^{10} \sum_{j=1}^{10} c_{i,j} x_i x_j.$$

In comparison to aPCE, the regression model will demonstrate the effect of not considering X as a non-deterministic value.

Last Observed Prediction

The simplest prediction method examined in this work is Last Observed Value. Thanks to its extremely low overhead, last observed prediction models are often utilized by scheduling algorithms which must have negligible impact to system performance [78]. The last observed value assumes that an ordered sequence of realizations of the response variable, y, is available. Let $y(k)$ represent the k^{th} element of

sequence. Last observed value then states that $y(k+1) = y(k)$. As such no consideration to the causes of system variation is given, rather the assumption is that system dynamics are slowly moving. Last observed value represents a good reference framework to evaluate the variability of system dynamics over time.

5.5 Prediction Accuracy of aPCE models for Application QoS and Power

Most modeling techniques currently utilized in the control of mobile systems ignore the two basic characteristics of mobile applications considered in this paper – namely, (1) the stochastic nature of an applications performance and power, and (2) the arbitrary nature of their distributions. For example, many dynamic energy management algorithms simply assume that future power and performance values will be approximately equal to the last observed value, thus ignoring the non-deterministic nature of the system [78]. Additionally, more sophisticated modeling techniques such as regression modeling are only optimal given the response variable follows a known distribution such as Guassian [116].

In contrast, the aPCE model developed in this paper is data-driven, and does not require any assumptions on the distribution of the random variables involved in the model. This raises the question: how much does accounting for this arbitrary non-determinism improve model accuracy?

To answer this question, the ten mobile workloads were executed 100 times at each frequency setting. An aPCE, regression and last observed prediction model was constructed for each application. Half of the data set was used to train the associated aPCE and regression models while the other half was used for testing. Figures 5.4 and 5.5 show the test data set's error in predicting performance and power respectively, for our proposed prediction scheme along with two other commonly used prediction frameworks. For all ten workloads aPCE outperforms all other techniques

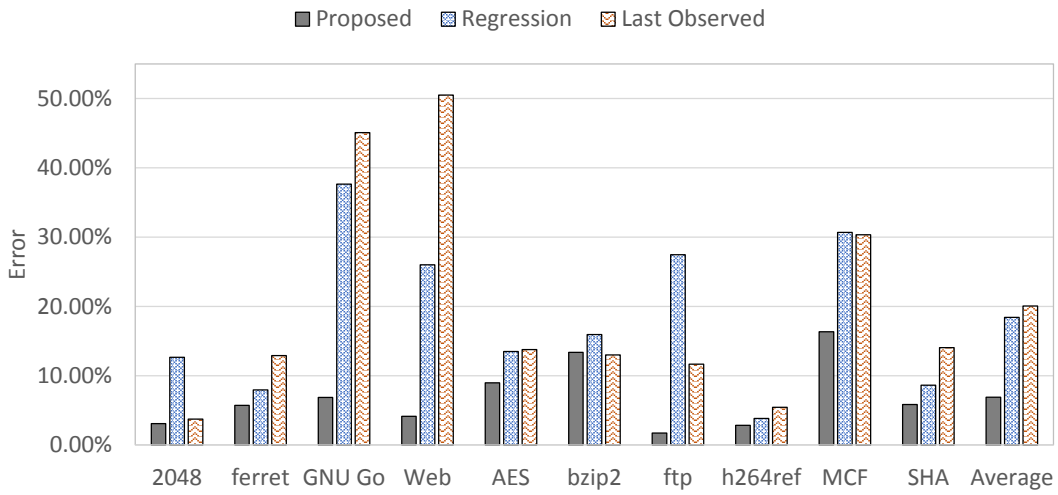


Figure 5.4: The performance prediction error of various modeling frameworks.

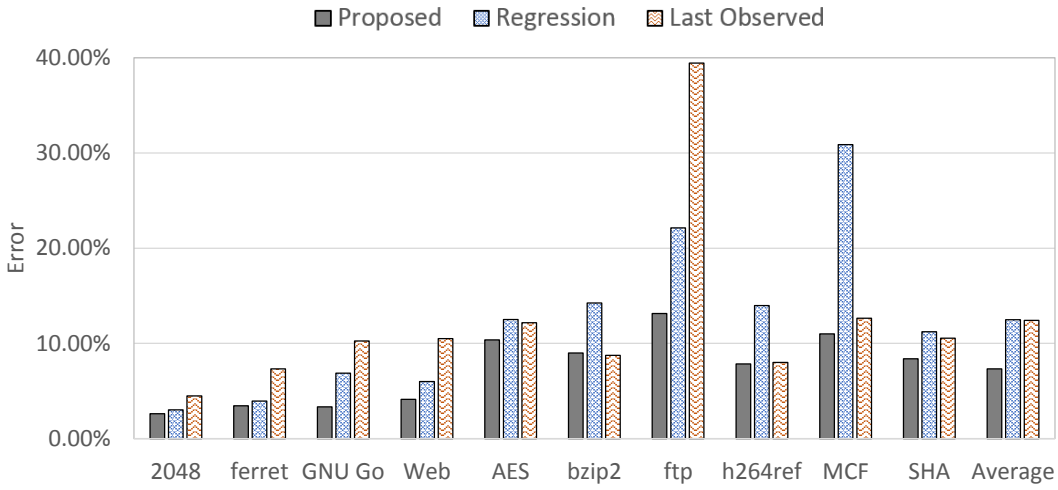


Figure 5.5: The power prediction error of various modeling frameworks.

for both performance and power modeling. However, for workloads with stable performance metrics, such as *2048*, *aes*, *bzip2*, *h264ref*, we can see that the benefits are minimal over the much more simplistic, last observed value method. Likewise, the benefit for aPCE power modeling is most present in interactive workloads and workloads that utilize numerous system resources such as *ftp*.

Overall, aPCE modeling produces a performance prediction error of 6.88% and a power prediction error of 7.33%, reducing performance prediction error by **2.67X**

over regression and **2.91X** over the simplistic last observed value. Additionally, the power prediction error is reduced by **1.70X** over both regression and last observed value. This indicates that it is beneficial to consider system non-determinism when constructing prediction models. However, this leads to two very important questions. First, are these models light-weight such that the computational overhead is negligible? Low computational complexity is a major reason why prediction methods such as last observed are still utilized rather than more accurate models. Second, exactly how much energy efficiency gains can achieved by this increased in accuracy? These questions are explored in the following section.

5.6 Model Viability for DEM of Mobile Systems

5.6.1 Modeling Overhead

In the prior section, it was demonstrated that modeling frameworks which consider system non-determinism of arbitrary distributions provide a significant reduction in prediction error. However, the viability of utilizing such a model is also determined by the overhead needed to develop and evaluate the model.

Figure 5.6 illustrates the measured relationship between computational complexity (in cycles) and the number of model terms along with the order of the model. Predictably, much like the case of regression techniques, the order of the model determines computational complexity. For the second order model with 10 variables as used in prior analysis, a computational cost of approximately 475 cycles are needed for each evaluation. When utilized in a DEM controller with periodic updates every 100ms, this equates to less than 0.0016% overhead per evaluation[4].

[4]Note that the number of evaluations is largely dependent on the design and implementation of the DEM controller. We therefore, leave this overhead metric as a function of the number of evaluations

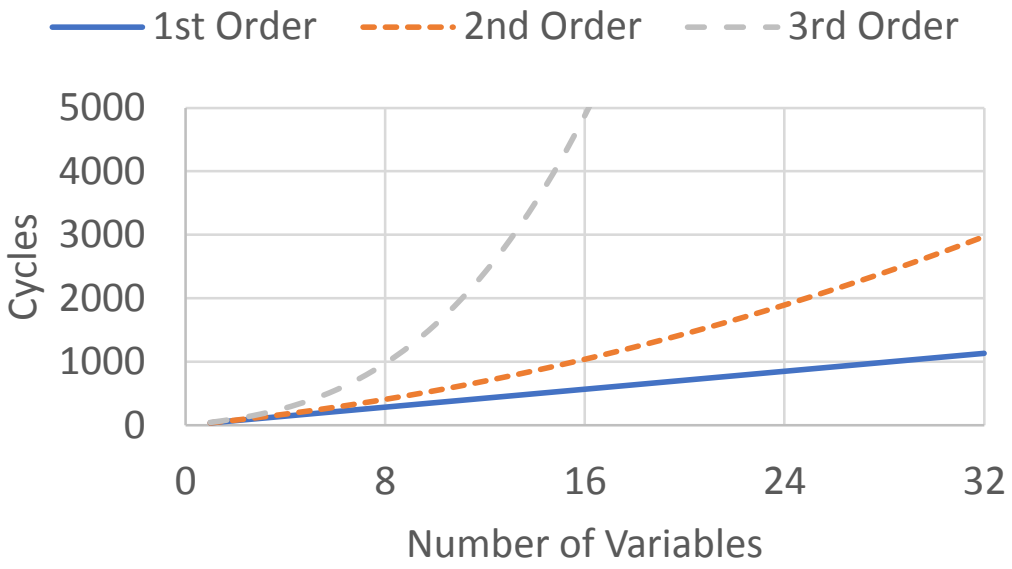


Figure 5.6: The number of cycles needed to evaluate aPCE models of various orders and input size.

Additionally, Figure 5.7 shows the measured energy overhead needed to evaluate the model. This indicates that 9.73×10^{-7} mAh of energy is needed for each evaluation of the model. Such an energy overhead is negligible in comparison to the idle current draw ($\tilde{1}$00 to 300mAh) of smartphones such as the Google Pixel smartphone.

5.6.2 Sensitivity Analysis of Model Error on Energy Efficiency Curve

Another factor when applying aPCE modeling to DEM control techniques is the sensitivity of the control parameters, i.e. frequency, on the objective function (energy efficiency). It is reasonable to expect that model error will result in a sub-optimal frequency selection leading to a loss of energy-efficiency. Furthermore, one would expect that the larger modeling errors of the regression and last observed prediction techniques should result in a greater loss of energy-efficiency when compared to the proposed aPCE framework.

