Constrained Energy Optimization in Heterogeneous Platforms using Generalized Scaling Models

by

Ujjwal Gupta

A Thesis Presented in Partial Fulfillment
of the Requirement for the Degree
Master of Science

Approved July 2014 by the
Graduate Supervisory Committee:

Umit Y. Ogras, Chair
Sule Ozev
Chaitali Chakrabarti

ARIZONA STATE UNIVERSITY

August 2014

ABSTRACT

Mobile platforms are becoming highly heterogeneous by combining a powerful multiprocessor system-on-chip (MpSoC) with numerous resources including display, memory, power management IC (PMIC), battery and wireless modems into a compact package. Furthermore, the MpSoC itself is a heterogeneous resource that integrates many processing elements such as CPU cores, GPU, video, image, and audio processors. As a result, optimization approaches targeting mobile computing needs to consider the platform at various levels of granularity.

Platform energy consumption and responsiveness are two major considerations for mobile systems since they determine the battery life and user satisfaction, respectively. In this work, the models for power consumption, response time, and energy consumption of heterogeneous mobile platforms are presented. Then, these models are used to optimize the energy consumption of baseline platforms under power, response time, and temperature constraints with and without introducing new resources. It is shown, the optimal design choices depend on dynamic power management algorithm, and adding new resources is more energy efficient than scaling existing resources alone. The framework is verified through actual experiments on Qualcomm Snapdragon 800 based tablet MDP/T. Furthermore, usage of the framework at both design and runtime optimization is also presented.

Dedicated to my parents

ACKNOWLEDGEMENTS

The thesis represents a step towards the goal of unifying computing at various levels. There have been several people who have directly or indirectly contributed to this work.

My deepest gratitude is to my advisor, Dr. Umit Ogras, for the patience, advice and guidance he offered me throughout the last year. This thesis would not be possible without his support. I would also like to thank Dr. Sule Ozev and Dr. Chaitali Chakrabarti for taking out time and being in my thesis defense committee.

I would like to thank Navyasree Matturu, Spurthi Korrapati and Sanat Kumar Panda for helping conduct the experiments on Qualcomm's Snapdragon 800 MDP/T. Their work helped accelerate the hardware validation of the proposed framework at least $2\times$ faster. Gaurav Singhla, Sankalp Jain and Jaspreet Kaur Sahota helped in the initial stages of board bring-up for measurements as well. I would also like to thank Dr. Karam S. Chatha for providing the idea of adding power constraint to the proposed framework.

My brother Dhruv and friend Ayush have been instrumental in providing feedback on this work as well. My parents, Udai, Rashmi, and my girlfriend Ramandeep deserve most of the credit for this work. I would like to thank them for their love, endless support and understanding even though I am thousands of miles away.

My master's term at Arizona State University provided an environment that made my studies enjoyable and more productive. I want to especially thank Dr. Martin Reisslein, whose course on Multimedia/ QoS gave me a teasing flavour of research and Dr. Armando Antonio Rodriguez whose teaching methodology inspired me to stay associated with academia. I would also like to thank Dr. Sarma Vrudhula and Kyle Gilsdorf for providing financial support during early semesters of my masters study and giving me many opportunities to teach in classroom.

TABLE OF CONTENTS

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

Figure Page

Chapter 1

INTRODUCTION

Increasing computational power at mobile form factor and new heterogeneous resources are leading to emerging applications. These applications, in turn, drive the evolution of embedded systems by demanding more energy efficiency and performance. Consequently, mobile devices have evolved from being basic feature phones to smart-phones/tablets and have taken the computing market by storm. The market push is towards integrating as much functionality as possible. For example, today's smart-phones are able to work as a music player, video game, internet hot-spot, internet browsing device, GPS, money (NFC), data storage, calender, personal digital assistant, and finally a phone. Due to a mixed view of an embedded system and a general purpose computer, they have evolved into highly heterogenous computing platforms as described in Figure 1.1.

As opposed to the traditional PC and server systems, the role of CPU is replaced with a multiprocessor system-on-a-chip (MpSoC) which integrates many processing elements (PEs). What is more, the application processor is just one of the many components in the device. Other major resources include display, flash memory, DRAM, baseband and radio frequency chips, power management IC, voltage regulators, camera, touch panel and battery. Application processor does not dominate the platform neither in terms of cost [44] nor power consumption [10]. In particular, display consumes a significant portion of power consumption ($\sim$30%) across a wide range of application use cases. Application use cases simply refer to different scenarios, such as phone talk, video playback, simultaneous browsing and audio playback under which the platform is being used. Different scenarios of interest are referred to as Key Performance Indicators (KPIs) and power/performance/thermal behaviour of the mobile devices under tens of KPIs are tracked by device manufacturers

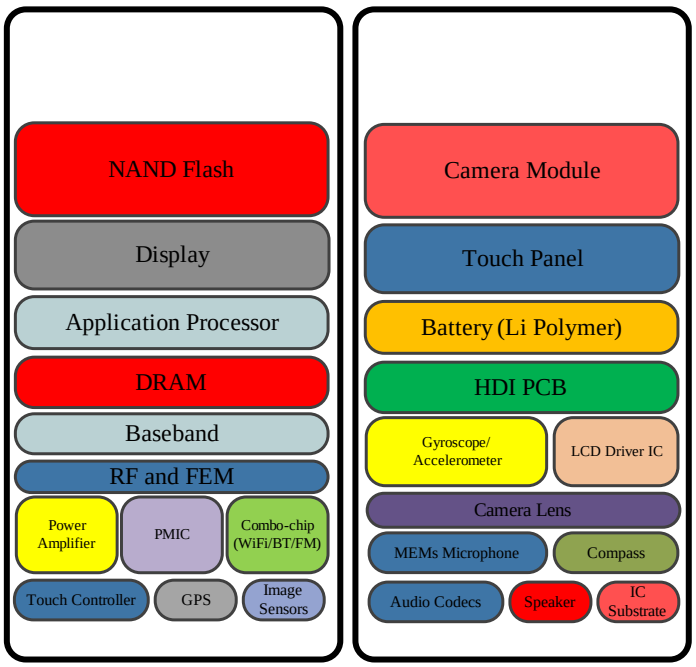


Figure 1.1: Mobile Platform: A Heterogenous Device Consisting of Several Components/Resources [44]. Application Processor is Only One of the Resources Not Dominating in Cost or Power.

for competitiveness. Furthermore, the key quality metrics such as power consumption and responsiveness of the whole device is not anymore a function of application processor and memory only due to the high degree of heterogeneity. A variety of resources such as camera module, GPS, Long-Term Evolution (LTE) modules can be the key resource that determine the user experience for different KPIs.

We also note that the application processor itself is also a heterogenous resource, as shown in Figure 1.2. While still orchestrating the operation of other PEs, the CPU cores neither dominate the power consumption, nor determine the performance under many application scenarios [10]. Modern MpSoCs house close to hundreds of unique PEs from multiple vendors integrated on a single chips. These PEs range from GPUs, which almost take half of the silicon real estate, to tiny PEs used for processing sensor data such as

gyroscope readings. Notable PEs found in most of the mobile processors include image signal processors, display processing engine, video encoder and decoders, digital signal processors, memory controllers, security engines and on-chip communication networks that interconnect the resources. As a result, research focus has to be broadened accordingly to pay attention to the *platform as a whole* rather than focusing on a subset. As the first step towards this goal, we present a resource centric view which expose the impact of individual resources by modelling this heterogeneity explicitly.

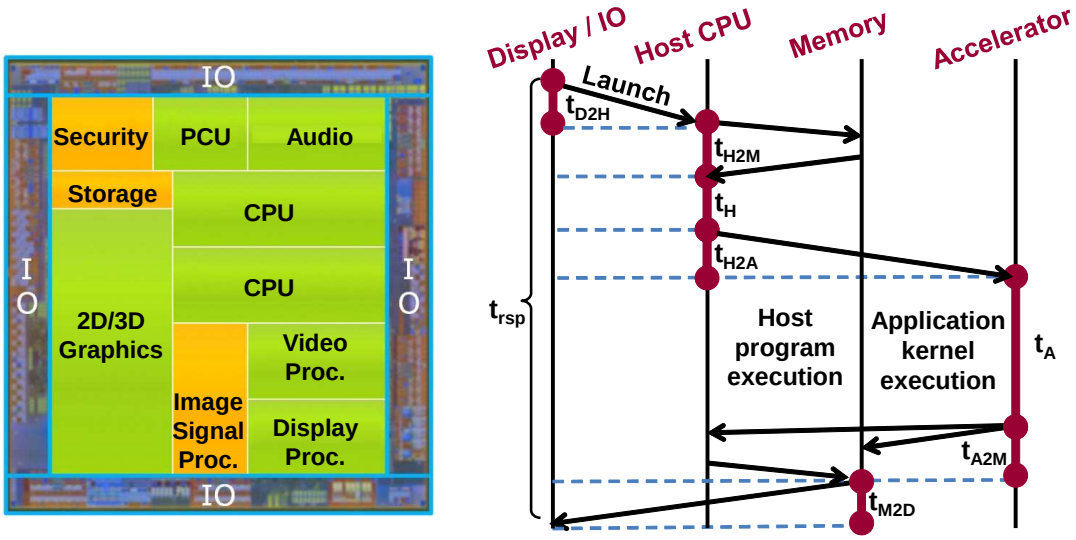


Figure 1.2: (a) Intel© Atom™ Processor $Z2760$ [32], (b) Illustration of the Resource Centric View.

1.1 Resource-Centric View of Mobile Devices

Despite the richness of the underlying platform only a subset of the resources are invoked during the lifetime of an application. For example, a navigation application goes through a number of phases as illustrated in Figure 1.2(b). After the user launches the application, the core running the OS fetches the host code from the memory and starts execution. Next, it triggers the accelerator (the GPS module), which loads the data from the memory, performs the assigned task, and writes the results back. Hence, the run-time and power con-

sumption of different application phases depend on the particular resource employed, while the synchronization overhead is determined by the communication network and I/O speed. Furthermore, the latency to reach the memory through the inteconnection network and memory controller, as well as the memory access time are additive to the total time. Hence, the response time for the GPS example can be obtained as $t_{\mathrm{rsp}} = t_{\mathrm{CPU}} + t_{\mathrm{comm}} + t_{\mathrm{mem}} + t_{\mathrm{GPS}}$ where the terms represent the time contributed by the CPU, communication, memory and GPS module. Likewise, power consumption is determined by the active resources during the lifetime of the application and their particular power states. Hence, the ability of the dynamic power management algorithms to put different PEs to sleep or other low power states has a significant impact on the total power consumption. In this work, we employ this resource centric view in two dual energy optimization formulations. The first one aims at minimizing the total energy drawn from the battery with a constraint on the response time of the user. This relies on the fact that as long as the response time is in the order of miliseconds, the users cannot distinguish the responsiveness. The other formulation minimizes the energy consumption with a constraint on the power consumption. This formulation aims at making best use of a fixed power budget which is given by thermal constraints.

1.2 Do Not Overlook the Background Power

While most of the inactive resources are put to sleep states when not in use, there are shared resources such as the display and PMIC which do not directly contribute to the application performance but remain powered on during the application lifetime. If these shared resources can be put in lower power states right after executing an application, it may make sense to speed up the execution to save overall energy. On the other hand, if there are other active applications which will keep the shared resources active anyway, speeding up will not necessarily reduce the platform energy. Hence, the optimality results depend critically on other platform resources and power management decisions.

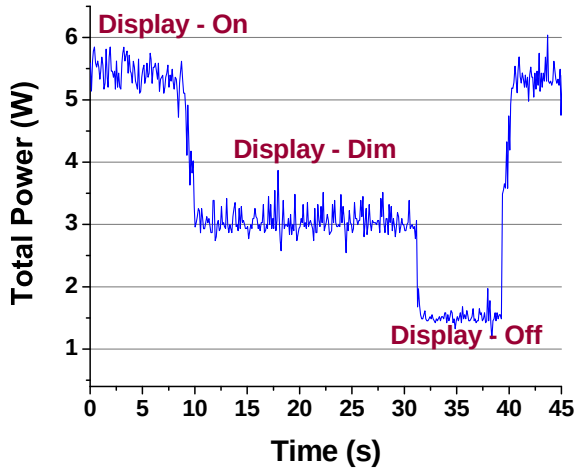


Figure 1.3: An Example of Total Power Consumed in a Qualcomm Snapdragon 800 MDP Tablet for Full Brightness, Dimmed and Off Display.

Significance of the background power can be understood better with the help of Figure 1.3, which shows the total power consumption of a mobile development platform when LCD display is on, dimmed and off. This plot clearly shows that the difference in total platform power when the display is on and off is about 4 W, which is significant given that the total power is less than 6W. We note that this measurement is performed at full brightness for a tablet LCD display. However, even when we dim the LCD brightness, which would mean a significant degradation in user experience, or use a mobile smartphone with a smaller display, the display power is still as large as the power of the rest of the components in the whole phone combined. When we add the power consumption of other common resources such as power management IC, voltage regulators, background power clearly becomes even more significant. Therefore, it is imperative to consider the background power during the optimization process.

1.3 Key Contributions

This thesis presents an energy optimization framework for heterogeneous mobile platforms. This framework enables reasoning about optimization both at the MpSoC and platform level by exposing the impact of each resource on power, performance and energy. It can be used for platform exploration to determine the type and characteristics of resources that should be employed and for finding the optimal operating conditions of a given hardware configuration. More precisely, the proposed framework can be used to optimize power/performance trade-off for each resource independently or co-optimize a subset of resources together. Furthermore, it can be used to *quantify* the energy savings that can be obtained by adding new resources or replacing existing ones with more energy efficient resources.

Our results not only confirm that voltage and frequency scaling alone offers limited improvements in energy efficiency due to the inverse relationship between power consumption and performance, but also provide *precise numeric evaluations*. Furthermore, we enable co-optimization of the platform resources at once and demonstrate that co-optimization is superior to optimizing the resources one-by-one. The proposed framework can also be used to evaluate the benefits of adding new CPU cores and an application-specific accelerators. We demonstrate the benefits of adding new CPU cores or an accelerator to an existing system using two mobile applications.