This loss of potential energy efficiency can be represented mathematically. That

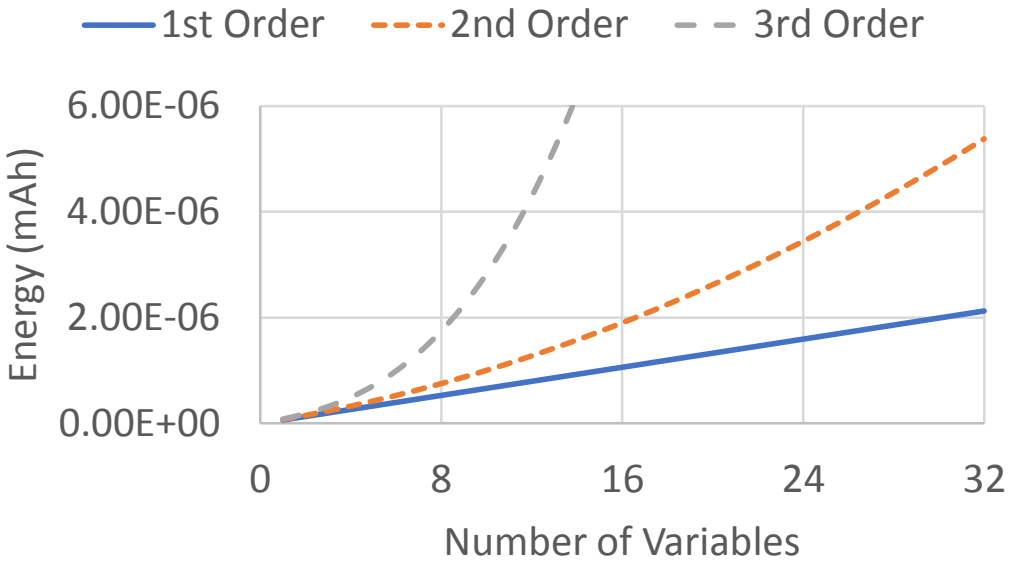


Figure 5.7: The energy needed to evaluate aPCE models of various orders and input size.

is, consider the true (i.e. measured) energy efficiency curves ($E(s)$) to be equal to $E(s) = Q(s)/P(s)$ where $Q(s)$ and P(s) are the application's performance and power as functions of operating frequency. The optimal frequency s^* can therefore be determined by evaluating $s^* = \text{argmax}_s\, E(s)$. However, in practice, the true representations of $Q(s)$ and $P(s)$ are not available, thus prediction models, $\hat{Q_j}(s)$ and $\hat{P_j}(s)$, are used as estimates. That is, the energy efficiency curve given prediction method j is estimated as $\hat{E}_j(s) = \hat{Q_j}(s)/\hat{P_j}(s)$ which suggests an optimal operating frequency of $\hat{s_j^*} = \text{argmax}\, \hat{E}_j(s)$. Depending on the magnitude of the discrepancy between the true and the estimated functions, $\hat{s_j^*}$ may vary greatly from the actual operating frequency thus leading to large losses of potential energy-efficiency improvements. This loss of energy-efficiency due to the prediction model can be modeled as

$$\text{Energy-Efficiency Loss} = E(s^*) - E(\hat{s_j^*}) \tag{5.22}$$

For example, consider Figure 5.8 which shows part of the energy efficiency curve

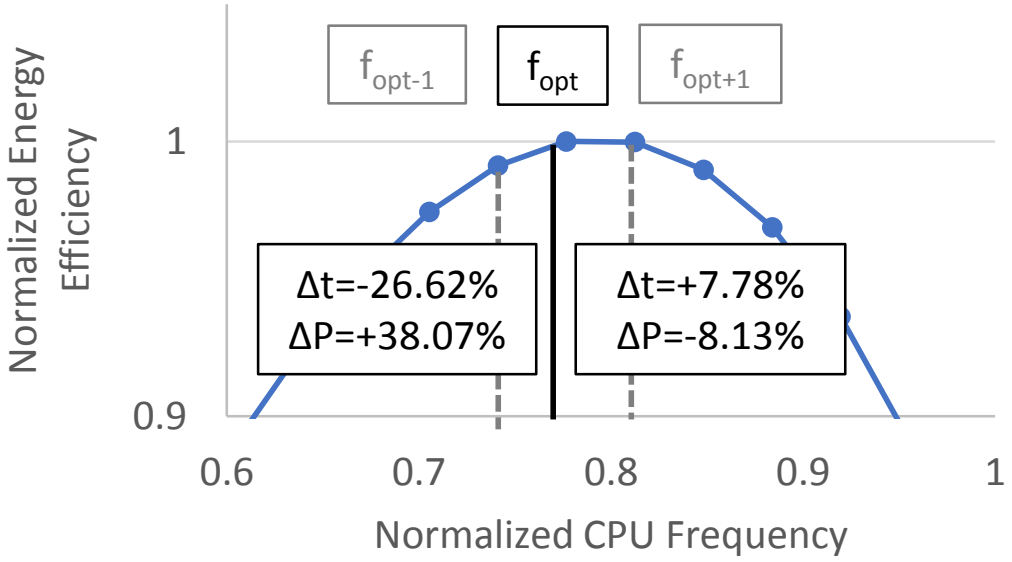


Figure 5.8: An illustration of model sensitivity on the energy efficiency of the FTP application.

for the FTP workload. In this case either a model error of +7.78% in performance prediction or -8.13% in power prediction can result in shifting the optimal frequency selection one setting higher. However, the net result of this is only a 0.5% loss in energy efficiency.

Figure 5.9 shows the average energy efficiency reduction when utilizing each of the three prediction schemes discussed in Section 5.5. As expected the aPCE framework results in the least wasted energy-efficiency (1.26% on average). This is a **3X** less wasted energy efficiency over regression based modeling techniques and an **8X** improvement over last observed value. Ultimately, this indicates that although extremely simple, last observed value is insufficient when applied to DEM controllers and a more sophisticated technique, such as that proposed in this paper, is needed.

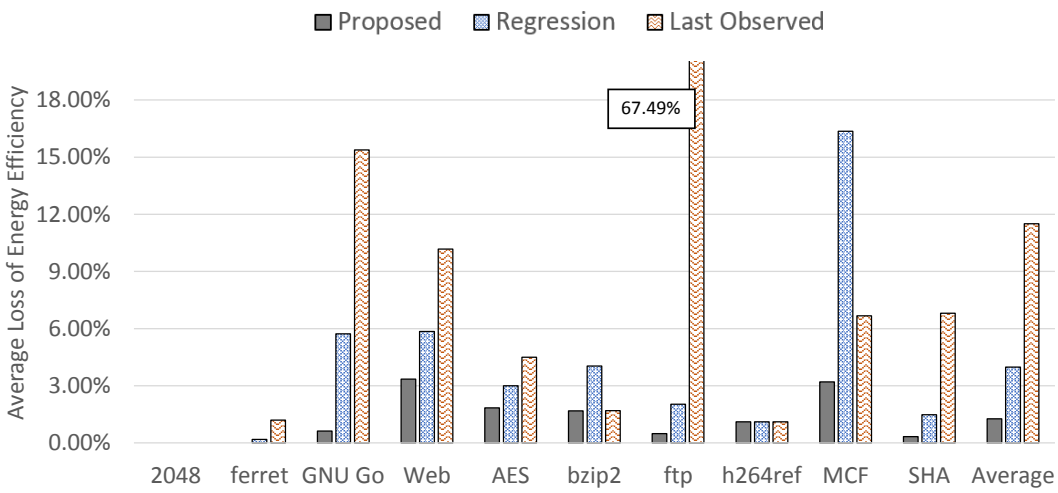


Figure 5.9: The effect of modeling error on determining the optimal energy efficiency.

5.7 Energy Efficiency Improvements from aPCE-based DEM

To validate the effectiveness of our controller we compare it against three DEM policies commonly found on modern smartphones: *Performance*, *PowerSave*, and *Interactive* [78]. As the name implies, the *Performance* policy attempts to achieve the maximum performance by always running at the maximum frequency. The *Powersave* policy aims to minimize the overall energy consumption by running the processor at the minimum frequency. The *Interactive* policy reacts to the workload activity and attempts to maximize performance, while minimizing energy consumption; aggressively increasing the cpu frequency as the workload size increases. A major limitation of these three DEM policies is the lack of consideration to performance requirements (i.e. they do not consider constraint (5.18)). To maintain a fair comparison between all four polices, the proposed controller will also not consider this constraint; however, in the next section an analysis of the effect of the QoS constraint on energy efficiency will be considered.

Figure 5.10 specifies the improved energy efficiency of the proposed controller over the existing governors. Several conclusions can be drawn from these results. First,

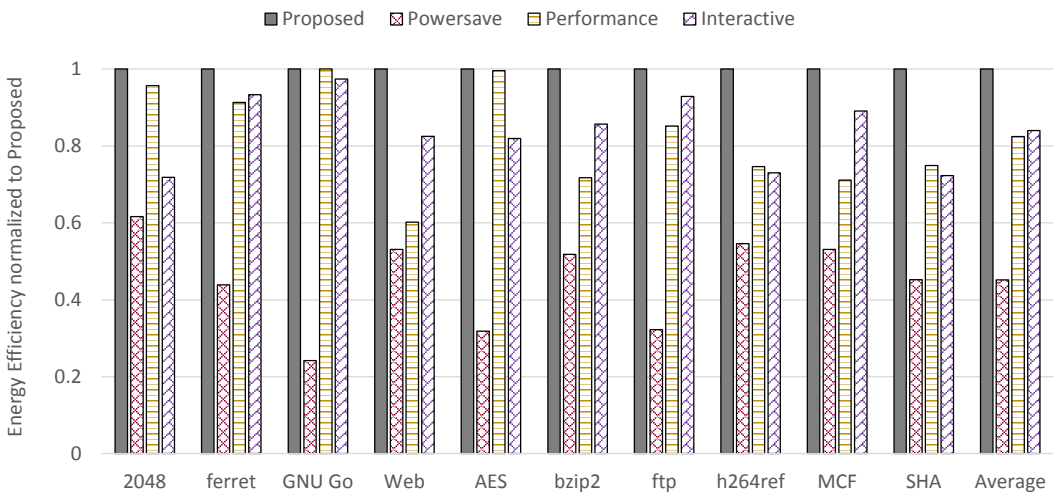


Figure 5.10: Comparison of the energy efficiency of the proposed controller against the *Performance*, *Interactive*, and *Powersave* frequency governors. The proposed controller exhibits negligible overhead as observable with the *GNU Go* and *aes* which have an optimal control policy is equivalent to the *Performance* governor.

the proposed controller does better than *Powersave* in all metrics – energy consumption, energy efficiency, and application performance. Additionally, on average, the proposed controller produces substantially higher energy efficiency than all existing DEM polices available on the smartphone. These results are thanks to the existence of an unique optimal operating frequency, which needs not be the maximum or minimum frequency.

Additionally, this optimal frequency changes based not only on the application, but also the phase of the application, co-scheduled tasks, and so-on. While the *Interactive* governor does assume the existence of this unique optimal frequency, it lacks the increased knowledge of process variability present in the proposed controller and only bases workload intensity on the number of tasks available. This typically leads to selecting a higher frequency than the optimal one. In contrast, the aPCE models utilized by the proposed controller allows to optimally track the frequency which can change based on the current workload and system conditions.