The second major contribution of this thesis is the development of an experimental setup, a methodology for collecting reliable experimental data and extensive set of experiments using a state-of-art mobile development board and real-life applications. Given the complexity of the state-of-art platforms and applications processors, it is extremely hard to follow a simulation methodology for accurate power and performance modeling. On one hand, existing simulation frameworks model only CPU cores, GPU, caches and mem-

ory [7, 36, 8]. Developing simulation models with accurate timing and power information for the rest of the models such as image signal processors, video/audio engines, touch panel is a daunting task. On the hand, even if we assume the availability of these models, running real mobile apps for a representative amount of time would require enormous amount of time and effort. This would require a complete virtual platform that can run the applications for minutes and capture the thermal throttling behaviour, which is critical for mobile platforms [5, 12]. The conflicting requirements on the accuracy and run-time make this task extremely difficult. As a result, it is crucial to be able to utilize the hardware development boards and existing commercial systems to validate the analytical models and optimization frameworks such as presented in this thesis. There is a wide spectrum of work done in these areas, but to the best of our knowledge we are the first to have integrated and experimentally proved the scaling models used for architectural exploration with dynamic power management [29, 2, 28, 33, 22, 58, 65, 16, 64, 39, 15]. To this end, the detailed experimental procedure developed in this work does not only enable us to validate our results but also will benefit research on the design and optimization of mobile platforms.

1.4 Thesis Organization

The rest of this thesis is organized as follows. In Chapter 2, we discuss the related research which involves power measurement using sampling and event driven approaches, modelling of power and performance, architecture exploration, and dynamic power management. In Chapter 3, we start by introducing the performance model, then we show that it is a generalization of Amdahl's law. We then formulate the power and energy models, respectively. After this, we formulate the energy optimization problem and present the solution to two dual formulations. Chapter 3 also presents an illustrative example to explain the importance of background power theoretically and discusses how to use our framework at design and runtime. Chapter 4 describes the host platform used to obtain the experimen-

tal data. We also provide detailed descriptions of benchmarks employed in this work and explain how power, time and energy are calculated using our experimental setup. Chapter 5 starts with the evaluation of our models using Matlab with the input data from [50]. Then, we present our own experiments using Qualcomm's Snapdragon 800 MDP and validate the power and performance models as well as the energy optimization framework. Finally, Chapter 6 presents the future work and conclusion of this work. This thesis also includes Appendix A to provide detailed step-by-step instructions of the measurement methodology developed in this work.

Chapter 2

RELATED WORK

Power consumption has been one of the major design consideration for many years [40]. This has led to the study of energy efficient techniques to harness the processing power within the power and thermal budgets [14, 19, 62]. For example, Vallina-Rodriguez and Crowcroft studied energy efficiency in mobile space by surveying software solutions to achieve energy efficiency from the year 1999 to May, 2011 [62]. They classified the research in six different levels: energy-aware operating systems, efficient resource management, the impact of users' interaction patterns with mobile devices and applications, wireless interfaces, sensors management, and integrating mobile devices with cloud computing services. They concluded that mobile resources need to be managed from an energy-efficient perspective *without diminishing the user experience*. This is an important design constraint for hardware and software engineers alike.

With the explosive growth in number of smart-phone users [20] and limited battery life, energy efficiency is an extremely critical metric. There are several challenges for minimizing energy consumption of smart-phones, a) increasing computational power along with sensing, storage and communication capabilities have open up many power hungry application domains, b) the usage pattern in smart-phones vary extensively, c) mobile devices are fanless, and are subject to tight surface, skin, temperature constraints which limit the peak power consumption, as the skin temperature affects the user experience. [61, 45]. Furthermore, one may have to make energy management decisions at different granularity levels (platform or MpSoC). For example, Youm Huh discusses the importance of power management at two different hierarchial levels (processor design level and PMIC power analog circuit level), which signifies the future direction of power management in mobile

devices [31]. This motivated us to create a framework which takes into account all the components of a mobile platform.

To address energy in computing systems, multiple problems like, modelling, design, management and run-time task scheduling need to be solved. Modelling deals with developing techniques for predicting key metrics such as power and performance. Important contributions in measurement and profiling of energy have been made by Carroll and Heiser [10], Pathak et al. [51, 50], Yoon et al. [68] and recently by Kim et al. [35]. Carroll and Heiser measured the energy and power for different resources on a mobile system for several common mobile applications. They measured power by directly inserting sense resistors on power supply rails of the relevant resources and measured voltage and current to get power. The most important contribution of their work was to show that many resources like display, GSM and Graphics can dominate the power consumption in the smart-phone. The energy consumed in resources other than CPU is extremely important and can not be ignored. Pathak et al. [50, 51] developed an energy profiling tool called Eprof and provided important insight into use of wakelocks in applications. Their approach was mainly from a software perspective on how a typical mobile application consumes power and discussed some key design points for application designers keeping hardware in perspective. They also provided insight on asynchronous power behaviour and affirms that power consumed by each I/O component is often comparable to or higher than that of CPU. Other techniques on fine grained power measurement were developed by Yoon et al. [68] and Kim et al. [35]. Out of all the measurement approaches mentioned above, Kim et al. [35] claim to have the least data measurement overhead by implementing their technique at the lowest kernel level. Furthermore, they state that the sampling based approaches are not good for power measurement of each resource since the overhead becomes large as the sampling frequency increases. On the other hand, a pure event driven approach is also not good, as there is a chance of missing certain events that occur in parallel. In our work, we are using a

proprietary energy profiling tool provided by Qualcomm Incorporated. Since it is an industrial tool developed specifically for the platform we are using, the measurement accuracy is expected to have industrial quality. For the experimental evaluation of our generalized power model we use a polynomial function of order 3, such a model has been widely used in literature, in particular we find a similar fitting approach in [34, 55].

Design-time techniques for optimization using power, performance and temperature have been an active research area. For example, a comprehensive design methodology for multiprocessor SoC using a component based approach has been presented in Cesario et al. [13]. An overview of design challenges faced by MpSoC designers, emphasizing the need for re-configurable programming models on application layer, which can run on different architectures is provided in Martin [37]. A static temperature aware processor frequency assignment for MpSoCs using convex optimization is presented in Murali et al. [41].

Dynamic approaches, on the other hand, deal with changes in voltage, frequency (DVFS) and sleep state of the resources. The work in [46] minimizes the energy consumption subject to performance constraint using DVFS with island partitioning on Network-on-Chip. Work by Ayoub et al. [4] focusses on performance constrained power minimization at OS-level in general purpose systems with a new DVFS algorithm. An optimal control approach to power management for multi-voltage and frequency island multiprocessor platforms under highly variable workloads is presented in Bogdan et al. [9]. In relation to these, the proposed framework in this thesis, provides optimal operating point for each applications. Hence, it can be used as a target point in dynamic algorithm.

Besides DVFS, one can also control scheduling of the tasks to the processing elements. Latency and resource constrained low-power task scheduling is discussed in Shiue and Chakrabarti [59] for resources operating at multiple voltages. A system wide dynamic task scheduling algorithm for energy efficiency is presented by Zhuo and Chakrabarti [69],

this work focusses on finding optimal scaling factor by which a task should be scaled while minimizing energy without any deadline constraints. In particular, we theoretically give the reasoning about the observations made by them on the relative values of CPU power and device power effects on energy efficiency. Nithi and Lind [43] implement a power-aware task monitor and scheduler to increase battery life by preserving it for more energy intensive tasks. Battery life extender tool developed by Metri et. al. [38] enables the reconguration of mobile devices in order to utilize only the resources required for specific tasks. It also provides an estimate of the impact of applications on the overall battery life. Other techniques such as scheduling the tasks in applications to maximize idle time of resources such that they can sleep and offloading computational intensive tasks to the cloud have been suggested in [60, 56, 18]. Energy management using hybrid techniques like DVS and DPM are discussed in Zhuo et al. [70]. High-level power management of embedded systems based on energy cost functions which are task specific have been proposed by Cho et al. [17].

There are also new approaches that are extending Amdahls Law to multicore computers [29, 2, 28, 33, 22, 58, 65] and energy consumption modeling [16, 64, 39, 15]. These approaches enable making prediction about the performance and power as architectures scale, e.g., Cho and Melhem [16] using Amdahls law determine the optimal processor frequencies in the serial and parallel regions with the goal of minimizing the total energy consumption. The power and performance models proposed in this thesis can be employed by these approaches and make them stronger. We refer the reader to a recent survey [1] which has an exhaustive list of studies on architecture exploration of multicore systems. The performance model in our optimization framework resembles Amdahl's law for multicore architectures. In particular, it is similar to the studies presented in Cassidy and Andreou [11] and Zidenberg et al. [71] in generalizing serial and parallel instruction sections to an arbitrary number of resource types. Unlike these studies, we develop a general power

model that captures different power states of PEs and takes Dynamic Power Management (DPM) decisions into account. Furthermore, we focus on heterogeneous mobile platforms and minimize the total energy consumption to maximize the battery lifetime with a constraint on maximum response time (or minimum throughput), since improving performance beyond user perception is not necessary.

Chapter 3

ENERGY OPTIMIZATION FRAMEWORK FORMULATION

In this chapter, we first present the performance, power and energy models for mobile platforms and validation of these models. Then, we formulate the constrained energy optimization framework constructed using these models outline the solution to the optimization problem. Finally, the chapter concludes by describing how the proposed framework can be utilized by dynamic management algorithms.

3.1 Performance Model

We use the response time and application throughput under a set of target applications as the primary performance metrics. Suppose that M out of N resources in the platform contribute to the total response time, and computation consists of one consolidated serial and C different concurrent phases. The duration of the serial phase can be written as $t_R = \sum_{i=0}^{R-1} t_i$, where t_i is the time spent by i^{th} resource and R is the number of resources which contribute to the serial phase. On the other hand, the duration of *each* parallel phase is given by the longest task in that phase rather than a linear sum, i.e., the duration of the k^{th} parallel phase can be written as $t_k = \max_{0 \le j \le N_k} t_j$, where N_k is the number of tasks in the k^{th} parallel phase. Then, the response time of the baseline hardware configuration can be expressed as $t_{\text{base}} = \sum_{i=0}^{R-1} t_i + \sum_{k=0}^{C-1} t_k$. One can vary the base performance by making *design* or *run-time* changes to any of the resources. For example, by doubling the speed of the i^{th} resource in the serial phase using a scaling factor of $s_i = 2$, its contribution to the response time can be halved. Hence, the response time can be expressed as,

$$\boxed{t(\boldsymbol{s}) = \sum_{i=0}^{R-1} \frac{t_i}{s_i} + \sum_{k=0}^{C-1} \max_{0 \le j \le N_k} \frac{t_{k,j}}{s_{k,j}},\ \forall s_i > 0,\ \forall s_{k,j} > 0} \tag{3.1}$$

Note that $s_i > 1$ implies that the i^{th} resource is running faster with respect to its baseline performance. This speed-up could be simply due to running at a higher frequency or any performance optimization. Likewise, $s_i < 1$ means that the i^{th} resource is running slower than baseline configuration.

3.1.1 Speed-up: Amdahl's Law Generalization

This subsection describes how the speed-up ($t_{\text{base}}/t(\boldsymbol{s})$) generalized Amdahl's Law.

Corollary 3.1.1 *In analogy to Amdahl's Law, which divides the execution time into sequential and parallel parts, our formulation reveals the contribution from multiple resources. It can be shown that $t_{\text{base}}/t(\boldsymbol{s})$ is a generalization of Amdahl's Law [2, 66, 11]. Hence, Equation 3.1 enables us to identify the resources with highest performance impact.*

Definition: We define the fraction, f_i, as the portion of time spent by computation or communication in i^{th} resource to the total response time before scaling.

$$f_i = \frac{t_i}{t_{\text{base}}} \tag{3.2}$$

Example: Consider a simple scenario where only the host CPU, H has to be scaled. Response time before and after scaling will be

$$t_{Total}^{before} = t_H + t_{Others} \tag{3.3}$$

$$t_{Total}^{after} = \frac{t_H}{s} + t_{Others} \tag{3.4}$$

The fraction of time spent in host CPU w.r.t. other resources is

$$f_H = \frac{t_H}{(t_H + t_{Others})} \tag{3.5}$$

The speed-up is given by fraction of total time without scaling and new or improved total time with scaling.

$$Speed - up = \frac{Total\ time\ without\ scaling}{Scaled\ total\ time}$$

$$= \frac{t_{Total}^{before}}{t_{Total}^{after}}$$

$$= \frac{t_H^{before} + t_{Others}}{\frac{t_H^{before}}{s} + t_{Others}} \tag{3.6}$$

Dividing the numerator and denominator by $T_H + T_{Others}$ gives,

$$Speed - up = \frac{1}{\frac{T_H}{s \times (T_H + T_{Others})} + \frac{T_{Others}}{T_H + T_{Others}}} \tag{3.7}$$

Substituting for f_H we get,

$$Speed - up = \frac{1}{\frac{f_H}{s} + (1 - f_H)} \tag{3.8}$$

Equation 3.8 looks exactly like Amdahl's law but with important distinctions. First, the speed-up can be obtained by an arbitrary method such as improved micro-architecture, a superior technology node, or by dynamic voltage-frequency scaling. Second, we have defined the fraction, f_i as the time fraction instead of normalized application dependent parallelizable code fraction.

Suppose there are more than one resources being scaled, e.g., host CPU, H and some accelerator, A. The accelerator can be a GPU, vector unit, memory, other CPUs, etc. Let s_H be the scaling factor for host CPU and s_A be the scaling factor for an accelerator.

$$t_{Total} = t_H + t_A + t_{Others} \tag{3.9}$$

The fraction of total time and time spent by computation in host and accelerator be f_H and f_A respectively. The speed-up using Equation 3.6 is, given as,

$$Speed - up = \frac{t_H + t_A + t_{Others}}{\frac{t_H}{s_H} + \frac{t_A}{s_A} + t_{Others}} \tag{3.10}$$

Dividing the numerator and denominator by $t_H + t_A + t_{Others}$ we get,

$$Speed - up = \frac{1}{\frac{t_H}{s_H \times (t_H + t_A + t_{Others})} + \frac{t_A}{s_A \times (t_H + t_A + t_{Others})} + \frac{t_{Others}}{(t_H + t_A + t_{Others})}} \tag{3.11}$$

$$Speed - up = \frac{1}{\frac{f_H}{s_H} + \frac{f_A}{s_A} + (1 - f_H - f_A)} \tag{3.12}$$

In general, for $R - 1$ resource speed-up we can write,

$$\boxed{Speed - up = \frac{1}{\sum_{i=0}^{R-1}(\frac{f_i}{s_i}) + (1 - \sum_{i=0}^{R-1} f_i)}} \tag{3.13}$$

Application Throughput: Throughput is a more suitable metric for applications with periodic tasks. For example, throughput in *frames/sec* is commonly used for video encoding/decoding applications. The critical period for the application, e.g., the time to encode/decode a single frame can be expressed in a way similar to Equation 3.1. Then, number of frames or bits divided by the period can be used to express the application throughput as follows.

$$B(\boldsymbol{s}) = \frac{1\ Frame}{t(\boldsymbol{s})} = \frac{1\ Frame}{\sum_{i=0}^{R-1} \frac{t_i}{s_i} + \sum_{k=0}^{C-1} \max_{0 \le j \le N_k} \frac{t_{k,j}}{s_{k,j}}},\ \forall s_i > 0,\ \forall s_{k,j} > 0 \tag{3.14}$$

3.2 Validation of Performance Model

We used a mobile platform development board [52] and a set of benchmarks [25, 49, 26] to make response time measurements. While the details of the experimental platform are described later in Chapter 4, here we show how they are used to validate our performance model.

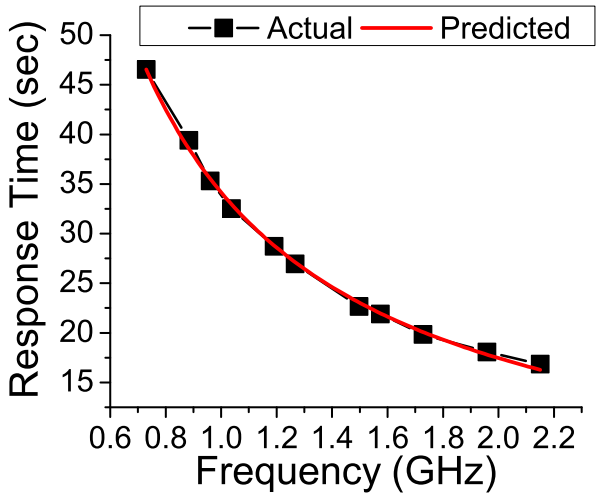


Figure 3.1: Result for Response Time Fitting with the Proposed Performance Model for JPEG Benchmark Running on 4 CPU Cores. A Good Fit with Adjusted-R^2 Value of 0.997 is Obtained.

We first set the frequency of the cores in the platform to a fixed value and ran the benchmark. During this run, we profiled the CPU time and the real time taken by the benchmark. In other words, we measured t_H and t_{Others} in Equation 3.3.

Then, we used Origin software [47] to fit the measured data to our performance model given in Equation 3.4. Note that, the scaling factor, $s = \frac{f_{new}}{f_{base}}$. The measured and predicted values for the JPEG benchmark when all four cores and only one core were active are shown in Figures 3.1 and 3.2, respectively. As Figures 3.1 and 3.2 clearly demonstrate, our model provides an accurate fit to the measured response time. More precisely, the Adjusted R-Square after fitting for four cores case is found as 0.997, while the mean squared error in t_H and t_{Others} is 0.41 and 0.46 respectively. The Adjusted R-Square after fitting for one core case is found as 0.992, while the mean squared error in t_H and t_{Others} is 2.21 and 2.48 respectively.

We also observe that the one core case in Figure 3.2, exhibit bigger mismatch between theory and measurement. The bigger different stems from the fact that the operating system running in the same core as the benchmark. In this case, the OS threads have a bigger

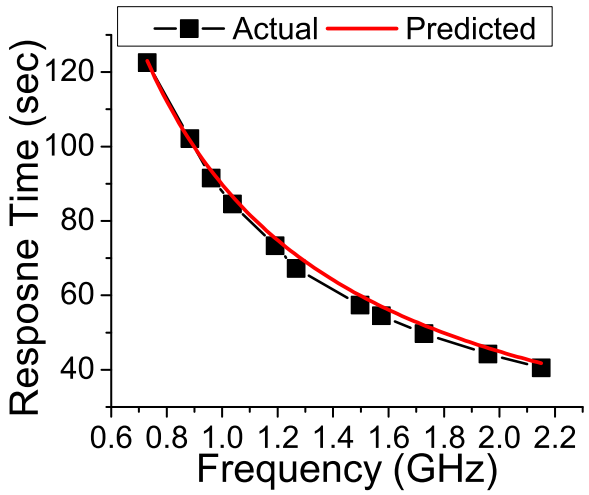


Figure 3.2: Result for Response Time Fitting with the Proposed Performance Model for JPEG Benchmark Running on 1 CPU Core. A Good Fit with Adjusted-R^2 Value of 0.992 is Obtained.

impact on the response time. This observation reinforces the importance of experiments with real platforms and applications. Capturing the impact of the OS (in this case Android OS) is extremely hard in simulation frameworks.

Our models are validated using many other benchmarks. The graph for other benchmarks are presented in Chapter 5 together with the validation of the overall framework.

3.3 Power and Energy Models

Each resource can be in a variety of power states ranging from simple *off* and *on* states to a complex set of sleep and active states. Let $\pi_{i,j}$ show the fraction of time that i^{th} resource is in j^{th} power state and the corresponding distribution be $\pi_i = [\pi_{i,0}, ..., \pi_{i,S}]$, where S is the number of power states. Likewise, $P_{i,j}$ is the power consumption of resource i in state j, and is affected by the performance scaling factors, $\boldsymbol{s}$ used in Equation 3.1. Some portion of the power consumption stays constant regardless of s_i (e.g., leakage power of untouched resources), some portion will change linearly (e.g., leakage power due to area scaling), and some portion will be proportional to s_i^{α} (e.g., dynamic power due to voltage/frequency

scaling, $\alpha \geq 2$, we use $\alpha = 2.5$ [67]). If we denote these components as $H(0)_i$, $H(1)_i$ and $H(2)_i$, the scaled power consumption of the i^{th} resource in j^{th} power state can be written as,

$$P_{i,j}(s_i) = H(0)_{i,j} + s_i H(1)_{i,j} + s_i^{\alpha} H(2)_{i,j} \tag{3.15}$$

Also there might be cases when we want to vary variable(s) which causes cubic change in some portion of power as well as quadratic change in some portion. E.g., If we take frequency as the scaling variable then for some processors like Snapdragon the voltage varies linearly with frequency, e.g., $V = m \times f + n$ and hence we can write power as,

$$P = s^3.H(3) + s^2.H(2) + s^1.H(1) + H(0) \tag{3.16}$$

Since we observe that we have two higher order terms in Equation 3.16, to make our power model capture any such general scaling trends we propose the following model if there are a total of nP power components,

$$P_{i,j}(s_i) = s_i^{\alpha(0)} H(0)_{i,j} + s_i^{\alpha(1)} H(1)_{i,j} + s_i^{\alpha(2)} H(2)_{i,j} + \cdots + s_i^{\alpha(nP)} H(nP)_{i,j} \tag{3.17}$$

$$P_{i,j}(s_i) = \sum_{k=0}^{nP-1} s_i^{\alpha(k)} H(k)_{i,j} \tag{3.18}$$

Please note that in Equation 3.17 $\alpha(0) = 0$, $\alpha(1) = 1$, and other higher order $\alpha(k)$ terms are ≥ 2 and not limited to integers.

Then, the average power consumption of the i^{th} resource can be expressed as $P_i(s_i) = [P_{i,0}(s_i), ..., P_{i,S}(s_i)] \times \pi_i^T$. On the other hand, the total power consumption of the background resources that do not contribute to the response time is denoted by P_{BG}. Then, the overall power consumption can be obtained as $P(\boldsymbol{s}) = \sum_{i=0}^{M-1} P_i(s_i) + P_{\text{BG}}$. Finally, the total energy consumption is:

$$E(\boldsymbol{s}) = \left(\sum_{i=0}^{R-1} \frac{t_i}{s_i} + \sum_{k=0}^{C-1} \max_{0 \leq j \leq N_k} \frac{t_{k,j}}{s_{k,j}} \right) . \left(\sum_{i=0}^{M-1} P_i(s_i) + u.P_{\text{BG}} \right) \tag{3.19}$$

Note that, we multiply the background power with a decision variable $u = \{0, 1\}$ to enable including or excluding P_{BG} in the objective function. When $u = 1$, the longer the response time, the higher the total platform energy will be. Likewise, power consumption of the scaled resources are multiplied by the response time to obtain the total energy consumption, while the effect of entering low power states is captured by the power state distribution π_i.

3.4 Validation of Power Model

Similar to the performance validation, we used the same mobile platform development board [52] and benchmarks [25, 49, 26] to make power measurements. In what follows we provide the validation results obtained using the JPEG application, while complete results are left to Chapter 4, as in the performance validation.

We first set the frequency of the cores in the platform to a fixed value and ran the JPEG benchmark. During this run, we profiled the CPU power and the total power taken by the benchmark. The leakage power was also estimated to be about 40% of the total CPU power. In other words, we measured the coefficients $H[3 : 0]$ in Equation 3.16. Then, we used Origin software [47] to fit the measured data to our power model given in Equation 3.16. Note that, the scaling factor, $s = \frac{f_{new}}{f_{base}}$. Since, the voltage is also a linear function of frequency, we can either add it into the model or give it separately as a variable. We find that using the voltage values directly is better as the solution takes less time to fit due to lower complexity (cubic vs. quadratic). Also, this means that we had to supply two input variables (voltage and frequency) and not just one variable (frequency) like in response time fitting, shown in Section 3.2.

The measured and predicted values for the JPEG benchmark when all four cores and only one core were active are shown in Figures 3.3 and 3.4, respectively. These figures clearly demonstrate, our models provides an accurate fit. More precisely, the Adjusted R-

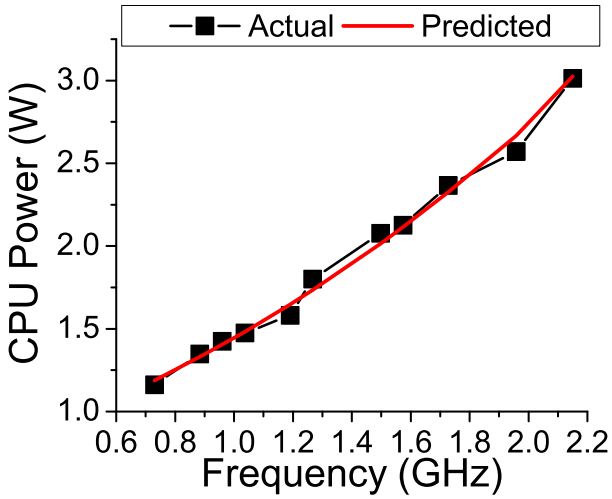


Figure 3.3: Result for Power Fitting (Excluding Background Power) with the Proposed Power Model for JPEG Benchmark Running on 4 CPU Cores. A Good Fit with Adjusted-R^2 Value of 0.991 is Obtained.

Square after fitting for four cores case is found as 0.990, while the mean squared error is 3.581×10^{-2}. The Adjusted R-Square after fitting for one core case is found as 0.933, while the mean squared error is 05.094×10^{-2}. We also observe that the one core case in Figure 3.4, exhibit bigger mismatch between theory and measurement. This is because of the operating system running in the same core as the benchmark, as well as, measurement error.

Our models are validated using many other benchmarks. The graph for other benchmarks are presented in Chapter 5 together with the validation of the overall framework.

CPU Voltage Fitting: As mentioned in Section 3.3, Snapdragon has linear relationship between voltage and frequency. The actual voltage values are proprietary and not made available. However, we found voltage values for the chip used in our board from operating system files, as shown in Table 3.1.

We obtain a fit for the voltage values as a function of frequency, $V = m \times f + n$ and get a nearly perfect linear relationship as shown in Figure 3.5. We find the Adjusted R-Square

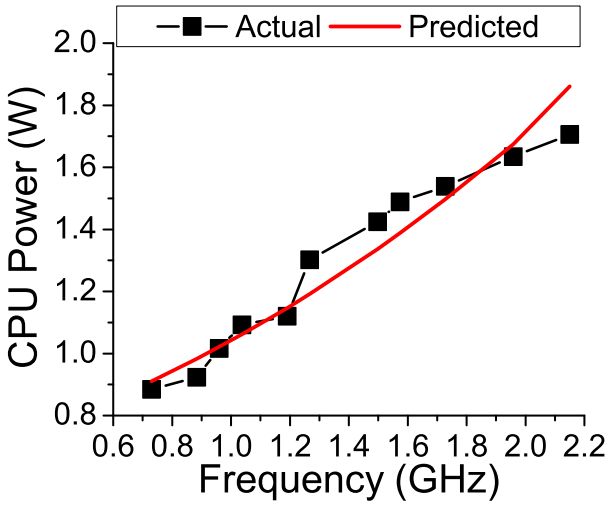


Figure 3.4: Result for Power Fitting (Excluding Background Power) with the Proposed Power Model for JPEG Benchmark Running on 1 CPU Core. A Fairly Good Fit with Adjusted-R^2 Value of 0.933 is Obtained.

after fitting to be 0.997 and mean squared error in m and n to be 2.44×10^{-9} and 3.7×10^{-3}, respectively. Researchers, who do not have this information can benefit from these fitted values, e.g., the approach presented in [34] uses a linear model for voltage and directly taken our predicted voltage values, instead of conducting the regression again. Note that the linear relationship is almost perfect for frequency range, 0.9 GHz to 2 GHz. Beyond this range, the linear model may not fit. For example, the lower frequencies (< 0.7 GHz) are usually at constant voltage to keep leakage current low, and thus do not scale linearly.