Furthermore, observing the *aes* and *Gnu Go* workloads, the optimal control strat-

egy is approximately equal to the performance governor. The proposed controller determines a similar control strategy and furthermore experiences negligible loss of energy efficiency due to the light overhead.

Ultimately, these energy improvements prolong the battery life of the smartphone. The aPCE framework improves energy efficiency by an average of 19% over the *Interactive* frequency governor. Additionally, the proposed framework has the benefit of providing a methodology to not only evaluate the expected performance and power conditions of mobile workloads, it also provides a mechanism to determine higher order statistical characteristics – a necessary and vital trait for real time workloads subject to user satisfaction requirements.

5.8 Ensuring User Satisfaction with Image Similarity Search

In the previous section, the QoS constraint (5.18) was ignored to maintain a fair comparison between the four frequency governors. However, generally it is desirable to include such a constraint to ensure high user satisfaction with the mobile device. In this section we explore how incorporating this constraint effects the mobile device's energy efficiency. Due to space limitations, we will a single workload, *ferret*, however, such analysis can be conducted for any workload in which performance has a measurable impact on user satisfaction. Ferret is an image similarity search algorithm which works in a manner such that an image is periodically provided by some source (e.g. camera) and given to the image search algorithm for processing against a local image database. The output of *ferret* is a ranking of the most similar images to the input image.

To ensure high user satisfaction, ferret must process these images faster than some given rate Δ. As Δ increases, it is logical to assume that the system will require faster processing frequencies in order to meet the QoS constraint; however,

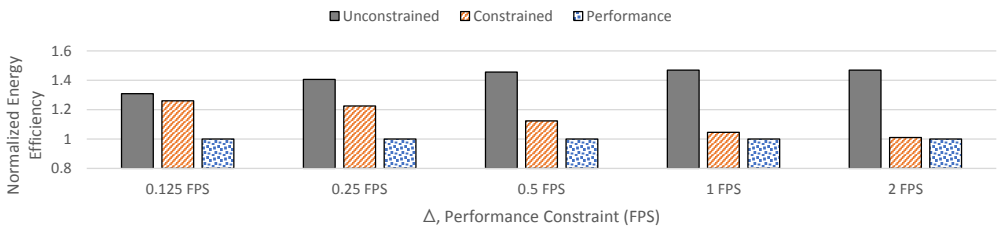


Figure 5.11: The energy efficiency values of the QoS constrained, unconstrained and performance polices. All values normalized with respect to Performance Governor. Higher is better.

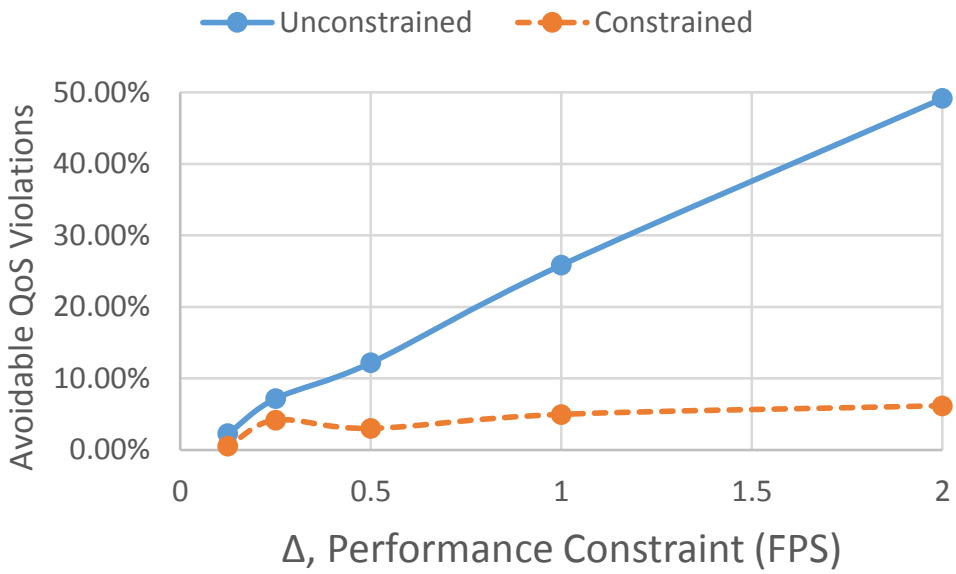


Figure 5.12: The number of QoS deadline violations for the QoS unconstrained and constrained policies as a function of Δ. Lower is better.

this comes at the cost of reduced energy-efficiency. Figure 5.11 shows the energy consumption for three control polices: *unconstrained*, *constrained*, and *performance*. The *unconstrained* policy represents the case in which the optimization problem in Section 5.3.1 is solved without QoS constraint (5.18) while the *constrained* policy includes this constraint with a Q value of 0.95. This Q value is selected to ensure that the QoS constraint is upheld the majority of the time, thus it is unlikely that the user will experience poor operating conditions. However, this value could be relaxed in order to gain energy efficiency. As expected, decreasing Δ results in reduced difference between the unconstrained and constrained policies. Therefore, it becomes

upto the system and application designers to determine the best value of Δ to balance the system's performance with its energy efficiency.

Additionally, it is important to evaluate how well the controller is able to uphold the QoS constraint. In other words, what is the number of avoidable QoS violations for each control policy? By definition, the Performance governor should never violate the QoS constraint if there exists a feasible solution. Additionally, the aPCE framework was accurate such that the constrained policy resulted in less than 5% QoS violations, within the tolerance specified by Q. In contrast, the unconstrained policy exhibits a large amount of violations. Figure 5.12 shows this relationship. For high values of Δ, excessive QoS violations will occur thus resulting in a poor user experience up to 50% of the time. This demonstrates the importance of considering QoS constraints in the optimization problem in order to maintain high user satisfaction from run-to-run.

5.9 Conclusion

In this paper we presented a framework to predict QoS (performance and power) for mobile devices. We considered user induced interruptions along with run-to-run variations within these quantities and constructed accurate models for these quantities. The models are constructed using various system events such as cache misses and page faults which can easily be recorded on modern mobile processors. The models were constructed using arbitrary polynomial chaos expansions which approximates stochastic systems by a set of orthogonal polynomial bases. These polynomials are constructed using only sample moments of the system events, requiring significantly less computation than with traditional techniques. To demonstrate our approach, we developed performance and power models for mobile workloads running on actual hardware. The proposed approach was shown to reduce the error by a factor of **2.67X** and **1.7X** for performance and power estimates respectively, over the regres-

sion based modeling techniques. Additionally, the practicality of these models were demonstrated by implementing a dynamic energy management controller on real mobile hardware. Able to effectively acclimate to the non-deterministic conditions of the system, the proposed controller improved the energy efficiency of the mobile device by **19%** over the existing governor implemented on Android devices.

Chapter 6

CONCLUSION

The use of mobile systems has penetrated every facet of modern life – from providing entertainment via gaming, video playback, and web browsing to professional use providing a email and network gateway, presentation editing, scheduling. However; the state-of-the-art in managing these systems is unfit to adapt to the highly interactive and non-deterministic characteristics of mobile applications. This dissertation provides some of the first steps to address these limitations by investigating several methods to accurately predict and analyze mobile workload characteristics as a function of controllable device parameters all while the system is subject to numerous sources of non-determinism. This is among the first work to provide an optimal representation of the non-deterministic characteristics of QoS and power – a necessary step to correct analysis/control of mobile systems. By utilizing the stochastic models, this work was able to improve device energy efficiency while maintaining specified user satisfaction levels both in an offline and online fashion. The offline, approach presented in this work proposes the use of stochastic network calculus techniques to conduct a state-space exploration of the mobile system – greatly reducing the number of experiments needed to determine the optimal operating conditions. The result of this is a light weight, static controller which may be bundled with any given application. Finally, this work provides an online method capable of dynamically adjusting to changing system conditions. This was accomplished through two tasks: first, a computationally efficient and optimal representation of power and QoS was constructed using polynomial chaos expansions; second, the design of closed-loop DEM controller is presented. The DEM controller was implemented on actual mobile

hardware and as such numerous implementation challenges were explored such as lack of real-time power sensors. The proposed controller is capable of increasing energy efficiency by more than 19% over existing Android governors.

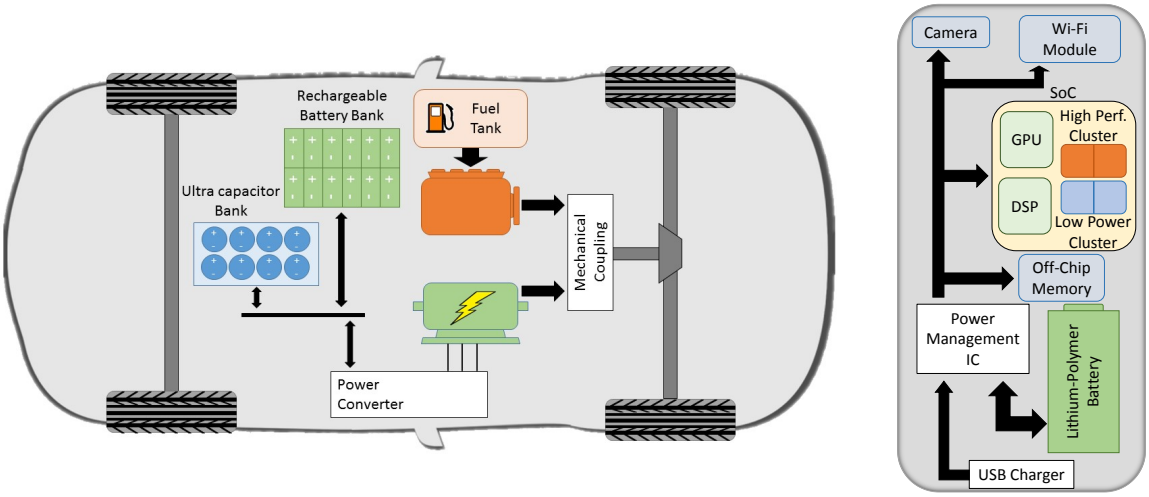
6.1 Possible Extensions and Future Work

While this dissertation provides necessary steps to defining and solving the DEM problem in the context of mobile systems; only a small subset of problems were investigated. There are many challenges not yet addressed in literature or do not have efficient solutions. Some of these challenges are outlined below.

6.1.1 Optimal Management for Hybrid Powered Systems

Historically, design practices for computing systems have focused primarily on maximizing performance, leaving energy minimization as an afterthought. However, in recent years the concept of sustainability has become mainstream. As such, computing and electronic systems have begun to adapt their design philosophy to incorporate this paradigm shift. A prevalent example of this is hybrid electric vehicles.