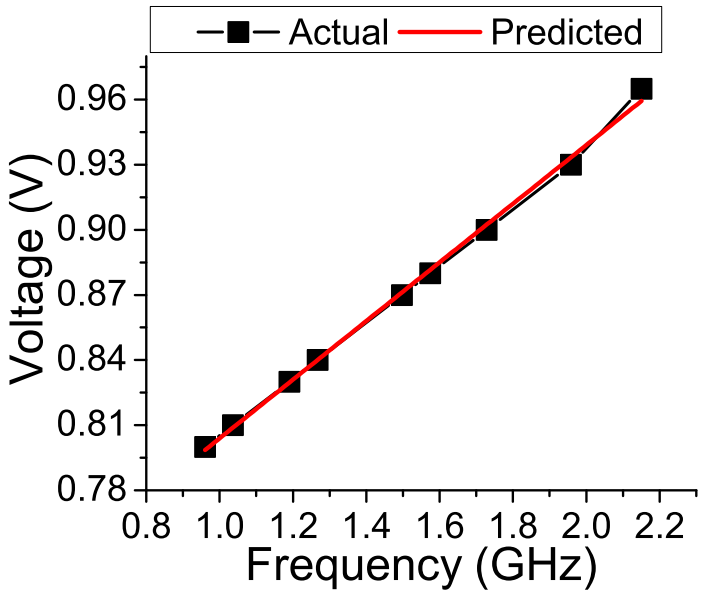


Figure 3.5: Result for Voltage Fitting With a Linear Model. A Good Fit with Adjusted-R^2 Value of 0.997 is Obtained.

Table 3.1: Voltage and Frequency Table for Snapdragon 800 Processor. Each Row Corresponds to the Possible Frequency Value for the Processor and its Voltage.

Frequency (MHz)	**Voltage (V)**
960.00	0.800
1036.80	0.810
1190.40	0.830
1267.20	0.840
1497.60	0.870
1574.40	0.880
1728.00	0.900
1958.40	0.930
2150.40	0.965

3.5 Solution of the Optimization Problem

The optimization goal is minimizing the total energy consumption under a response time and/or power constraint. The reason for a response time constraint is clear, there is no need to increase the performance of an application beyond user perception. Similarly, many mobile applications do not require sustained performance, they involve short bursts of computation with occasional user activity [54], this makes power to be a major constraint in mobile systems. This optimization goal can be expressed using a nonlinear optimization problem with inequality constraints:

$$\begin{aligned} &\text{minimize} \quad E(\boldsymbol{s}) \\ &\text{subject to} \quad C1: \; g_t(\boldsymbol{s}) = t(\boldsymbol{s}) - t_{\max} \leq 0, \\ &\qquad\qquad\; C2: \; g_p(\boldsymbol{s}) = P(\boldsymbol{s}) - P_{\max} \leq 0, \\ &\qquad\qquad\; C3: \; g_T(\boldsymbol{s}) = G_t^{-1} \times P(\boldsymbol{s}) + T_{\text{amb}} - \boldsymbol{T_{\max}} \leq 0 \end{aligned} \tag{3.20}$$

The first inequality ensures that the new response time does not exceed a maximum time constraint. Setting $t_{\max} = t_{\text{base}}$ implies that the optimized system should have same or better performance as the baseline configuration. Furthermore, one can enforce a higher (lower) performance target by choosing $t_{\max} < t_{\text{base}}$ ($t_{\max} > t_{\text{base}}$). In a similar fashion the second inequality, which is the power constraint can be enforced. $C3$ in Equation 3.20 specifies that the ambient temperature (T_{amb}) plus temperature increase due to scaled average power multiplied by the thermal conductivity matrix (G_t) [57] does not exceed maximum temperature ($\boldsymbol{T_{\max}}$) constraints at points of interest. We note that this constraint serves as a design aid, and it does not provide strict run-time guarantees since instantaneous power can exceed the average power. Therefore, dynamic thermal management techniques that would watch the actual temperature and throttle the system if necessary are still needed. Furthermore, the thermal constraints could be relaxed since it is acceptable to violate them at run-time for short time durations [54]. Details of thermal modeling are skipped since it

is not the focus of this work.

We can also add equality constraints to this optimization problem when focusing on a subset of the platform. For example, setting $s_i = 1$ implies that the i^{th} resource remains unchanged. In general, we can add the following equality constraints:

$$h_i(\boldsymbol{s}) = s_i - 1 = 0, \text{ for } 0 \leq i \leq m - 1 \tag{3.21}$$

where m is the number of resources that are not touched.

The solution to the optimization problem can be found by using Langrange multiplier method with inequality constraints [6]. In summary, we define the Lagrangian function

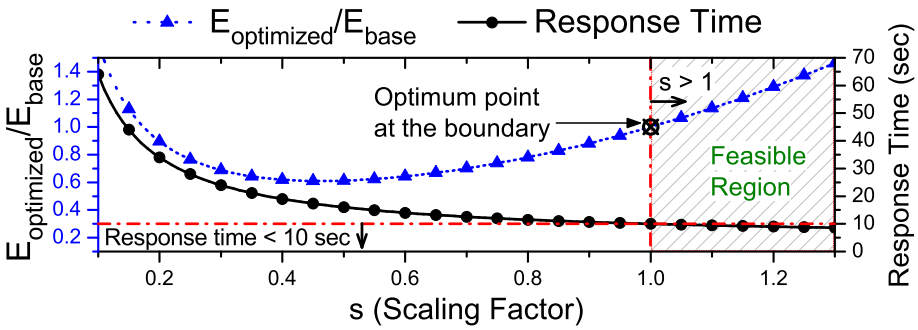
$$L(\boldsymbol{s}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = E(\boldsymbol{s}) + \boldsymbol{\lambda^T} h(\boldsymbol{s}) + \boldsymbol{\mu^T} g(\boldsymbol{s}) \tag{3.22}$$

where $h(\boldsymbol{s})$ and $g(\boldsymbol{s})$ are the combined equality and inequality constraints, respectively, while $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$ are the Lagrange multipliers. The optimum solution $\boldsymbol{s^*}$ can be found by solving the systems of equations described by:

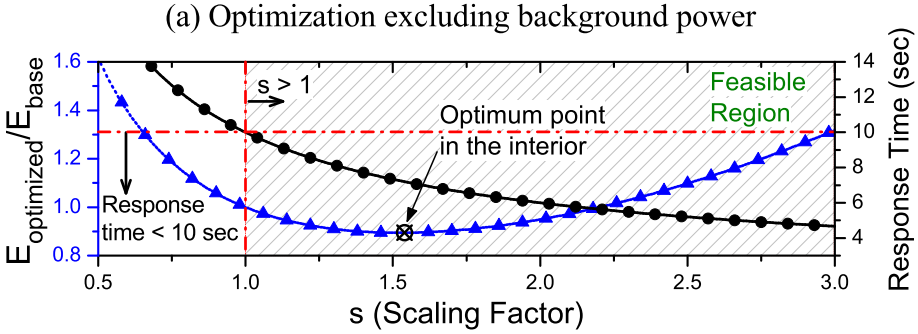
$$\begin{aligned} \nabla L(\boldsymbol{s}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = \nabla E(\boldsymbol{s^*}) + \boldsymbol{\lambda^T} \nabla h(\boldsymbol{s^*}) + \boldsymbol{\mu^T} \nabla g(\boldsymbol{s^*}) = 0 \\ \boldsymbol{\mu^T} g(\boldsymbol{s^*}) = 0 \\ \mu_i \geq 0 \; \forall i \end{aligned} \tag{3.23}$$

3.6 Illustrative Example

The solution to Equation 3.20 can be either at the boundary defined by the inequality constraints ($g(\boldsymbol{s^*}) = 0$) or in the interior region. For illustration purposes, we consider a simple example with two resources and only the timing constraint ($g(\boldsymbol{s})$). We set the scaling parameter of one of the resources to $s_0 = 1$ and optimize the other one to minimize the energy consumption without increasing the original response time, $t_{\text{base}} = 10$. Figure 3.6(a) shows that the energy savings could be maximized by slowing down the scaled resource (left ordinate). However, the response time plot (right ordinate) clearly shows that



(a) Optimization excluding background power



(b) Optimization including background power

Figure 3.6: Illustrative example of the Optimization Problem Presented in Section 3.5 with Two Resources (N=2). By Including Background Power in Optimization, Significant Energy Savings can be Obtained by Speedup.

the reduction in energy comes at the expense of the response time. When we consider the response time constraint $t_{\text{base}} \leq 10$, then the optimum point moves to the boundary. Note that in this case the energy consumption can be reduced compared to the baseline only if the timing constraint is relaxed. On the other hand, the example in Figure 3.6(b) illustrates a case where there is a substantial platform power due to display and PMIC. Their power consumption is relatively steady during the execution, and they can be powered down upon completing the execution. Under this scenario, running the application faster improves the platform energy as depicted in Figure 3.6(b). More precisely, accelerating the scaled resource by $1.5\times$ improves both the response time and the overall energy. In summary, it is very important to note that the optimality depends on overall platform power and the DPM algorithm which determines when the resources can go to low power states.

Introducing New Cores and Accelerators: Restricting optimization only to scaling the existing resources limits the energy savings due to the inverse relationship between power consumption and performance. Energy efficiency can be improved further by adding more resources or replacing the existing resources with more efficient ones. For example, a parallel task can be divided into multiple PEs to reduce the run time with a only linear increase in the power consumption, as opposed to the relation in Equation 3.15. Similarly, application specific accelerators, like image processors, can execute a certain task both faster and with lower power consumption compared to general purpose processors. We can analyze both of these cases using the proposed framework due to the generality of our approach and models introduced in sections 3.1 to 3.5.

3.7 Dynamic Implementation of Framework

To solve the optimization problem mentioned in Equation 3.20, we currently use Matlab's `fmincon` solver on a desktop x86 machine. However, we can not use an optimization solver like `fmincon` for each application on a mobile device at run time, this is to keep the decision overhead less. In turn, this requires us to explore other techniques, e.g., saving information at design time, from different usage scenarios called Key Performance Indicators (KPIs). In this section, we demonstrate how to use KPIs at design time and obtain energy efficiency at run time. In particular, one can run many KPIs on the mobile platform and then obtain optimized scaling factors from the proposed framework for each. Then, we can use the optimal scaling factors at run time. Note, we may not be able to use the scaling factors directly as each resource needs to satisfy the maximum and minimum scaling parameters for which they are configured. Here, we present an illustrative example to clarify the implementation.

Illustration: Consider a system with three resources R1, R2, and R3, and three KPIs, A1, A2, and A3. We can determine optimal scaling parameters for each KPI at design

time. In this example, let us also take frequency as the scaling variable, and suppose, we obtain optimal frequencies for co-optimizing all resources together for each of the three KPIs as shown in Table 3.2. It is clearly possible, that there exists no set which has all absolute maximum or minimum scaling frequencies. Therefore, we use the maximum and minimum range for all KPIs per resource. In this particular example, the range for each resource will be $\mathrm{R1}\ \epsilon\ [0.35, 0.80]$, $\mathrm{R2}\ \epsilon\ [1.20, 2.20]$ and $\mathrm{R3}\ \epsilon\ [1.10, 1.30]$, respectively. There can be a case, when a resource can not satisfy the optimal scaling frequency obtained. Assuming $\mathrm{R2}$ can only run at maximum frequency of $2\ \mathrm{GHz}$, then, in this case its scaling range will get reduced to, $\mathrm{R2}\ \epsilon\ [1.20, 2.00]$. Once, these parameters have been determined at design time, one can map them to the performance counters in the platform. After this, the governor [48] or scheduler will watch the performance counter for each new application that runs on the platform and make decisions to scale the resource parameters.

Table 3.2: Illustrative Example Showing Optimal Sets of Frequencies in GHz for Three KPIs. Each Row Represents Optimal Frequency Set for a Particular KPI.

	R1	R2	R3
A1	0.40	1.90	1.10
A2	0.80	1.20	1.20
A3	0.35	2.20	1.30

Now, we describe the KPI method from a mathematical perspective, suppose, we run M KPIs on a system with N components (e.g., number of rows and columns in Table 3.2 will be M and N, respectively), we can have three cases, 1) either run each KPI separately or 2) run a KPI together with another KPI(s) or 3) run all KPIs together. Number of possible

combinations of these M KPIs are,

$$\begin{aligned} \text{Total optimal scaling sets} &= C(M,1) + C(M,2) + C(M,3) + \cdots + C(M,M) \\ &= 2^M - 1 \end{aligned} \tag{3.24}$$

Out of all the optimal scaling sets, S_i^*, where i is the index of KPI (e.g., each row in Table 3.2), each scaling set has N scaling parameters in it. We find $S_{i,max}^*$ and $S_{i,min}^*$ for each resource. After comparing with the actual parametrization possible, these scaling parameters will be mapped to performance counters. Subsequently, the governor can dynamically vary the parameters during runtime for new applications.

$$S_i^* = \begin{pmatrix} s_0 & s_1 & s_2 & \cdots & s_N \end{pmatrix} \tag{3.25}$$

Chapter 4

EXPERIMENTAL SETUP AND METHODOLOGY

We used a two-step approach to validate the proposed energy optimization framework. To achieve our objective, we developed a Matlab model that can take the response time contributions and the power consumption of various platform resources in off, idle and active states as input. Then, the model solved the optimization problem described in Section 3.5 and gave the optimal scaling value. In other words, the model took the response time and power parameters detailed in Sections 3.1 and 3.3 and solved constrained optimization problem given in Equation 3.20. Then, as the first step, we predicted the inputs using published results in [50] and used them to validate our models and investigate the scaling behavior. The summary of the first step is explained in Section 5.1. It is important to emphasize that obtaining power consumption and response time decomposition per resource is quite challenging unless there is a proper experimental setup. Therefore, after validating the Matlab models, we constructed an experimental setup using Snapdragon based development platform and performed detailed measurements ourselves. The details of the experimental setup are below while the results are presented in Section 5.2.