In regards to high level system design, mobile devices and hybrid vehicles share many commonalities. As illustrated in Figure 6.1, both systems contain multiple (unstable) energy producers and consumers with some type of energy buffer in between. Examples of energy producers in hybrid vehicles are various renewable energy systems (i.e. solar, regenerative brakes, etc.) as well as fossil fuel tanks and fuel cells. This is analogous to mobile systems which also have non-consistent power sources via USB or wall charging. In regards to power consumers, the main components for hybrid vehicles are the gas and electric engines. Much like a heterogeneous architecture of cpu cores found in mobile devices, these two engines have unique performance (torque) and power characteristics.



(a) (b)

Figure 6.1: An illustration of the power flow and components in (a) a hybrid vehicle and (b) a mobile phone.

A key difference between the two systems is the energy buffers. In mobile devices this is typically a single lithium-polymer battery – designed with the goal of maximizing energy density. In contrast, the hybrid vehicle uses several different energy buffers. The first being a bank of rechargeable batteries. These batteries are typically manufactured with the goal of either maximizing cycle lifetime or minimizing recharge time. Additionally, a ultra-capacitor bank may be utilized in order to provide quick, large bursts of power to the electrical engine, something the battery banks are incapable of providing. As such the drive control logic can divert part of the battery charge to the capacitor bank. A fuel tank provides additional energy storage which can be converted to torque via the combustion engine. Unlike, mobile devices, hybrid vehicles have a distinct separation between certain power consumers and producers, thus multiple power flows must be considered when controlling the vehicle.

Finally, with hybrid vehicles, one must consider the energy loss due to drive-trains and power conversion. In mobile systems, this type of power conversion is managed by a collection of voltage regulators or the power management unit (PMU) which

Table 6.1: Example Parameters for hybrid engine control.

Parameter	Description
F	Fuel tank level.
SoC_i	State of charge of the batteries within the battery bank.
$T_{ch}, \omega ch$	Torque applied by the electrical engine to recharge the battery bank.
$T_{elec}, \omega elec$	Torque and angular applied by the electrical engine to power the wheels.
$T_{mech}, \omega mech$	Torque and angular applied by the mechanical engine to power the wheels.
$T_{wheel}, \omega wheel$	Torque and angular velocities of speed.

typically has an efficiency of 95% or better. In contrast drive-trains have much worse efficiency, with as low as 20% power loss for combustion engines and 80% for electric engines [117]. This efficiency is largely related to the speed of the vehicle and thus, power splitting and scheduling techniques are needed to reduce this deficiency.

To optimize these systems numerous control strategies have already been explored including neural networks, dynamic programing, fuzzy rules, and even stochastic control [118]. However, much like with mobile systems, these works assume standard distributions to the sources of non-determinism as well as finite horizon conditions on the stochastic quantities. Our aPCE framework can be applied to these types of systems both for offline analysis and online control. Table 6.1 describes several inputs and variables used in electric assist control strategies [119]. Additionally, the formulation of the optimization problem is more complex than the case of mobile devices. Vehicle control requires multi-objective optimization requiring battery and fuel potentials as well as numerous other conditions to be maximized.

6.1.2 Adaptive Network Offloading

In recent years, the computing behavior of users has shifted largely to mobile devices. Mobile devices are now used in enterprise, information systems, gaming, educational, entertainment, and health care domains. As such the demand and expectations placed on these devices has increased dramatically. Although these devices have steadily become more and more powerful thanks to various design improvements and DEM techniques, ultimately these devices are limited by a finite energy source and provide less performance than their server/desktop counterparts. One method to minimize the effect of computation-intensive, power-hungry tasks is data/computation offloading. That is, the mobile device must dynamically determine whether or not to offload a given portion of a workload to some cloud computing network in order to optimize some energy, battery, or performance objective.

Currently, most existing literature in this area assumes mechanistic and deterministic workload power and performance characteristics. This typically results in a simple logic based control such that deterministic models are used to calculate the energy needed to compute the application versus offloading it. The device will then simply choose the lesser of the two results. However, these works typically either overlook or ignore any non-determinisms in the analysis of the mobile networks by assuming the network delay follows standard distributions (primarily the exponential distribution). As such, these works fail to accurately represent general network conditions along with the non-deterministic nature of mobile workload power and performance.

6.1.3 Task Migration

The primary focus of this work has been the development of frequency sensitive QoS and power models in presence of numerous system non-determinisms and utilize these models for DVFS. However; another viable control method to improve energy-efficiency is task migration. As mobile devices become more and more heterogeneous to service the numerous types of user demands, tasks will inherently become more suited to specific processing elements. However, the overhead involved with task migration may not be insignificant. Additionally, due to shared resource contention, task migration could hinder the performance of other running tasks. It is impossible for application designers to pre-characterize their workload for all possible systems, thus it becomes extremely important for the system to characterize the tasks to determine where/how to optimally execute them. From the system perspective, however, this becomes a learning problem. While some literature has already explored this issue, the non-deterministic nature of mobile workloads has been ignored. As such, this changes the nature of the learning problem into a stochastic one.

6.1.4 Fairness

Fair allocation of resources is a fundamental requirement of operating systems. Applications with similar priorities are expected to share equal resources (both time and in the case of heterogeneous computing, the physical processing element). For typical desktop and server systems, fairness is not an issue; however, systems which implore DEM techniques often ignore fairness requirements.

REFERENCES

[1] "How loading time affects your bottom line." `https://blog.kissmetrics.com/loading-time`, 2011. Accessed: 2015-05-14.

[2] S. Eyerman, K. Hoste, and L. Eeckhout, "Mechanistic-empirical processor performance modeling for constructing cpi stacks on real hardware," in *Performance Analysis of Systems and Software (ISPASS), 2011 IEEE International Symposium on*, pp. 216–226, IEEE, 2011.

[3] M. Pricopi, T. S. Muthukaruppan, V. Venkataramani, T. Mitra, and S. Vishin, "Power-performance modeling on asymmetric multi-cores," in *Compilers, Architecture and Synthesis for Embedded Systems (CASES), 2013 International Conference on*, pp. 1–10, IEEE, 2013.

[4] K. Van Craeynest, A. Jaleel, L. Eeckhout, P. Narvaez, and J. Emer, "Scheduling heterogeneous multi-cores through performance impact estimation (pie)," in *ACM SIGARCH Computer Architecture News*, vol. 40, pp. 213–224, IEEE Computer Society, 2012.

[5] T. S. Karkhanis and J. E. Smith, "A first-order superscalar processor model," in *ACM SIGARCH Computer Architecture News*, vol. 32, p. 338, IEEE Computer Society, 2004.

[6] C. Raiciu, C. Paasch, S. Barre, A. Ford, M. Honda, F. Duchene, O. Bonaventure, and M. Handley, "How hard can it be? designing and implementing a deployable multipath tcp," in *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, 2012.

[7] I. Grigorik, *High Performance Browser Networking: What every web developer should know about networking and browser performance*. O'Reilly Media, Incorporated, 2013.

[8] J. Donald and M. Martonosi, "Techniques for multicore thermal management: Classification and new exploration," in *ACM SIGARCH Computer Architecture News*, vol. 34, pp. 78–88, 2006.

[9] C. Isci, A. Buyuktosunoglu, C.-Y. Cher, P. Bose, and M. Martonosi, "An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget," in *Proceedings of the International Symposium of Microarchitecture*, pp. 347–358, 2006.

[10] D. Brooks and M. Martonosi, "Dynamic thermal management for high-performance microprocessors," in *Proceedings of the Seventh International Symposium on High-Performance Computer Architecture*, pp. 171–182, 2001.

[11] C. Isci, G. Contreras, and M. Martonosi, "Live, runtime phase monitoring and prediction on real systems with application to dynamic power management," in *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 359–370, 2006.

[12] V. Hanumaiah, S. Vrudhula, and K. S. Chatha, "Maximizing performance of thermally constrained multi-core processors by dynamic voltage and frequency control," in *Proceedings of the 2009 International Conference on Computer-Aided Design*, pp. 310–313, ACM, 2009.

[13] R. Rao and S. Vrudhula, "Performance optimal processor throttling under thermal constraints," in *Proceedings of the 2007 international conference on Compilers, architecture, and synthesis for embedded systems*, pp. 257–266, ACM, 2007.

[14] V. Hanumaiah, S. Vrudhula, and K. S. Chatha, "Performance optimal online DVFS and task migration techniques for thermally constrained multi-core processors," *IEEE Transactions on Computer-Aided Design*, vol. 30, pp. 1677–1690, November 2011.

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

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

[17] V. Hanumaiah and S. Vrudhula, "Temperature-aware dvfs for hard real-time applications on multicore processors," *IEEE Transactions on Computers*, vol. 61, no. 10, pp. 1484–1494, 2012.

[18] Y. Zhu, M. Halpern, and V. J. Reddi, "Event-based scheduling for energy-efficient qos (eqos) in mobile web applications," in *Proceedings of the IEEE 21st International Symposium on High Performance Computer Architecture*, pp. 137–149, 2015.

[19] V. Hanumaiah, D. Desai, B. Gaudette, C.-J. Wu, and S. Vrudhula, "STEAM: A Smart Temperature and Energy Aware Multicore Controller," *ACM Transactions on Embedded Computing Systems*, vol. 13, Sept. 2014.

[20] S. Murali, A. Mutapcic, D. Atienza, R. Gupta, S. Boyd, and G. De Micheli, "Temperature-aware processor frequency assignment for mpsocs using convex optimization," in *Hardware/Software Codesign and System Synthesis (CODES+ ISSS), 2007 5th IEEE/ACM/IFIP International Conference on*, pp. 111–116, IEEE, 2007.

[21] S. Kanev, K. Hazelwood, G.-Y. Wei, and D. Brooks, "Tradeoffs between power management and tail latency in warehouse-scale applications," in *IEEE International Symposium on Workload Characterization*, pp. 31–40, 2014.

[22] A. K. Coşkun, K. A. Whisnant, K. C. Gross, *et al.*, "Static and dynamic temperature-aware scheduling for multiprocessor socs," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 16, no. 9, pp. 1127–1140, 2008.

[23] M. A. Oxley, S. Pasricha, A. A. Maciejewski, H. J. Siegel, J. Apodaca, D. Young, L. Briceno, J. Smith, S. Bahirat, B. Khemka, *et al.*, "Makespan and energy robust stochastic static resource allocation of a bag-of-tasks to a heterogeneous computing system," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 26, no. 10, pp. 2791–2805, 2015.

[24] A. Colin, A. Kandhalu, and R. Rajkumar, "Energy-efficient allocation of real-time applications onto heterogeneous processors," in *Embedded and Real-Time Computing Systems and Applications (RTCSA), 2014 IEEE 20th International Conference on*, pp. 1–10, IEEE, 2014.

[25] M. K. Jeong, M. Erez, C. Sudanthi, and N. Paver, "A qos-aware memory controller for dynamically balancing gpu and cpu bandwidth use in an mpsoc," in *Proceedings of the 49th Annual Design Automation Conference*, pp. 850–855, ACM, 2012.

[26] U. Y. Ogras, P. Bogdan, and R. Marculescu, "An analytical approach for network-on-chip performance analysis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 29, no. 12, pp. 2001–2013, 2010.

[27] A. Alimonda, S. Carta, A. Acquaviva, A. Pisano, and L. Benini, "A feedback-based approach to dvfs in data-flow applications," *the IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 11, pp. 1691–1704, 2009.

[28] J. Sim, A. Dasgupta, H. Kim, and R. Vuduc, "A performance analysis framework for identifying potential benefits in GPGPU applications," in *ACM SIGPLAN Notices*, vol. 47, pp. 11–22, 2012.

[29] P. Bogdan, R. Marculescu, S. Jain, and R. T. Gavila, "An optimal control approach to power management for multi-voltage and frequency islands multiprocessor platforms under highly variable workloads," in *Proceedings of the Sixth IEEE/ACM International Symposium on Networks on Chip*, pp. 35–42, 2012.

[30] J. R. Wernsing and G. Stitt, "Elastic computing: A portable optimization framework for hybrid computers," *Parallel Computing*, vol. 38, no. 8, pp. 438–464, 2012.

[31] M. Damavandpeyma, S. Stuijk, T. Basten, M. Geilen, and H. Corporaal, "Throughput-constrained DVFS for scenario-aware dataflow graphs," in *Proceedings of the IEEE 19th Real-Time and Embedded Technology and Applications Symposium*, pp. 175–184, 2013.

[32] M. Fidler, "Survey of deterministic and stochastic service curve models in the network calculus," *Communications Surveys & Tutorials, IEEE*, vol. 12, no. 1, pp. 59–86, 2010.

[33] S. Schliecker, M. Negrean, G. Nicolescu, P. Paulin, and R. Ernst, "Reliable performance analysis of a multicore multithreaded system-on-chip," in *Proceedings of the 6th IEEE/ACM/IFIP international conference on Hardware/Software codesign and system synthesis*, pp. 161–166, 2008.

[34] Y. Qian, Z. Lu, and W. Dou, "Analysis of communication delay bounds for network on chips," in *Proceedings of the 2009 Asia and South Pacific Design Automation Conference*, pp. 7–12, 2009.

[35] "Ansys icepak features: Cooling software for the electronics industry." `http://www.ansys.com/products/icepak/features.asp?name=p1`.

[36] M. R. Stan, K. Skadron, M. Barcella, W. Huang, K. Sankaranarayanan, and S. Velusamy, "Hotspot: A dynamic compact thermal model at the processor-architecture level," *Microelectronics Journal*, vol. 34, no. 12, pp. 1153–1165, 2003.

[37] V. Hanumaiah, R. Rao, S. Vrudhula, and K. S. Chatha, "Throughput optimal task allocation under thermal constraints for multi-core processors," in *Proceedings of the 46th Annual Design Automation Conference*, pp. 776–781, ACM, 2009.

[38] B. Gaudette, V. Hanumaiah, S. Vrudhula, and M. Krunz, "Optimal range assignment in solar powered active wireless sensor networks," in *INFOCOM, 2012 Proceedings IEEE*, pp. 2354–2362, IEEE, 2012.

[39] B. Gaudette, V. Hanumaiah, M. Krunz, and S. Vrudhula, "Maximizing quality of coverage under connectivity constraints in solar-powered active wireless sensor networks," *ACM Transactions on Sensor Networks (TOSN)*, vol. 10, no. 4, p. 59, 2014.

[40] V. Hanumaiah, S. Vrudhula, and B. Gaudette, "Processor control system," Mar. 13 2014. US Patent App. 14/207,936.

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

[42] Intel Corporation, *Intel® 64 and IA-32 Architectures Software Developer's Manual*. No. 325462-052US, September 2014.

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

[44] C. G. Jones, R. Liu, L. Meyerovich, K. Asanovic, and R. Bodik, "Parallelizing the web browser," in *Proceedings of the First USENIX Workshop on Hot Topics in Parallelism*, 2009.

[45] W. Huang, K. Sankaranarayanan, R. J. Ribando, M. R. Stan, and K. Skadron, "An improved block-based thermal model in hotspot 4.0 with granularity considerations," in *Proceedings 6th Annual Workshop on Duplicating, Deconstructing, and Debanking (WDDD07), San Diego, CA, June*, vol. 10, pp. 1–10, 2007.

[46] V. Hanumaiah, S. Vrudhula, and K. S. Chatha, "Performance optimal online dvfs and task migration techniques for thermally constrained multi-core processors," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 11, pp. 1677–1690, 2011.

[47] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "Mcpat: an integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 469–480, ACM, 2009.

[48] M. Zapater, O. Tuncer, J. L. Ayala, J. M. Moya, K. Vaidyanathan, K. Gross, and A. K. Coskun, "Leakage-aware cooling management for improving server energy efficiency," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 10, pp. 2764–2777, 2015.

[49] Y. Zhang, Y. Liu, L. Zhuang, X. Liu, F. Zhao, and Q. Li, "Accurate cpu power modeling for multicore smartphones," tech. rep., February 2015.

[50] S. Mittal, "A survey of techniques for improving energy efficiency in embedded computing systems," *International Journal of Computer Aided Engineering and Technology*, vol. 6, no. 4, pp. 440–459, 2014.

[51] K. De Vogeleer, G. Memmi, P. Jouvelot, and F. Coelho, "The energy/frequency convexity rule: Modeling and experimental validation on mobile devices," in *International Conference on Parallel Processing and Applied Mathematics*, pp. 793–803, Springer, 2013.

[52] W. Liao, L. He, and K. M. Lepak, "Temperature and supply voltage aware performance and power modeling at microarchitecture level," *Proceedings of the IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 7, pp. 1042–1053, 2005.

[53] M. Huang, J. Renau, S.-M. Yoo, and J. Torrellas, "A framework for dynamic energy efficiency and temperature management," in *Proceedings of the 33rd annual ACM/IEEE International Symposium on Microarchitecture*, pp. 202–213, 2000.

[54] J. S. Seng, D. M. Tullsen, and G. Z. Cai, "Power-sensitive multithreaded architecture," in *Proceedings of the 2000 International Conference on Computer Design*, pp. 199–206, 2000.

[55] V. Hanumaiah and S. Vrudhula, "Energy-efficient operation of multicore processors by dvfs, task migration, and active cooling," *IEEE Transactions on Computers*, vol. 63, no. 2, pp. 349–360, 2014.

[56] P. Pillai and K. G. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems," in *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles*, 2001.

[57] T. Pering, T. Burd, and R. Brodersen, "Voltage scheduling in the lpARM microprocessor system," in *Proceedings of the International Symposium on Low Power Electronics and Design*, 2000.

[58] J. R. Lorch and A. J. Smith, "Improving dynamic voltage scaling algorithms with pace," in *Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, 2001.

[59] D. Grunwald, C. B. Morrey, III, P. Levis, M. Neufeld, and K. I. Farkas, "Policies for dynamic clock scheduling," in *Proceedings of the 4th Conference on Symposium on Operating System Design & Implementation - Volume 4*, 2000.

[60] K. Flautner and T. Mudge, "Vertigo: Automatic performance-setting for linux," in *Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, 2002.

[61] R. Jejurikar, C. Pereira, and R. Gupta, "Leakage aware dynamic voltage scaling for real-time embedded systems," in *Proceedings of the 41st Annual Design Automation Conference*, 2004.

[62] K. Kang, J. Kim, S. Yoo, and C.-M. Kyung, "Temperature-aware integrated DVFS and power gating for executing tasks with runtime distribution," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 29, pp. 1381–1394, Sept 2010.

[63] A. Kahng, S. Kang, R. Kumar, and J. Sartori, "Enhancing the efficiency of energy-constrained DVFS designs," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 21, pp. 1769–1782, Oct 2013.

[64] J. S. Lee, K. Skadron, and S. W. Chung, "Predictive temperature-aware DVFS," *IEEE Transactions on Computers*, vol. 59, pp. 127–133, Jan 2010.

[65] D. Shingari, A. Arunkumar, and C.-J. Wu, "Characterization and throttling-based mitigation of memory interference for heterogeneous smartphones," in *Proceedings of the IEEE International Symposium on Workload Characterization*, pp. 22–33, 2015.

[66] M.-S. Alouini, A. Abdi, and M. Kaveh, "Sum of gamma variates and performance of wireless communication systems over nakagami-fading channels," *Proceedings of the IEEE Transactions on Vehicular Technology*, vol. 50, no. 6, pp. 1471–1480, 2001.

[67] "Mozilla firefox." `https://www.mozilla.org`.

[68] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The parsec benchmark suite: Characterization and architectural implications," in *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, pp. 72–81, 2008.

[69] "Videolan organization." `http://www.videolan.org`.

[70] "Android NDK." `https://developer.android.com/tools/sdk/ndk/index.html`. Accessed: 2014-8-11.

[71] Y. Zhu and V. J. Reddi, "Webcore: architectural support for mobileweb browsing," in *Proceeding of the 41st annual international symposium on Computer architecuture*, pp. 541–552, 2014.

[72] J.-Y. Le Boudec and P. Thiran, *Network calculus: a theory of deterministic queuing systems for the internet*, vol. 2050. Springer Science & Business Media, 2001.

[73] D. Pandiyan, S.-Y. Lee, and C.-J. Wu, "Performance, energy characterizations and architectural implications of an emerging mobile platform benchmark suite-mobilebench," in *Proceedings of the IEEE International Symposium on Workload Characterization*, pp. 133–142, 2013.

[74] A. S. V. Palladi and A. Starikovskiy, "The ondemand governor: past, present and future," in *Proceedings of Linux Symposium*, vol. 2, pp. 3–3, 2001.

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

[76] A. Ramachandra and M. DeHart, "Consumerscape 360: Smartphone purchase drivers," *IDC Research*, 2017.

[77] Y. Zhu and V. J. Reddi, "High-performance and energy-efficient mobile web browsing on big/little systems," *Proceedings of the 19th High Performance Computer Architecture*, 2013.