4.1 Data Measurement Setup

4.1.1 Host Platform

We performed data collection using a mobile platform based on the Qualcomm Snapdragon 800 processor [52] running Android Jelly Bean 4.2.2 [24] based on the Linux kernel version 3.4.0. The processor has four ARM cores which can be hot-plugged. Furthermore, each CPU core has four power sleep states – $C0$, $C1$, $C2$ and $C3$. The $C3$ state is the deepest sleep mode and has longest wake up time. While, $C0$ state is the shallowest sleep

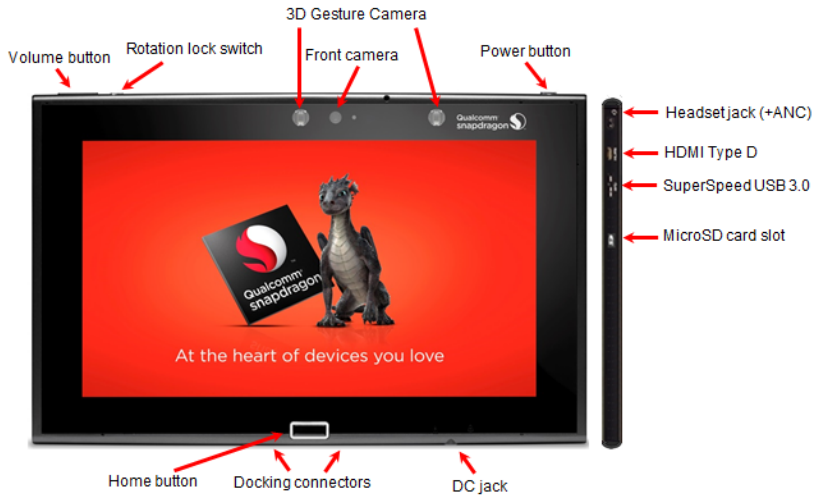


Figure 4.1: A State of the Art Mobile Development Platform/Tablet Based on Qualcomm Snapdragon 800 [52], Used in Experimental Evaluation.

mode with instant wake up. There are fourteen discrete frequencies to scale the operating frequency of each CPU core as shown in Table 4.1. Furthermore, the voltage in Snapdragon processor varies as a linear function of frequency, as shown in Section 3.4. We took frequency as the scaling variable and obtained the power and time values for major resources like CPU and display in the tablet.

4.1.2 Benchmarks

We used a representative set of benchmarks for mobile platforms, one set comprised of Basic math large, JPEG, Patricia from Mi-bench [25] and the other set, Realistic-generalized web browsing (R-GWB), Audio and Video benchmarks from [49, 26]. However, we can not measure the response time of each resource for the second set of benchmarks, because, we can not run their main processes directly from terminal, in isolation. For example, the web benchmark contains a static HTML start page and JavaScript based dynamic content emulation of a given set of popular websites. Loading the static page and starting the dynamic content, are two separate processes. One can invoke the first process through

terminal, but not the second process, without making changes to the benchmark itself. In this work, we present the first set of benchmarks to explain experimental verification of our framework.

Table 4.1: Frequency Table for Snapdragon 800 Processor. There are 14 Different Frequency Values Available to the User.

Frequency (MHz)
300.00
422.40
652.80
729.60
883.20
960.00
1036.80
1190.40
1267.20
1497.60
1574.40
1728.00
1958.40
2150.40

A brief summary of all the benchmarks is as follows:

Basic math large: Basic math is used as a test for automotive and industrial control applications. It performs mathematical calculations which do not have dedicated hardware support in an embedded processor. The input data is a fixed set of constants.

JPEG encode/decode: JPEG is a consumer devices test benchmark. It is a standard compression image format used in cases when some data loss is acceptable. We took three images totaling a size of about 14 MB and first decoded, then encoded them sequentially

one after the other.

Patricia: Patricia is used to test the network capabilities of embedded processors, e.g., switch and router applications. Patricia is a radix tree, which is used instead of full trees with very sparse leaf nodes. The input data used in this benchmark is a list of disguised IP traffic from a highly active web server for a 2 hour period.

Web Browser, Audio/Video: The web browser benchmark is used to emulate user behaviour like iterations, scroll, page change, zoom-in/out on a real web browser. The application is based on HTML [63] and JavaScript [42]. All the dynamic content like, counting of iterations, scroll, etc. are implemented in JavaScript, while the results and index pages are for showing static results are implemented in HTML. We used native Android web browser for this task. For video and audio we have used Android's native media player.

4.1.3 Power and Response Time Measurement

Qualcomm's Trepn [53] profiler on the mobile device under test enabled us to measure power or electrical current across the entire platform as well as for each resource separately. The profiler initially established a baseline during the first 5 seconds. It then profiled the power(s) every 100 milliseconds. Using this setup, we measured the power consumption of CPU, display and memory. As a result, we obtained a fairly accurate power consumption decomposition as a function of time. Next, the logged data was saved in form of tables as .CSV format with time stamps. Along with the power measurements we also used `time` command in android shell to get the `real` , `user` and `system` time of an application [23]. We used the real time as the primary performance metric (response time). CPU time is the sum of user and system time. A detailed procedure to measure power and time are given in Appendix A.

4.1.4 Energy Measurement

We defined, the energy with background power as the product of battery power and real time, and energy without background power is the product of total CPU power (sum of average CPU delta power, average CPU absolute power and Graphics, WLAN/BT/FM, SD-Card if applicable) and real time.

4.2 Data Analysis

The .CSV file had information of all the selected data points (resources) along with the delta values at each instant of time from start of the profiler to end of the profiler. We calculated three powers based on this data; 1) Average Battery power, 2) Average CPU Delta Power, 3) Average CPU Absolute Power. Current version of Trepn profiler does not give rail to rail absolute power for any of the resources except total battery power.

4.2.1 Average Battery Power

The average battery power is the power consumed while the application was running with display on. We calculated it by taking average of the battery power (raw) values for the time range during which the benchmark ran.

4.2.2 Average CPU Delta Power

When Trepn starts profiling, it establishes a baseline power value during the "Baselining interval" time. The CPU delta power is the difference between the current power and the initial baseline power. This notified us whether running the application results in an increase in the power consumption or not. We found the average CPU delta power by taking average of all the values of CPU delta power when the benchmark was finished running.

4.2.3 Average CPU Absolute Power

Since, Trepn profiler does not give absolute power consumed by the CPU, we took average CPU absolute power equal to the average battery power when Display, Wi-Fi and all non-essential background processes are off. That is, no application was running in the background/foreground except Trepn profiler. We show in Section 5.2 that this method yields good results when more than one CPU cores are online, but sometimes does not work well with only one core online case. The results obtained in raw form can have variations because of many processes running in the operating system, hence, we normalized the values to obtain the average value for CPU absolute power.

Chapter 5

EVALUATION AND VALIDATION OF FRAMEWORK

The framework is implemented with the power, performance and energy models presented in Chapter 3. After implementing these models in Matlab, we optimize energy using `fmincon` function with response time and power constraints. The proposed framework can predict power, time and energy values at any frequency by using only one base frequency (scaling value) and the corresponding power and time values as its inputs. Given, a set of constraints and the background power, we can also find the optimum point of operation for a given setup/benchmark. In this chapter, we first provide analytically evaluated results of our framework in which input data is taken from Pathak et al. [51]. Then we experimentally evaluate the framework with data obtained from our setup as explained in Chapter 4.

5.1 Analytical Evaluation

In this section, we first explain our methodology for evaluation and then discuss results for optimization using only scaling and adding new resources. We also show a comparison of optimization of resources one after the other and co-optimization of all the resources together.

5.1.1 Methodology

We start with a baseline system, i.e., all the scaling factors in Equation 3.1 and Equation 3.15 are equal to one, and optimize it by finding the optimal scaling coefficients as explained in Section 3.5. We use the document converter (docconv) and photo uploader (puploader) applications presented in [51] as the driver applications and adopt the power

and performance values reported therein. Hence, the baseline systems are already viable design points rather than ad-hoc choices. The total power of the shared platform resources is set to $P_{\mathrm{BG}} = 1W$ considering that about 0.5 W is dissipated in the display [10] itself. All the energy savings reported hereafter in Section 5.1, are with respect to the *total platform energy*. Finally, we set the response time constraint as $t_{\max} = t_{\mathrm{base}} + t_{\mathrm{slack}}$, where t_{slack} is 10% of the total run time for both applications to provide extra timing headroom when the optimum solution is at a boundary point like in Figure 3.6(a).

5.1.2 Optimization Using Scaling Only

This section, illustrates the proposed framework by optimizing the baseline system without increasing the number of resources. We use four approaches and summarize the results in Table 5.1. First, we scale only the dominant resource, i.e., the one with the largest impact on energy before optimization, while setting the scaling factors of other resources to one (*row 1*). The second approach scales all resources incrementally one by one (*row 2*). That is, the dominant resource is optimized first. Then, its scaling factor is kept constant while the next resource is optimized, and this is repeated until all resources are scaled. In contrast to this, the third one co-optimizes all resources at once (*row 3*), while the fourth approach replaces the dominant resource with a more energy efficient resource (*row 4*). We apply these four approaches under two scenarios. The first scenario assumes that the DPM algorithm turns shared resources like display off as soon as the execution is complete. Hence, the background power is taken into account during optimization ($P_{\mathrm{BG}} = 1W$ in Table 5.1). In the second scenario, we assume that multiple applications are running at the same time. Hence, we set $P_{\mathrm{BG}} = 0$ since the shared resources remain on even after the execution of the target application.

Scaling only the dominant resource does not provide any significant energy savings, since the baseline system is already a good starting point and the inverse relationship be-

tween power and performance does not leave much room for improvement. Likewise, incremental optimization provides little improvement except for docconv with $P_{\mathrm{BG}} = 1$, where accelerating both CPU and memory helps in reducing the total energy. We observe that co-optimization consistently outperforms incremental optimization. The most notable savings are obtained for the puploader application with $P_{\mathrm{BG}} = 1$, where co-optimization reduces the energy to $0.87\times$, while incremental optimization decreases to $0.94\times$ of the baseline energy. Finally, we observe that using an energy efficient processing element is by far the most effective approach for all scenarios. While this conclusion is intuitive, our formulation quantifies the savings by providing precise numbers under various scenarios.

5.1.3 Optimization Using Additional Resources

Introducing new cores: In this experiment, we increase the number of resources by adding new CPU cores to the baseline system. After adding each core, we distribute the workload of the original CPU core equally among all the cores, and solve the optimization problem presented in Section 3.5. We neglect the synchronization overhead and assume that the workload is fully parallelizable since dealing with full details of synchronization is beyond the scope of this thesis. We note that a communication time term, which is an increasing function of the number of cores, could be used to approximate this overhead. Figure 5.1 shows that adding new cores results in significant improvement in the energy and in particular when the number of cores is 8 and below. However, we observe diminishing rate of returns since the savings in run-time level out. What is more, reported energy savings are optimistic as the synchronization overhead is neglected, and they come at the expense of additional core area. Therefore, we analyze next the energy savings obtained using domain-specific accelerators.

Introducing an accelerator: We add an accelerator, which takes $5\times$ less time and consumes $5\times$ less power compared to a CPU core, to the baseline system. We note that

domain-specific accelerators are indeed more energy efficient [30]. Instead of distributing the workload equally to multiple cores, we increase the percentage of workload offloaded to the accelerator. Figure 5.2 shows the improvement in energy consumption as the amount of workload offloaded to the accelerator increases. We show the x-axis in equivalent number of cores for easier comparison with Figure 5.1. For example, offloading 75% of the workload, which is equivalent to sharing the workload among 4 cores *in this setup*, results in slightly larger savings than increasing the number of cores to 4 under ideal parallelization assumption. Furthermore, this improvement can be obtained with much smaller area overhead due to the superior area efficiency of accelerators. This same trend is also confirmed using the puploader application as shown in Figures 5.3 and 5.4.

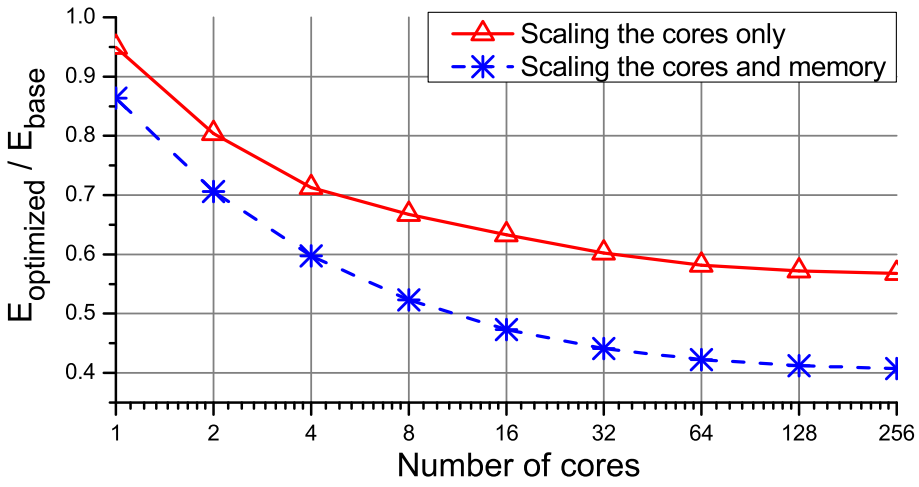


Figure 5.1: Normalized Energy Consumption with Increasing Number of Cores for Docconv Application, After Including Background Power and Co-optimization.

5.1.4 Assumptions

The dynamic power scaling exponent $\alpha = 2.5$ in Equation 3.15, collective power of all resources not being scaled, Platform power, (P_0) is 1W. This is a modest assumption considering about 0.5 W is dissipated in Display [10] itself. Response time can be relaxed

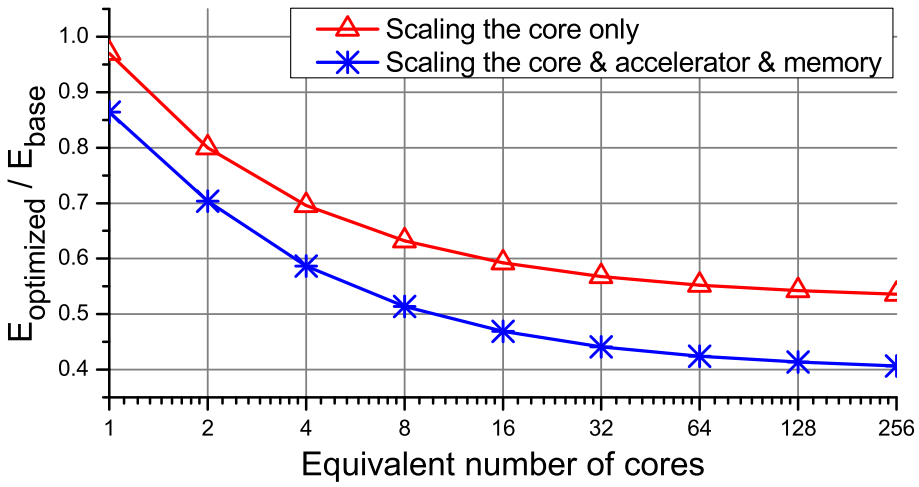


Figure 5.2: Normalized Energy Consumption with Increasing Equivalent Core Workload for Docconv Application, After Including Background Power and Co-optimization.

by 5% in docconv and puploader. Multicore and accelerator architectural change results of docconv and puploader are similar but we show here docconv results only because CPU plays a dominant role in it unlike puploader where network takes up more energy. Overheads in communication are neglected for illustration purpose.