[78] V. Pallipadi and A. Starikovskiy, "The ondemand governor," in *Proceedings of the Linux Symposium*, vol. 2, pp. 215–230, sn, 2006.

[79] Y. Endo, Z. Wang, J. B. Chen, and M. I. Seltzer, "Using latency to evaluate interactive system performance," *ACM SIGOPS Operating Systems Review*, vol. 30, no. si, pp. 185–199, 1996.

[80] G. Lindgaard, G. Fernandes, C. Dudek, and J. Brown, "Attention web designers: You have 50 milliseconds to make a good first impression!," *Behaviour & information technology*, vol. 25, no. 2, pp. 115–126, 2006.

[81] D. Lo, T. Song, and G. E. Suh, "Prediction-guided performance-energy trade-off for interactive applications," in *Proceedings of the 48th International Symposium on Microarchitecture*, pp. 508–520, ACM, 2015.

[82] M. B. Breughe, S. Eyerman, and L. Eeckhout, "Mechanistic analytical modeling of superscalar in-order processor performance," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 11, no. 4, p. 50, 2015.

[83] R. Barik, N. Farooqui, B. T. Lewis, C. Hu, and T. Shpeisman, "A black-box approach to energy-aware scheduling on integrated cpu-gpu systems," in *Proceedings of the 2016 International Symposium on Code Generation and Optimization*, pp. 70–81, ACM, 2016.

[84] J. Zhao, H. Cui, J. Xue, X. Feng, Y. Yan, and W. Yang, "An empirical model for predicting cross-core performance interference on multicore processors," in *Proceedings of the 22nd international conference on Parallel architectures and compilation techniques*, pp. 201–212, IEEE Press, 2013.

[85] S. Breß, F. Beier, H. Rauhe, K.-U. Sattler, E. Schallehn, and G. Saake, "Efficient co-processor utilization in database query processing," *Information Systems*, vol. 38, no. 8, pp. 1084–1096, 2013.

[86] M. Boyer, J. Meng, and K. Kumaran, "Improving gpu performance prediction with data transfer modeling," in *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2013 IEEE 27th International*, pp. 1097–1106, IEEE, 2013.

[87] Y. Zhu and V. J. Reddi, "Greenweb: language extensions for energy-efficient mobile web computing," in *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pp. 145–160, ACM, 2016.

[88] S. Oladyshkin and W. Nowak, "Data-driven uncertainty quantification using the arbitrary polynomial chaos expansion," *Reliability Engineering & System Safety*, vol. 106, pp. 179–190, 2012.

[89] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, *et al.*, "The gem5 simulator," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 1–7, 2011.

[90] T. Šimunić, L. Benini, and G. De Micheli, "Cycle-accurate simulation of energy consumption in embedded systems," in *Proceedings of the 36th annual ACM/IEEE Design Automation Conference*, pp. 867–872, ACM, 1999.

[91] D. A. Freedman, *Statistical models: theory and practice*. Cambridge Univ. Press, 2005.

[92] D. Xiu and G. E. Karniadakis, "The wiener-askey polynomial chaos for stochastic differential equations," *SIAM Journal on Scientific Computing*, vol. 24, no. 2, pp. 614–644, 2002.

[93] N. Wiener, "The homogeneous chaos," *American Journal of Mathematics*, vol. 60, no. 4, pp. 897–936, 1938.

[94] R. G. Ghanem and P. D. Spanos, *Stochastic finite elements: a spectral approach*. Courier Corporation, 2003.

[95] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 58, no. 1, pp. 267–288, 1996.

[96] R. Kohavi *et al.*, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *Ijcai*, vol. 14, pp. 1137–1145, 1995.

[97] T. Zhang, "Solving large scale linear prediction problems using stochastic gradient descent algorithms," in *Proceedings of the twenty-first international conference on Machine learning*, p. 116, ACM, 2004.

[98] J. Cioffi and T. Kailath, "Fast, recursive-least-squares transversal filters for adaptive filtering," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 32, no. 2, pp. 304–337, 1984.

[99] Google, "Pixel, phone by google - made by google." `https://madeby.google.com/phone/`, 2013. Accessed: 2017-03-31.

[100] Qualcomm, "Snapdragon 821 processor." `https://www.qualcomm.com/products/snapdragon/processors/821`, 2017. Accessed: 2017-03-31.

[101] Monsoon Solutions Inc., "Power monitor." `https://www.msoon.com/LabEquipment/PowerMonitor/`, 2017. Accessed: 2017-03-31.

[102] Google, "Webview android developers." `https://developer.android.com/reference/android/webkit/WebView.html`. Accessed: 2016-08-15.

[103] Amazon, "The top sites on the web." `http://www.alexa.com/topsites`. Accessed: 2016-08-30.

[104] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "Mibench: A free, commercially representative embedded benchmark suite," in *Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop on*, pp. 3–14, IEEE, 2001.

[105] M. van der Schee, "2048.c." `https://github.com/mevdschee/2048.c`. Accessed: 2016-10-07.

[106] M. L. Li and et al., "Gnu go." `https://www.gnu.org/software/gnugo/`, 2009. Accessed: 2016-10-07.

[107] J. Seward, "bzip2." `http://www.bzip2.org`, 2007. Accessed: 2016-10-07.

[108] D. Chaffey, "Mobile marketing statistics compilation." `http://www.smartinsights.com/mobile-marketing/mobile-marketing-analytics/mobile-marketing-statistics/`, 2016. Accessed: 2016-11-10.

[109] C. Isci and M. Martonosi, "Runtime power monitoring in high-end processors: Methodology and empirical data," in *Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*, p. 93, IEEE Computer Society, 2003.

[110] A. Verma, G. Dasgupta, T. K. Nayak, P. De, and R. Kothari, "Server workload analysis for power minimization using consolidation," in *Proceedings of the 2009 conference on USENIX Annual technical conference*, pp. 28–28, USENIX Association, 2009.

[111] E. M. Elnozahy, M. Kistler, and R. Rajamony, "Energy-efficient server clusters," in *International Workshop on Power-Aware Computer Systems*, pp. 179–197, Springer, 2002.

[112] V. Patel, "Kalman-based stochastic gradient method with stop condition and insensitivity to conditioning," *SIAM Journal on Optimization*, vol. 26, no. 4, pp. 2620–2648, 2016.

[113] S. Reda and A. Belouchrani, "Blind identification of power sources in processors," in *Proceedings of the 2017 Design Automation and Test in Europe*, 2017.

[114] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li, "Ferret: a toolkit for content-based similarity search of feature-rich data," *ACM SIGOPS Operating Systems Review*, vol. 40, no. 4, pp. 317–330, 2006.

[115] T. Rivlin, *An Introduction to the Approximation of Functions*. Blaisdell book in numerical analysis and computer science, Dover Publications, 1969.

[116] J. Neter, M. H. Kutner, C. J. Nachtsheim, and W. Wasserman, *Applied linear statistical models*, vol. 4. Irwin Chicago, 1996.

[117] M. A. Roscher, R. Michel, and W. Leidholdt, "Improving energy conversion efficiency by means of power splitting in dual drive train ev applications," *International Journal of Vehicular Technology*, vol. 2013, 2013.

[118] Y.-H. Cheng and C.-M. Lai, "Control strategy optimization for parallel hybrid electric vehicles using a memetic algorithm," *Energies*, vol. 10, no. 3, p. 305, 2017.

[119] A. Panday and H. O. Bansal, "A review of optimal energy management strategies for hybrid electric vehicle," *International Journal of Vehicular Technology*, vol. 2014, 2014.

[120] R. H. Cameron and W. T. Martin, "Transformations of weiner integrals under translations," *Annals of Mathematics*, vol. 45, no. 2, pp. 386–396, 1944.

[121] R. H. Cameron and W. T. Martin, "The orthogonal development of non-linear functionals in series of fourier-hermite functionals," *Annals of Mathematics*, vol. 48, no. 2, pp. 385–392, 1947.

[122] P. Cinnella, P. Congedo, L. Parussini, and V. Pediroda, "Quantification of thermodynamic uncertainties in real gas flows," *International Journal of Engineering Systems Modelling and Simulation*, vol. 2, no. 1-2, pp. 12–24, 2010.

[123] S. S. Isukapalli and P. G. Georgopoulos, "Computational methods for the efficient sensitivity and uncertainty analysis of models for environmental and biological systems," *Computational Chemodynamics Laboratory Environmental and Occupational Health Sciences Institute*, vol. 170, 1999.

[124] S. Vrudhula, J. M. Wang, and P. Ghanta, "Hermite polynomial based interconnect analysis in the presence of process variations," *IEEE Transactions on Computer-Aided Design of Integrated circuits and systems*, vol. 25, no. 10, pp. 2001–2011, 2006.

APPENDIX A

POLYNOMIAL CHAOS EXPANSION

A.1 A brief history of Polynomial Chaos Theory

When analyzing dynamical systems, a key task is to understand the relationship between the system's inputs and the system's output or response. In the event that the system inputs are represented by random quantities such as random variables, stochastic processes or random fields; the system response will also be a random quantity. One of most popular methodologies to represent such a relationship is polynomial chaos expansion or PCE for short.

Polynomial chaos expansion (also called Wiener-Hermite expansion or Fourier-Hermite expansion) dates back to 1938 when Norbert Wiener examined the case in which a system's response could be represented by a collection of random variables following Brownian Motion which he coined Homogeneous Chaos [93]. Utilizing his prior harmonic's equations, Wiener defined Homogeneous Chaos as the set of all multiple integrals taken with respect to a multidimensional Wiener process. The end result of his work was the derivation of Hermite polynomials which construct an orthonormal random basis for his Homogeneous Chaos.

Several years later, Robert Cameron and William Martin refined Wiener's work [120, 121]. They provided a more intuitive formulation for the Wiener-Hermite expansion via a Fourier expansion of the brown motion process. Therefore, the actual system response, y_{act} can be represented by a system of random variables:

$$y(\boldsymbol{\xi}) = \sum_{j=0}^{\infty} c_j H_j(\boldsymbol{\xi}), \tag{A.1}$$

where c_j are coefficients of each Hermite polynomial basis functions, H_j subject to the random input vector, $\boldsymbol{\xi} = \{\xi_1 \xi_2 \dots \xi_m\}$ – a multidimensional Brownian field.

Additionally, Cameron and Martin proved that the polynomial representation of y of any distribution has optimal L2 convergence to the actual process y for the case of $\boldsymbol{\xi}$ being a Brownian field. That is, given an infinite number of samples $\boldsymbol{\xi}_i$ and corresponding response $y_{act,i}$

$$\lim_{i \to \infty} E\left[y_{act,i} - y(\boldsymbol{\xi}_i)\right] = 0. \tag{A.2}$$

A.2 Generalized Polynomial Chaos

For decades the works of Wiener, Cameron, and Martin served as the foundation for all PC works. It has been applied to numerous domains such as modeling expanding gases [122], biological ecosystems [123], and the effects of process variations on circuit delay [124] to name a few. However, it was not until 2001 Dongbin Xiu and George Karnidakis formally generalized the work of Cameron and Martin to various continuous and discrete distributions. They called this the Generalized Polynomial Chaos (gPC) framework [92].

Given a probability space (Ω, A, Γ) with sample space Ω, σ algebra A, and probability measure Γ; the goal of gPC is to find an optimal mapping of the system response $y_a ct = f(\boldsymbol{\xi})$ to

$$y(\boldsymbol{\xi}) = \sum_{j=0}^{\infty} c_j \Phi_j(\boldsymbol{\xi}), \tag{A.3}$$

where c_j are the deterministic coefficients, $\xi \in \Omega$ are independent random variables[1], and $\Phi_j(\xi)$ are a collection of multi-dimensional orthogonal polynomials in the random variables $\boldsymbol{\xi}$. It should be noted that the collection of Φ_k constitutes an orthonormal basis of an infinite dimensional Hilbert space. We denote this space Θ.

A Hilbert space provides a means of defining the inner product on a space of random variables.

Definition A.2.1 (Hilbert Space)**.** Let $\mathcal{H}$ be a vector space over some field $\mathcal{F}$ with inner product $\langle f, g\rangle \;\; f, g \in \mathcal{H}$ defined. The norm in $\mathcal{H}$ is $\|f\| = \sqrt{\langle f, f\rangle}$, and the metric is $d(f, g) = \|f - g\|$. $\mathcal{H}$ is called a Hilbert space if it is complete as a metric space.

Definition A.2.2 (Orthogonality)**.** Two elements, x and y of an inner product space are said to be orthogonal if $\langle x, y\rangle = 0$. If in addition, $\|x\| = \|y\| = 1$, they are orthonormal.

Definition A.2.3 (Orthonormal Basis)**.** An orthonormal sequence $\{\phi_k\}_{k=1}^{\infty}$, in a Hilbert space is called an orthonormal basis if the only element outside the basis that is orthogonal to every element in the basis is the zero element. That is, an orthonormal basis is a maximal subset of elements that are mutually orthogonal.

Lemma A.2.1. Let $\{\phi_k\}_{k=1}^{\infty}$ be an orthonormal basis of a Hilbert space. Then the infinite series $\sum_{k=1}^{\infty} \langle x, \phi_k\rangle \phi_k$ converges in the norm to x.

This lemma states that in order to obtain a convergent infinite series representation of an element in Θ, we need to find an orthonormal basis. In a random space such as Θ, the inner product of any two elements is the expectation of their product or in other words the correlation between the two elements. Thus if we assume a probability measure of Γ over the sample space Ω than the inner product on Θ is defined as

$$\langle \phi_i(\boldsymbol{\xi}), \phi_j(\boldsymbol{\xi})\rangle = E[\phi_i(\boldsymbol{\xi}), \phi_j(\boldsymbol{\xi})] = \int_{\Omega} \phi_i(\boldsymbol{\xi})\phi_j(\boldsymbol{\xi}) d\Gamma(\boldsymbol{\xi}) \tag{A.4}$$

As shown by [93], a Hermite polynomial is a valid function for ϕ when $\boldsymbol{\xi}$ is a collection of zero mean, unit variance Gaussian random processes.

Definition A.2.4 (Hermite Polynomial)**.** Let $\xi_1, \xi_2, \ldots,$ be an infinite collection of unit Gaussian random variables. The Hermite polynomial of order k is defined as

$$H_k(i_1, i_2, \ldots, i_p) = (-1)^j e^{\frac{\boldsymbol{\xi}^T\boldsymbol{\xi}}{2}} \frac{d^j}{d\xi} e^{-\frac{\boldsymbol{\xi}^T\boldsymbol{\xi}}{2}}. \tag{A.5}$$

where $\boldsymbol{\xi} = [\xi_{i_1}, \xi_{i_2}, \ldots \xi_{i_k}]$.

[1]If the input $\boldsymbol{x}$ is not independent, then it must be possible to decorrelate $\boldsymbol{x}$ into *bmxi* such that the elements ξ are independent. This can be accomplished using methods such as principal component analysis and Karhunen-Loeve expansion.

	Variable Distribution	Polynomial Class
Continuous	Gaussian	Hermite
	Log-normal	Hermite
	Gamma	Laguerre
	Beta	Jacobi
	Uniform	Legendre
Discrete	Poisson	Charlier
	Binomial	Krawtchouk
	Negative Binomial	Meixner
	Hypergeometric	Hahn

Table A.1: Optimal Polynomials for Various Probability Spaces.

Note that any choice of k variables from $\{\xi_1, \xi_2, \dots, \}$ is allowed, including repetitions. Thus if we truncate the infinite set $\{\xi_1, \xi_2, \dots, \}$ to r variables, there will be $(k+r-1)!/k!(r-1)!$ polynomials of degree k. As an example we provide the results for a 2 variable polynomial of degrees 0, 1, 2, and 3.

$$\text{order } 0: \quad H_0(\{\}) = 1, \tag{A.6}$$

$$\text{order } 1: \quad H_1(1) = \xi_1, \quad H_1(2) = \xi_2, \tag{A.7}$$

$$\text{order } 2: \quad H_2(1,1) = \xi_1^2 - 1, \quad H_2(1,2) = \xi_1\xi_2, \tag{A.8}$$

$$H_2(2,2) = \xi_2^2 - 1, \tag{A.9}$$

$$\text{order } 3: \quad H_3(1,1,1) = \xi_1^3 - 3\xi_1, \tag{A.10}$$

$$H_3(2,1,1) = \xi_1^2\xi_2 - \xi_2, \tag{A.11}$$

$$H_3(2,2,1) = \xi_1\xi_2^2 - \xi_1, \tag{A.12}$$

$$H_3(2,2,2) = \xi_2^3 - 3\xi_2, \tag{A.13}$$

The fact that the elements of $\boldsymbol{\xi}$ are Gaussian is not a restriction. Many previous works have examined the optimal polynomial to use in the expansion for numerous distributions of $\boldsymbol{\xi}$. Table A.1 summarizes these polynomials. Additionally, these polynomials can be mixed in the event that the elements of the input follow different distributions. The process of constructing the mixed basis functions is outlined by Ghanem and Spannos [94] and will be explored in Section A.3.2.

The unknowns in the expansion of the stochastic response as shown in Equation (A.3) are the deterministic coefficients, c_j. In practice, we truncate this expansion to a finite number of basis functions, thus the number of unknowns to solve for are finite. One method to determine these coefficients is based on the principle of orthogonality.

Given two finite dimensional inner products spaces V of dimension n and W of dimension $m < n$, we wish to find the best approximation of $v \in V$ by a vector $w \in W$[2]. The principle of orthogonality states that optimal w must be orthogonal to the error $v - w$ such that $\langle v - w, w \rangle = 0$.

[2]Best is in the sense of minimizing the norm of the error, i.e. $\|v - w\|$.

Therefore to determine the coefficients α_j where max value of j is truncated to r thus we must solve a system of r equations

$$\langle y(\boldsymbol{\xi}) - f(\xi), \Phi_j(\boldsymbol{\xi}) \rangle = 0, \quad j = 0, 1, \ldots, r. \tag{A.14}$$

A.3 Arbitrary Polynomial Chaos

In the previous section we explored the case in which the actual system dynamics, $f(\boldsymbol{\xi})$ are known along with the distributions of each element of $\boldsymbol{\xi}$; however in practice you may not know the true distribution of $\boldsymbol{\xi}$ nor the true response function $f(\boldsymbol{\xi})$. In 2011, Oladyshkin and Nowak developed arbitrary polynomial chaos (aPC) to handle such situations [88]. Arbitrary polynomial chaos is a data-driven extension to the gPC by estimating the probability measure $\boldsymbol{\xi}$ using a finite number of moments.

A.3.1 Single Variable aPC

To begin our examination of aPC we will only consider the one-dimensional case (i.e. the size of $\boldsymbol{\xi}$ is one). Much like gPC, aPC considers a stochastic process in the probability space (Ω, A, Γ) with space of events Ω, σ-algebra A and probability measure Γ. Additionally it is assumed that the stochastic model $y_{act} = f(\xi)$ exists with $\xi \in \Omega$ although it need not be known. Arbitrary polynomial chaos states that $f(\xi)$ may be expanded as

$$f(\xi) \approx \sum_{i=0}^{d} c_i P^{(i)}(\xi) \tag{A.15}$$

where d is the order of the expansion and c_i are the expansion coefficients. The polynomials $P^{(k)}(\xi),\ k = 0, \ldots, d$ forms an orthogonal basis with respect to the probability measure Γ. It is the goal of aPC to determine the values of all c_i along with the polynomial functions $P^{(i)}(\xi)$ and will do so using a moment-based analysis.

The structure of $P^{(k)}(\xi)$ is as follows

$$P^{(k)}(\xi) = \sum_{i=0}^{k} p_i^{(k)} \xi^i, \quad k = \{0, \ldots, d\}, \tag{A.16}$$

Where $p_i^{(k)}$ are the coefficients of $P^{(k)}(\xi)$.

Applying the condition of orthonormality to the polynomial $P^{(k)}(\xi)$ results in the condition

$$\int_{\xi \in \Omega} P^{(k)}(\xi) P^{(l)}(\xi) d\Gamma(\xi) = \delta_{kl} \quad \forall l = 0, \ldots, d, \tag{A.17}$$

where δ_{kl} is the Kronecker delta[3].

If we assume instead of normality, the polynomials $P^{(i)}(\xi),\ i = 0, \ldots, d$ are monic we introduce the condition

$$p_k^{(k)} = 1 \quad \forall k \tag{A.18}$$

[3]The Kronecker delta function, δ_{kl}, is equal to 1 only if $k = l$; otherwise it is 0.