5.1.5 Observations

Table 5.1 show the contrast in values for six cases with slack for docconv and puploader applications. When platform power is included during optimization we get less savings than when we don't include it. In reality this result can be misleading. E.g., when platform power is ignored during the optimization with New IP scaling, we observe a reported savings of about 26.73% for CPU in docconv and 36.09% for network in puploader applications which are significantly higher than the actual savings of 1.35% and 10.39% respectively. The actual savings can be calculated by adding the energy due to platform power consumed during the entire time application is active. If the platform is active irrespective of the application being considered then the reported savings are also the actual

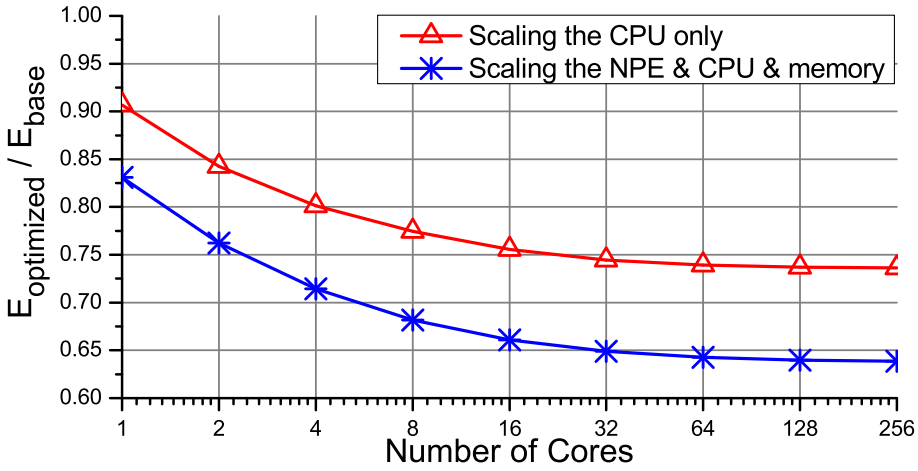


Figure 5.3: Normalized Energy Consumption with Increasing Number of Cores for Puploader Application, After Including Background Power and Co-optimization.

savings.

Figure 5.1 shows plot of normalized energy when more cores are added to the docconv application system. When no core is added to the system it is same as original with one CPU corresponding to number of cores equal to 1. Even for a single core we can get some savings with scaling the core, e.g., using DVFS. The reason for saturation of energy as the number of cores increase is because although the time decreases linearly with addition of cores, exponential component of power increases linearly too, which causes bigger change in power than in time.

Figure 5.2 shows plot when an accelerator with equivalent core area is added to the original system of docconv. Typical energy efficiency of an accelerator compared to the core is about 25 [30]. In contrast to Figure 5.1 we can see that if all resources are scaled then we can get same energy efficiency with much less area. We would be limited in getting better energy savings with only scaling the CPU because the workload being processed by it becomes less.

We observe, when system is accelerated sufficiently high, energy savings drop and the

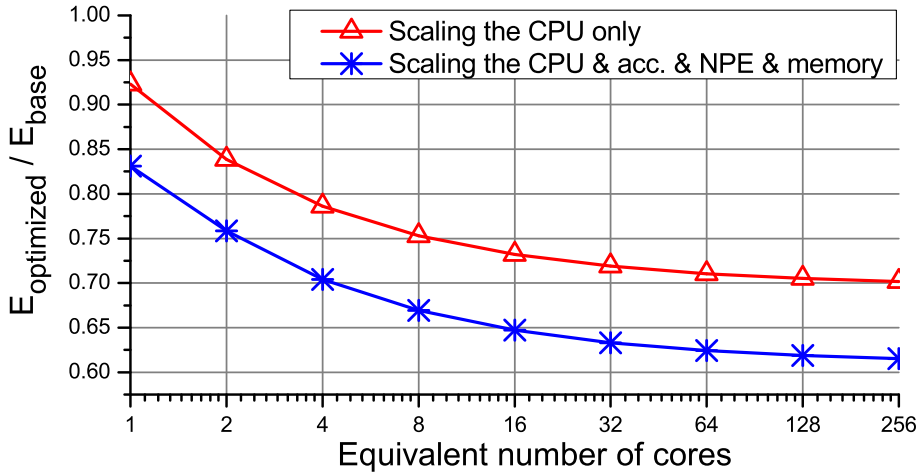


Figure 5.4: Normalized Energy Consumption with Increasing Equivalent Core Workload for Pu-ploader Application, After Including Background Power and Co-optimization.

optimized results become same. This is because of the competition between time and power scaling. After a certain point the optimized results saturate due to response time deadline. Similar set of plots would come for energy savings with new IPs too, due to space consideration we have not included the plot when platform power is ignored and new CPU/Network IP is added replacing the original resource.

Table 5.1: Normalized Energy of Docconv and Puploader Apps with Memory (Disk for Puploader), CPU and Network Processing Element (NPE). Incremental Optimization is Shown Using '→', While '+' Signs Implies Co-Optimization.

Application	P_{BG}	**Optimized Resource(s)**	**Scaling Vector**			$\frac{E_{\mathrm{opt}}}{E_{\mathrm{base}}}$
			Mem	**CPU**	**NPE**	
Document converter	0 W	CPU only	1.00	0.85	–	0.91
		CPU → Mem	1.00	0.85	–	0.91
		CPU + Mem	0.91	0.91	–	0.89
		New CPU + Mem	0.85	0.97	–	0.69
	1 W	CPU only	1.00	1.38	–	0.95
		CPU → Mem	1.52	1.38	–	0.88
		CPU + Mem	1.62	1.61	–	0.86
		New CPU + Mem	1.67	1.91	–	0.78
Photo uploader	0 W	NPE only	1.00	1.00	0.85	0.94
		NPE → CPU → Mem	1.00	1.00	0.85	0.94
		NPE + CPU + Mem	0.77	0.95	0.95	0.87
		New NPE + CPU + Mem	0.69	0.86	1.05	0.61
	1 W	NPE only	1.00	1.00	1.17	0.99
		NPE → CPU → Mem	0.96	1.20	1.17	0.98
		NPE + CPU + Mem	0.98	1.21	1.20	0.97
		New NPE + CPU + Mem	1.01	1.25	1.53	0.83

5.2 Experimental Evaluation

This section presents the results of the experimental evaluation performed using snapdragon MDP [52]. The actual and predicted results of response time, power and energy are obtained by running the Mi-bench [25] benchmarks – Basic math, JPEG and Patricia on the platform. We repeated this profiling for each frequency. Then, we chose a baseline frequency of 1.0368 GHz and fit the response time and power to our models presented in Chapter 3. This gives us the power state and response time distribution in resources. We then give the corresponding power and time values as input to our framework and obtain the optimal frequency. We observe this optimal frequency value to be same as the frequency value of minimum energy obtained with actual experiments. We also observe, how the increase/decrease in background power and constraints affects the optimum frequency. The results obtained for this are similar to the discussion in illustrative example in Section 3.6.

First, we discuss both 4 core and 2 core online cases of each benchmark and then present the 1 core online case. We observe that our models can not capture the memory time when running on 1 core in case of Basic-math and Patricia benchmarks. But they are able to fit for JPEG benchmark, this is explained in Section 5.2.4. The goodness of fit and error values are shown in the Tables 5.2 and 5.3.

5.2.1 JPEG Benchmark with Four and Two Cores Online

The JPEG benchmark runs are captured for a range of frequencies as shown in Figures 5.5, 5.6 and 5.11 for four, two and one core, respectively.

In case of 4 cores, the CPU load for the entire run duration of the benchmark was less than 100%. This information is important because, higher CPU load can cause random lags in process execution. As seen from Figure 5.5 the response time and CPU power are very closely captured by our models, the predicted and the actual values are similar.

The response time is inversely proportional to the scaling factor. Hence, it decreases as frequency is increased. Similarly, as power is a function of order greater than 1, we see that power increases with increase in frequency. These are clear intuitive trends, but quantified prediction of the result is not intuitive, and is one of the main contributions of our work. We also find, the energy without background power (P_{BG}) is similar between predicted and actual values, as seen in Figure 5.5. While, we have an exception here, an outlier point in energy at about 1.2 GHz. This is caused due to transition from over estimating to under estimating of power value at this point and is consistent among all measurements. We see that minima occurs at about 1.6 GHz for both predicted and actual values. A similar trend is seen in energy with background power included. The minima point for this energy occurs at about 3.5 GHz according to our framework, unfortunately we can not run the actual processor at that frequency to verify the result experimentally. It is my thesis that the trend in frequencies lower than 2.2 GHz should be enough for the reader to get convinced of the outcome. If not, this scenario at least shows software engineers that they should run their application at the highest frequency possible for maximum energy efficiency.

We make the same observations for the JPEG benchmark run with 2 cores as seen in Figure 5.6, this time our predicted minima is close to the actual minima, but not very accurate. First, we observed that the CPU was running on 100% load. This, together with operating system processes and profiler overhead causes less accurate response time calculations. There can be several reasons for this observed inaccuracy but we do not have a definitive way to verify it as of now. The verification would require very detailed tracing capability of each process at run time. The current profiling tools available in the industry are not advanced enough to carry this task seamlessly.

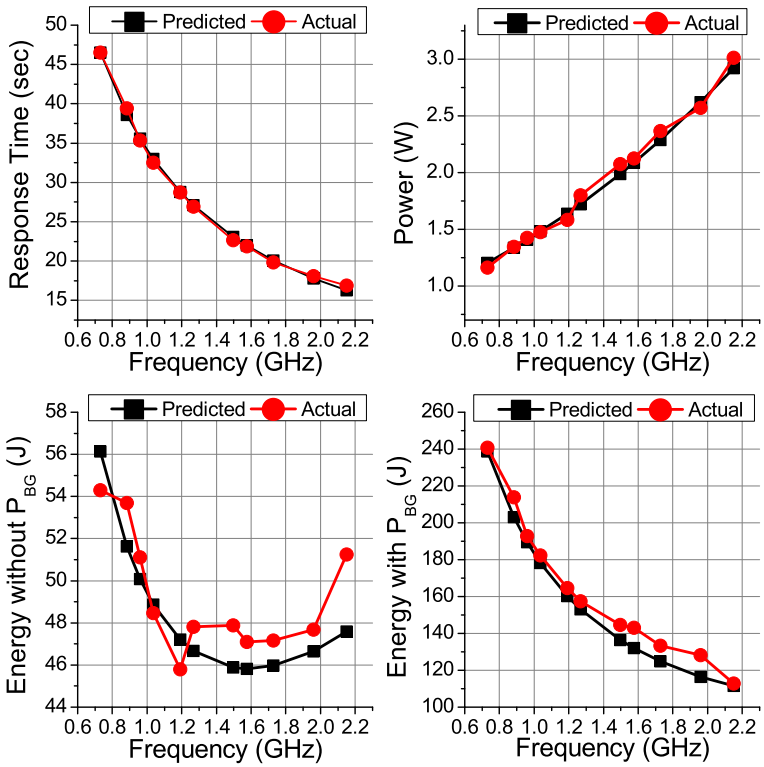


Figure 5.5: Measurement Results Showing Predicted and Actual Values for JPEG Benchmark Running on 4 CPU Cores. Anticlockwise – The Top Right Chart Shows Power (Excluding Background Power), Next Response Time, Then Energy (Excluding Background Power) Showing Same Optimal Frequency for Predicted and Actual Values, and Next Chart for Energy (Including Background Power) Showing Speedup Saves Energy.

5.2.2 Basic Math Large Benchmark with Four and Two Cores Online

We observe a similar behaviour to the JPEG benchmark presented in Section 5.2.1. The minima occurs at the same frequency (about 1.7 GHz) for both predicted and actual values as seen from Figure 5.7. The energy has been over estimated because we have over estimated response time for the mid range of frequency values from 1 GHz to 1.8 GHz.

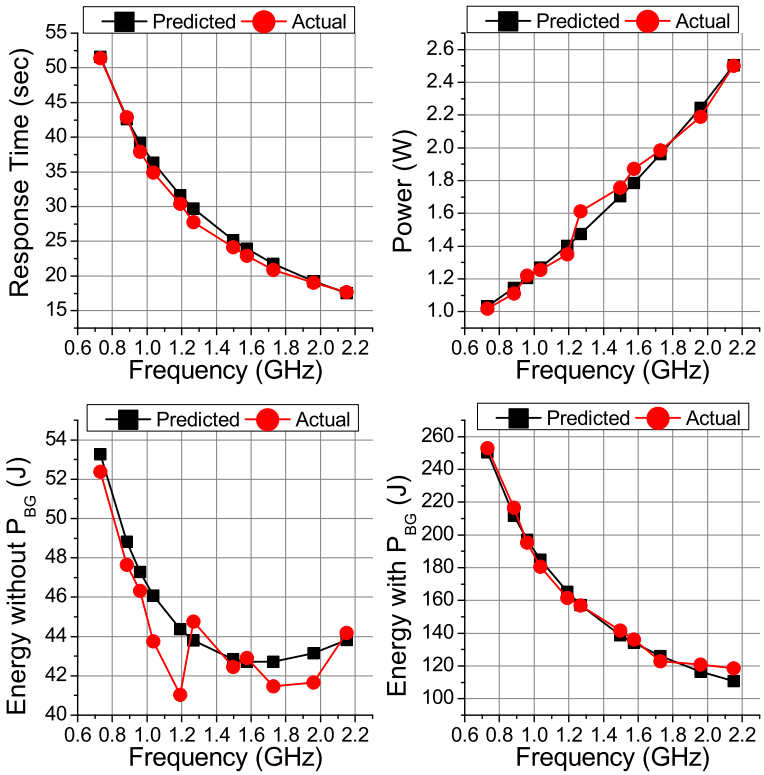


Figure 5.6: Measurement Results Showing Predicted and Actual Values for JPEG Benchmark Running on 2 CPU Cores. Anticlockwise – The Top Right Chart Shows Power (Excluding Background Power), Next Response Time, Then Energy (Excluding Background Power) Showing a Conservative Prediction for Optimal Frequency, and Next Chart for Energy (Including Background Power) Showing Speedup Saves Energy.