Therefore a closed set of equations emerge to determine the coefficients of k^{th} polynomial.

$$\begin{aligned}
\int_{\xi\in\Omega} p_0^{(0)} \left[\sum_{i=0}^{k} p_i^{(k)}\xi^i\right] d\Gamma(\xi) &= 0 \\
\int_{\xi\in\Omega} \left[\sum_{i=0}^{1} p_i^{(1)}\xi^i\right] \left[\sum_{i=0}^{k} p_i^{(k)}\xi^i\right] d\Gamma(\xi) &= 0 \\
\int_{\xi\in\Omega} \left[\sum_{i=0}^{2} p_i^{(2)}\xi^i\right] \left[\sum_{i=0}^{k} p_i^{(k)}\xi^i\right] d\Gamma(\xi) &= 0 \\
&\vdots \\
\int_{\xi\in\Omega} \left[\sum_{i=0}^{k-1} p_i^{(k-1)}\xi^i\right] \left[\sum_{i=0}^{k} p_i^{(k)}\xi^i\right] d\Gamma(\xi) &= 0 \\
p_k^{(k)} &= 1.
\end{aligned} \tag{A.19}$$

We can simplify these equations by utilizing (A.18) along with substituting the first equation into the second, the first and second into the third, and so-on.

$$\int_{\xi\in\Omega} \underbrace{p_0^{(0)}}_{1} \left[\sum_{i=0}^{k} p_i^{(k)}\xi^i\right] d\Gamma(\xi) = 0$$

$$\int_{\xi\in\Omega} \sum_{i=0}^{k} p_i^{(k)}\xi^i d\Gamma(\xi) = 0 \tag{A.20}$$

$$\int_{\xi\in\Omega} \left[\sum_{i=0}^{1} p_i^{(1)}\xi^i\right] \left[\sum_{i=0}^{k} p_i^{(k)}\xi^i\right] d\Gamma(\xi) = 0$$

$$\int_{\xi\in\Omega} \underbrace{p_1^{(1)}}_{1} \xi \left[\sum_{i=0}^{k} p_i^{(k)}\xi^i\right] d\Gamma(\xi) + \underbrace{\int_{\xi\in\Omega} p_0^{(1)} \left[\sum_{i=0}^{k} p_i^{(k)}\xi^i\right] d\Gamma(\xi)}_{\text{by (A.20) this is equal to 0}} = 0$$

$$\int_{\xi\in\Omega} \sum_{i=0}^{k} p_i^{(k)}\xi^{i+1} d\Gamma(\xi) = 0 \tag{A.21}$$

$$\int_{\xi\in\Omega} \left[\sum_{i=0}^{2} p_i^{(2)}\xi^i\right] \left[\sum_{i=0}^{k} p_i^{(k)}\xi^i\right] d\Gamma(\xi) = 0$$

$$\int_{\xi\in\Omega} \underbrace{p_2^{(2)}}_{1} \xi^2 \left[\sum_{i=0}^{k} p_i^{(k)}\xi^i\right] d\Gamma(\xi) + \underbrace{\int_{\xi\in\Omega} p_1^{(2)}\xi \left[\sum_{i=0}^{k} p_i^{(k)}\xi^i\right] d\Gamma(\xi)}_{\text{by (A.21) this is equal to 0}} + \underbrace{\int_{\xi\in\Omega} p_0^{(2)} \left[\sum_{i=0}^{k} p_i^{(k)}\xi^i\right] d\Gamma(\xi)}_{\text{by (A.20) this is equal to 0}} = 0$$

$$\int_{\xi\in\Omega} \sum_{i=0}^{k} p_i^{(k)}\xi^{i+2} d\Gamma(\xi) = 0 \tag{A.22}$$

$$\vdots$$

$$\int_{\xi\in\Omega} \sum_{i=0}^{k} p_i^{(k)}\xi^{i+k-1} d\Gamma(\xi) = 0 \tag{A.23}$$

$$p_k^{(k)} = 1. \tag{A.24}$$

Finally, we take notice that the i^{th} moment of ξ's distribution can be represented by

$$\mu_i = \int_{\xi\in\Omega} \xi^i d\Gamma(\xi) \tag{A.25}$$

Thus we can reduce (A.20-A.22) to be functions of moments

$$\begin{aligned}
\sum_{i=0}^{k} p_i^{(k)} \mu_i &= 0 \\
\sum_{i=0}^{k} p_i^{(k)} \mu_{i+1} &= 0 \\
\sum_{i=0}^{k} p_i^{(k)} \mu_{i+2} &= 0 \\
&\vdots \\
\sum_{i=0}^{k} p_i^{(k)} \mu_{i+k-1} &= 0 \\
p_k^{(k)} &= 1
\end{aligned} \tag{A.26}$$

Therefore the coefficients of the k^{th} polynomial basis can be solved using a system of linear equations related to the $2k-1$ moments of the random variable. The solutions up to the third order polynomial are the following:

$$\begin{aligned}
\text{order 0:} \quad & P^{(0)}(\xi) = 1 \\
\text{order 1:} \quad & P^{(1)}(\xi) = \xi - \mu_1 \\
\text{order 2:} \quad & P^{(2)}(\xi) = \xi^2 + \frac{\mu_3 - \mu_1\mu_2}{\mu_1^2 - \mu_2}\xi + \frac{\mu_2^2 - \mu_1\mu_3}{\mu_1^2 - \mu_2} \\
\text{order 3:} \quad & P^{(3)}(\xi) = \xi^3 \\
& \quad + \frac{-\mu_5\mu_1^2 + \mu_4\mu_1\mu_2 + \mu_1\mu_3^2 - \mu_2^2\mu_3 + \mu_5\mu_2 - \mu_4\mu_3)}{\mu_4\mu_1^2 - 2\mu_1\mu_2\mu_3 + \mu_2^3 - \mu_4\mu_2 + \mu_3^2}\xi^2 \\
& \quad + \frac{-\mu_2^2\mu_4 + \mu_2\mu_3^2 - \mu_1\mu_5\mu_2 - \mu_1\mu_3\mu_4 - \mu_5\mu_3 + \mu_4^2}{\mu_4\mu_1^2 - 2\mu_1\mu_2\mu_3 + \mu_2^3 - \mu_4\mu_2 + \mu_3^2}\xi \\
& \quad \frac{-\mu_5\mu_2^2 + 2\mu_2\mu_3\mu_4 - \mu_3^3 + \mu_1\mu_5\mu_3 - \mu_1\mu_4^2}{\mu_4\mu_1^2 - 2\mu_1\mu_2\mu_3 + \mu_2^3 - \mu_4\mu_2 + \mu_3^2}
\end{aligned}$$

If we note that the moment generating function of zero mean, unit variance Gaussian random variables is $M(\xi) = e^{\frac{\xi^2}{2}}$ we can see that the (A.5) reduces to the above polynomials with simple substitution.

Additionally it might be useful to find the orthonormal basis, $\Psi^{(k)}$, rather than the monic. To do so we scale by the norm of $P^{(k)}$:

$$\Phi^{(k)} = \frac{1}{\|P^{(k)}\|} \sum_{i=0}^{k} p_i^{(k)} \xi^i \tag{A.27}$$

where

$$\left\|P^{(k)}\right\|^2 = \int_{\xi\in\Omega}\left[\sum_{i=0}^{k} p_i^{(k)}\xi^i\right]. \tag{A.28}$$

Once again if we note (A.25) it is possible to reduce (A.28) to a function of the $2k$ moments of ξ's distribution.

This shows that the orthonormal basis functions can be constructed with only $2k$ sample moments of ξ.

Finally, to solve for the coefficients we leverage the convergence theorem of PC models. Given N samples of ξ and the corresponding measured response y_{act} we solve the linear least square minimization problem.

$$\min_{c_0,c_1,\ldots,c_d}\sum_{n=1}^{N}\left(y_{act,n} - \sum_{i=0}^{d} c_i P^{(i)}(\xi_n)\right)^2. \tag{A.29}$$

A.3.2 Multivariate aPC

Arbitrary polynomial chaos can also be applied to multi-input systems. To do so we utilize the results of Ghanem and Spannos [94]. Given the M-dimensional input vector $\boldsymbol{\xi} = [\xi_1,\ \xi_2,\ \ldots,\ \xi_M]$ such that all parameters within $\boldsymbol{\xi}$ are independent, the multi-dimensional basis can be constructed as a simple product of the corresponding univariate polynomials.

The procedure to determine Ψ_k, the multivariate basis of order k, can be summarized as the following:

Using the set of equations shown in (A.26), determine the set of univariate polynomials

$$P_j^{(k)}(\xi_j) = \sum_{i=0}^{k} p_{i,j}^{(k)}\xi_j, \quad k = 0\ldots d, \quad j = 0\ldots M. \tag{A.30}$$

The multivariate polynomial can be solved as

$$\Phi_i(\boldsymbol{\xi}) = \prod_{j=1}^{M} P_j^{(\lambda_j^i)}(\xi_j), \quad \sum_{j=1}^{M}\alpha_j^k \le k, \quad i = 1\ldots R, \tag{A.31}$$

where $R = (k+M-1)!/k!(M-1)$ is the total number of multivariate polynomials of degree k and λ_j^i is a multivariate index which is capable of enumerating all possible products of individual univariate basis functions. In other words, think of Λ as an $R \times M$ matrix such that the sum of each row of Λ is no more than k. The element at row i, column j is $\lambda_j^i \in \{0,\ldots,k\}$ which indicates the degree of $P_j^{(i)}$. Figure A.1 illustrates an 2-variable, 3^{rd} order example. The set $\{\Phi_i(\boldsymbol{\xi})\}_{i=1}^{R}$ will constitute the multivariate basis, Ψ_k.

$$\Lambda = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 1 & 1 \\ 2 & 0 \\ 0 & 2 \\ 2 & 1 \\ 1 & 2 \\ 3 & 0 \\ 0 & 3 \end{bmatrix} \quad \overbrace{\begin{array}{ll} \Phi_0(\xi) = P_1^{(0)}(\xi)P_2^{(0)}(\xi) & \text{Order 0} \\ \Phi_1(\xi) = P_1^{(1)}(\xi)P_2^{(0)}(\xi) & \text{Order 1} \\ \Phi_2(\xi) = P_1^{(0)}(\xi)P_2^{(1)}(\xi) & \\ \Phi_3(\xi) = P_1^{(1)}(\xi)P_2^{(1)}(\xi) & \\ \Phi_4(\xi) = P_1^{(2)}(\xi)P_2^{(0)}(\xi) & \text{Order 2} \\ \Phi_5(\xi) = P_1^{(0)}(\xi)P_2^{(2)}(\xi) & \\ \Phi_6(\xi) = P_1^{(2)}(\xi)P_2^{(1)}(\xi) & \\ \Phi_7(\xi) = P_1^{(1)}(\xi)P_2^{(2)}(\xi) & \text{Order 3} \\ \Phi_8(\xi) = P_1^{(3)}(\xi)P_2^{(0)}(\xi) & \\ \Phi_9(\xi) = P_1^{(0)}(\xi)P_2^{(3)}(\xi) & \end{array}}^{\Psi_3(\xi)}$$

Figure A.1: Third order example of multivariate aPC basis.