For the case when basic math large benchmark runs on 2 cores, we observe an outlier point at about 1.2 GHz. Except for this the minima point is correctly predicted at about 1.9 GHz.

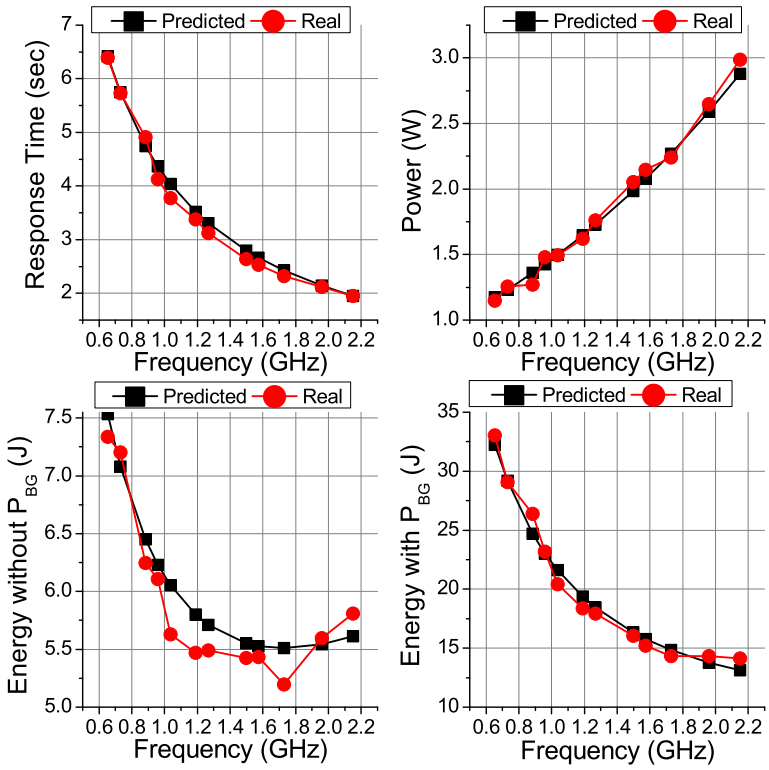


Figure 5.7: Measurement Results Showing Predicted and Actual Values for Basic Math Large Benchmark Running on 4 CPU Cores. Anticlockwise – The Top Right Chart Shows Power (Excluding Background Power), Next Response Time, Then Energy (Excluding Background Power) Showing Same Optimal Frequency for Predicted and Actual Values, and Next Chart for Energy (Including Background Power) Showing Speedup Saves Energy.

5.2.3 Patricia Benchmark with Four and Two Cores Online

We observe a similar behaviour to the JPEG benchmark presented in Section 5.2.1. From Figure 5.9, we see over prediction of the time because of which the energy is over predicted as well. To solve this problem, we fit the time after excluding the two lowest frequencies, we observe a good fit for response time and hence energy prediction is also

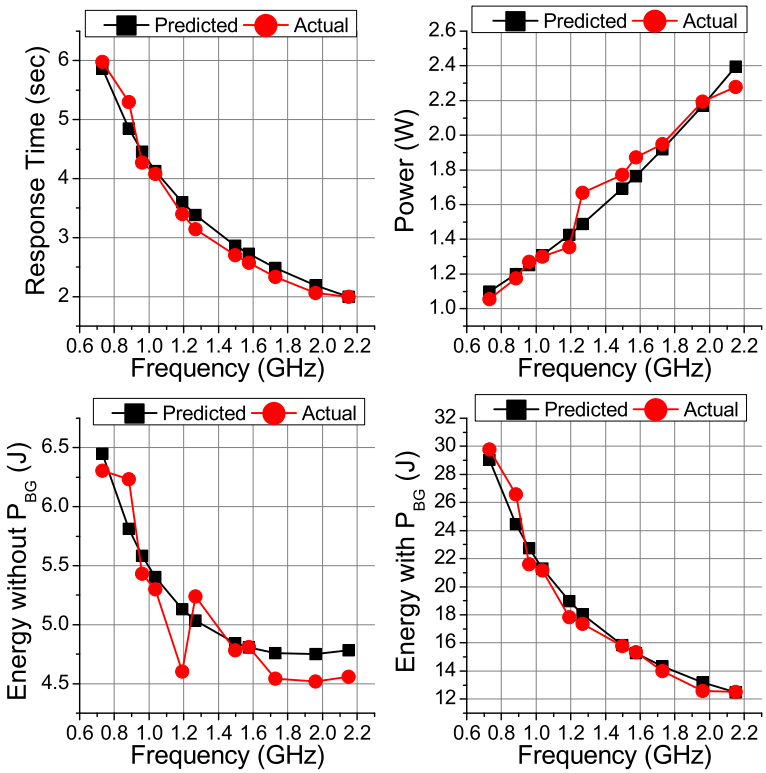


Figure 5.8: Measurement Results Showing Predicted and Actual Values for Basic Math Large Benchmark Running on 2 CPU Cores. Anticlockwise – The Top Right Chart Shows Power (Excluding Background Power), Next Response Time, Then Energy (Excluding Background Power) Showing Same Optimal Frequency for Predicted and Actual Values, and Next Chart for Energy (Including Background Power) Showing Speedup Saves Energy.

more accurate, as shown in Figure 5.10 for 2 core case. Note, at lower frequencies, the operating system processes start interfering with the application, so it makes sense to under predict the response time at these frequencies.

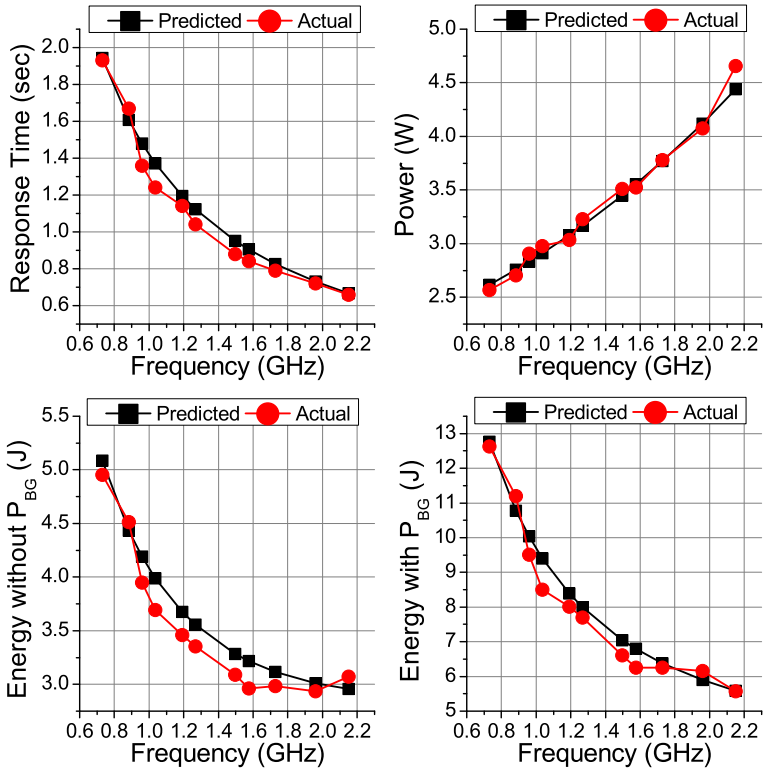


Figure 5.9: Measurement Results Showing Predicted and Actual Values for Patricia Benchmark Running on 4 CPU Cores. Anticlockwise – The Top Right Chart Shows Power (Excluding Background Power), Next Response Time, Then Energy (Excluding Background Power) Showing Same Optimal Frequency for Predicted and Actual Values, and Next Chart for Energy (Including Background Power) Showing Speedup Saves Energy.

5.2.4 Benchmarks with One Core Online

We observe, JPEG benchmark running on one core gives results similar to those obtained with two and four core configuration in Section 5.2.1, i.e., we get approximately good fits for time, power and energy. But for one core case of Basic math large and Patricia, the response time is highly over predicted, which causes the energy to be over predicted

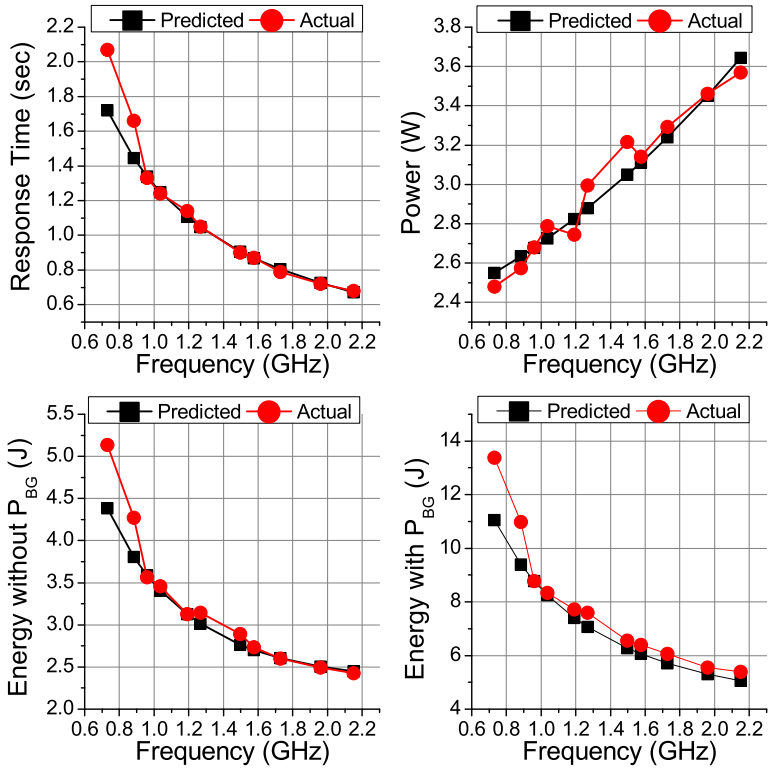


Figure 5.10: Measurement Results Showing Predicted and Actual Values for Patricia Benchmark Running on 2 CPU Cores. Anticlockwise – The Top Right Chart Shows Power (Excluding Background Power), Next Response Time, Then Energy (Excluding Background Power) Showing Same Optimal Frequency for Predicted and Actual Values, and Next Chart for Energy (Including Background Power) Showing Speedup Saves Energy.

as well. This is because, both Basic math large and Patricia benchmarks are highly CPU intensive applications. Hence, collective time taken by other resources in series with CPU (most likely memory), is not constant like in JPEG. In fact, as seen from the Figures 5.13 and 5.12 the memory time scales at a much higher gradient than the model fitting curve, which is inverse scaling of order one (in all other cases, this time is constant). We also

observe that at higher frequencies, both benchmark's memory response times saturate and become constant ($\geq$ 1.6 GHz). This shows, the anomaly has been caused by the operating system and other background processes creating lag at lower frequencies, followed by the observation, all the benchmarks were running at 100% CPU load on 1 core. The predicted and actual values are provided in Figures 5.11, 5.14 and 5.15.

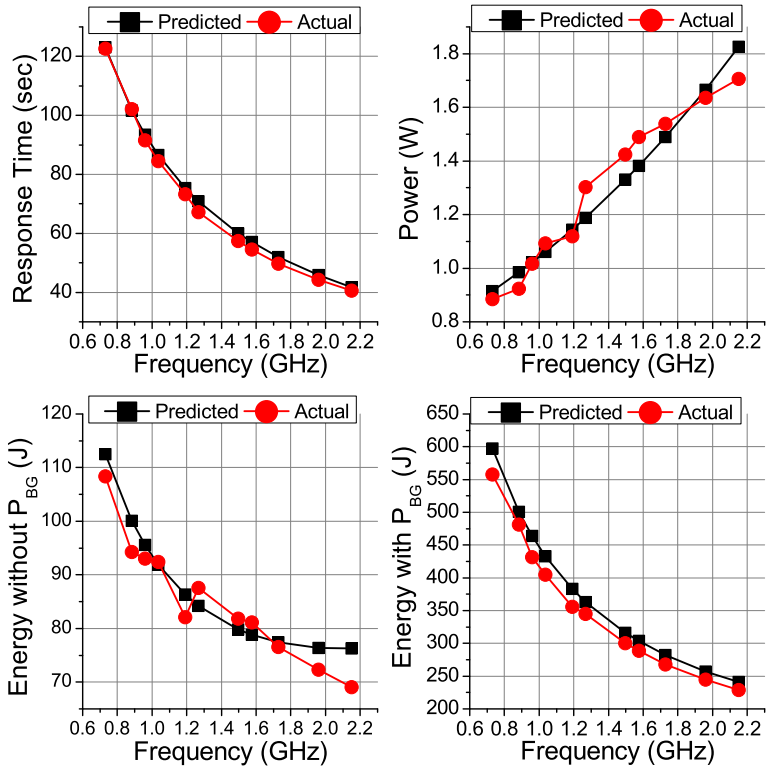


Figure 5.11: Measurement Results Showing Predicted and Actual Values for JPEG Benchmark Running on 1 CPU Core. Anticlockwise – The Top Right Chart Shows Power (Excluding Background Power), Next Response Time, Then Energy (Excluding Background Power) Showing a Conservative Prediction for Optimal Frequency, and Next Chart for Energy (Including Background Power) Showing Speedup Saves Energy.

5.2.5 Increasing the Number of CPU Cores

In this subsection, we present the experimental results of increasing the number of cores and changing the frequency of CPU cores in each configuration shown in Figures 5.16, 5.17 and 5.18. Each of these figures belong to the three benchmarks, JPEG, Basic math large, and Patricia, respectively and show the actual and predicted energy including background power. As seen from Figure 5.16, for frequency $0.73\ \mathrm{GHz}$ the actual energy decreases as we increase the number of cores. For practical purposes, the change in energy from two cores to four cores is very small and we can see the saturation of energy. While, for frequency of $1.73\ \mathrm{GHz}$ we observe that energy first drastically decreases then slightly increases as the number of cores are increased. The increase in energy is because of the added core power dominating the saturated savings in response time. The optimal number of cores for minimizing energy, would increase, if the application can utilize the parallelism more effectively.

5.2.6 Increasing Background Power

Here, the impact of background power on the optimal frequency, for minimizing the energy is provided. The Figure 5.19, we show the optimal frequencies as the background power increases from $0 - 4\ \mathrm{W}$. After the background power increases substantially, the change in CPU energy due to frequency scaling becomes a small fraction of total energy. Therefore, the energy savings by increasing the CPU frequency starts plateauing, which is in analogy with Amdahl's law. Other benchmarks also follow the same trend.

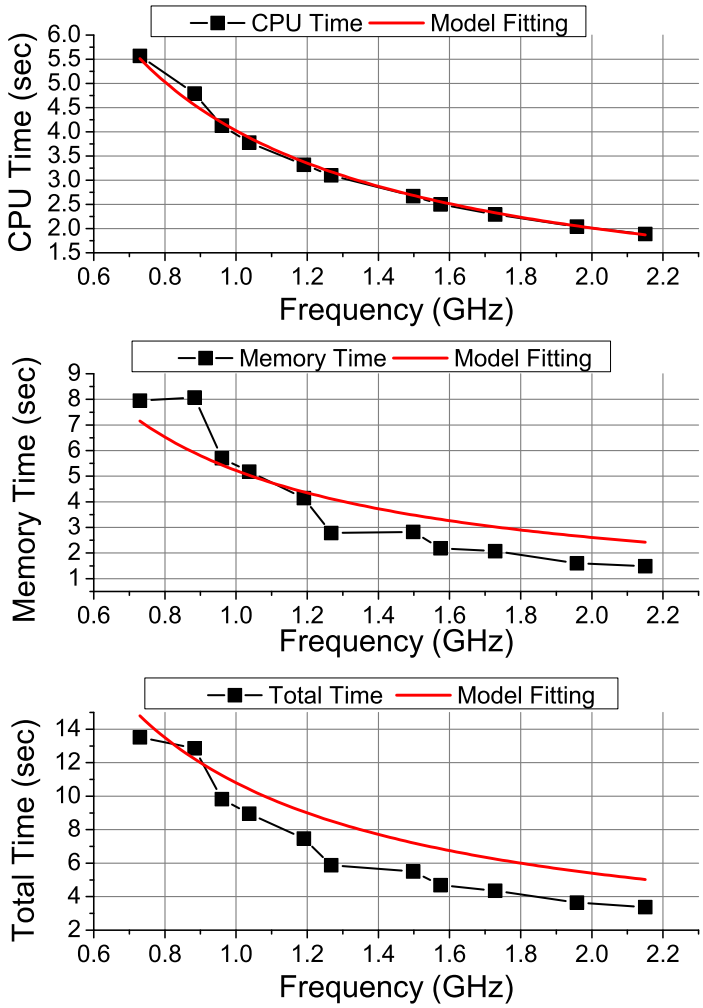


Figure 5.12: Timing Measurement Results for Basic Math Large Benchmark Running on 1 CPU Core. t_{other} is Not Constant at Frequencies$\leq$ 1.6 GHz Which Leads to Over Prediction of Total Response Time. The Fitting Functions are $\frac{t_{cpu}}{s}$, $\frac{t_{other}}{s}$ and $\frac{t_{cpu}}{s} + t_{other}$ Respectively.

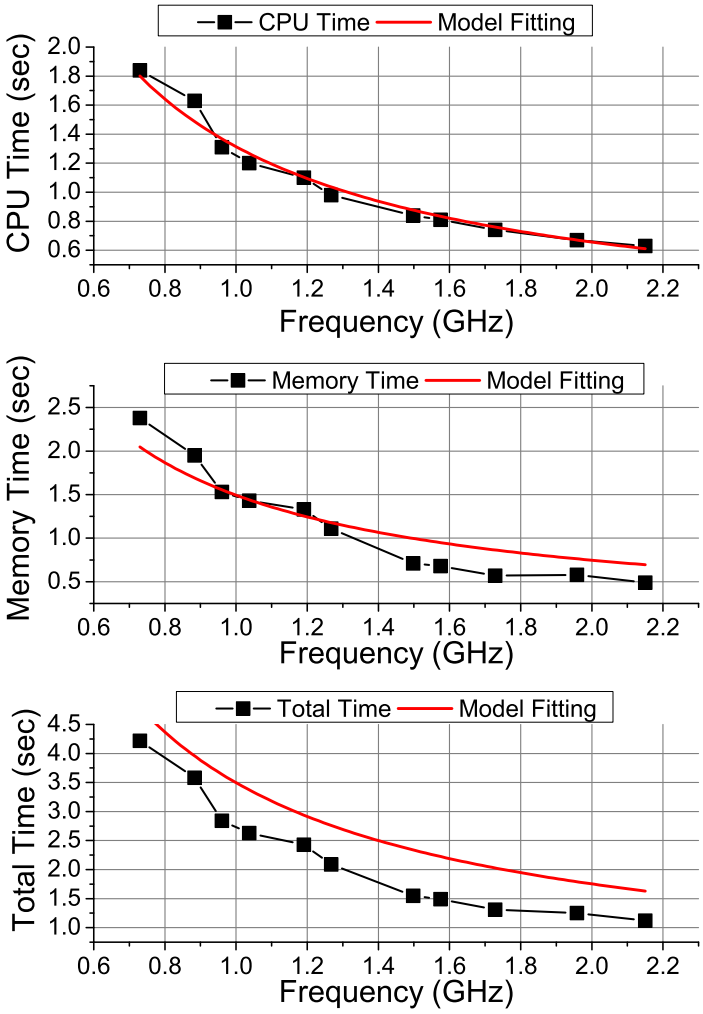


Figure 5.13: Timing Measurement Results for Patricia Benchmark Running on 1 CPU core. t_{other} is Not Constant at Frequencies$\leq$ 1.5 GHz Which Leads to Over Prediction of Total Response Time. The Fitting Functions are $\frac{t_{cpu}}{s}$, $\frac{t_{other}}{s}$ and $\frac{t_{cpu}}{s} + t_{other}$ Respectively.

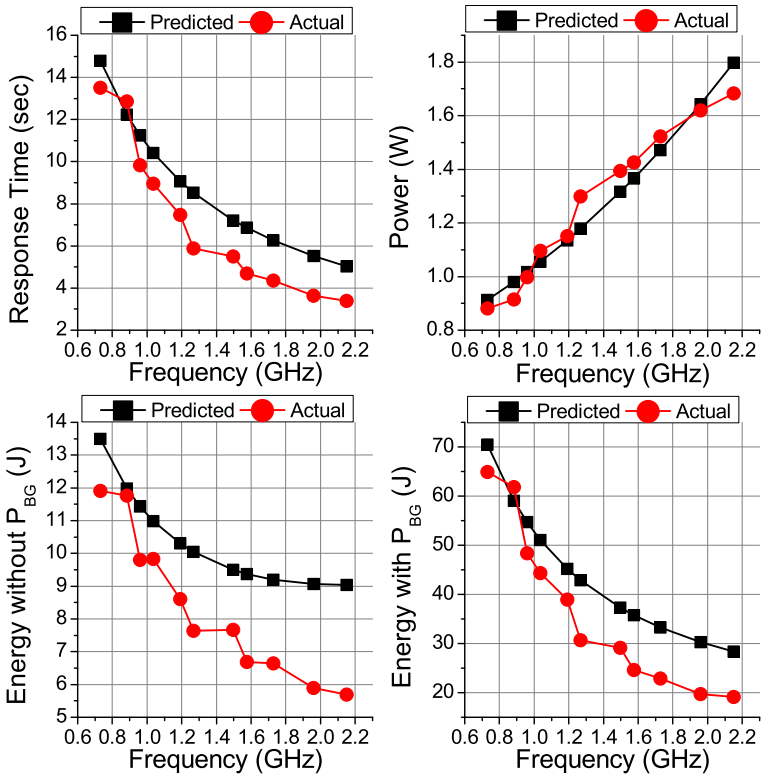


Figure 5.14: Measurement Results Showing Predicted and Actual Values for Basic Math Large Benchmark Running on 1 CPU Core. Anticlockwise – The Top Right Chart Shows Power (Excluding Background Power), Next Response Time, Then Energy (Excluding Background Power), and Next Chart for Energy (Including Background Power). The Response Time is Not a Good Fit Due to Non-linearity of Other Resources' Time in the System

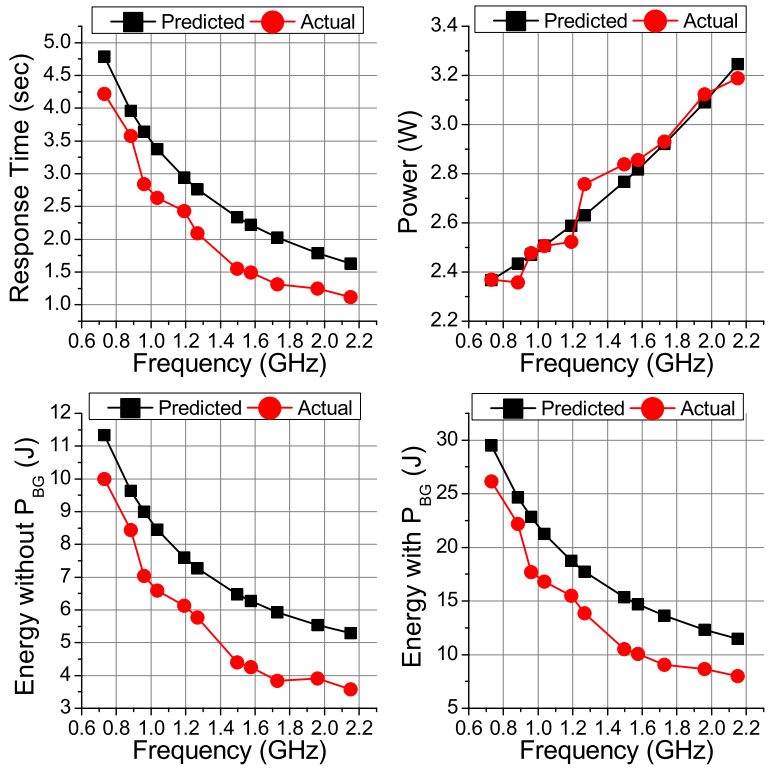


Figure 5.15: Measurement Results Showing Predicted and Actual Values for Patricia Benchmark Running on 1 CPU Cores. Anticlockwise – The Top Right Chart Shows Power (Excluding Background Power), Next Response Time, Then Energy (Excluding Background Power), and Next Chart for Energy (Including Background Power). The Response Time is Not a Good Fit Due to Non-linearity of Other Resources' Time in the System

Table 5.2: Root Mean Square Error, Mean Percentage Error and Maximum Percentage Error for Predicted Values of Response Time and Power. The Max. Error Freq. is the CPU Frequency Corresponding to the Maximum Percentage Error Value.

Metric	**Benchmark**	**Number of Online CPU Cores**	**RMSE (unit)**	**Mean Percentage Error (%)**	**Max. Error (%)**	**Max. Error Freq. (GHz)**
Response Time (sec)	JPEG	4	0.39	1.22	3.40	2.15
		2	1.03	3.08	7.01	1.27
		1	2.09	3.09	5.39	1.27
	Basic Math Large	4	0.16	4.01	7.23	1.04
		2	0.20	4.97	8.56	0.88
		1	1.74	30.24	51.6	1.96
	Patricia	4	0.07	5.42	10.54	1.04
		2	0.12	3.55	16.90	0.73
		1	0.65	34.31	54.79	1.73
Power (mW)	JPEG	4	58.72	2.58	4.43	1.27
		2	58.59	2.81	8.60	1.27
		1	72.23	4.58	8.76	1.27
	Basic Math Large	4	59.65	2.76	6.86	0.88
		2	81.79	3.88	10.75	1.27
		1	66.1	4.42	9.29	1.27
	Patricia	4	81.39	1.86	4.58	2.15
		2	79.45	2.21	5.18	1.50
		1	58.05	1.62	4.61	1.27

Table 5.3: Root Mean Square Error, Mean Percentage Error and Maximum Percentage Error for Predicted Values of Energy With and Without Background Power. The Max. Error Freq. is the CPU Frequency Corresponding to the Maximum Percentage Error Value.

Metric	**Benchmark**	**Number of Online CPU Cores**	**RMSE (unit)**	**Mean Percentage Error (%)**	**Max. Error (%)**	**Max. Error Freq. (GHz)**
Energy without BG Power (J)	JPEG	4	1.75	3.12	7.17	2.15
		2	1.50	2.79	8.12	1.19
		1	3.88	4.07	10.41	2.15
	Basic Math Large	4	0.23	3.56	7.50	1.04
		2	0.25	4.13	11.48	1.19
		1	2.21	28.21	58.84	2.15
	Patricia	4	0.19	5.13	8.73	1.57
		2	0.27	3.54	14.6	0.73
		1	1.74	33.84	54.34	1.73
Energy with BG Power (J)	JPEG	4	7.31	4.12	9.18	1.96
		2	3.94	2.32	6.70	2.15
		1	23.11	5.88	7.78	1.19
	Basic Math Large	4	0.82	3.77	7.11	2.15
		2	0.88	3.21	7.91	0.88
		1	8.57	28.89	53.41	1.96
	Patricia	4	0.44	4.68	10.59	1.04
		2	0.89	6.33	17.40	0.73
		1	4.06	32.52	50.29	1.73

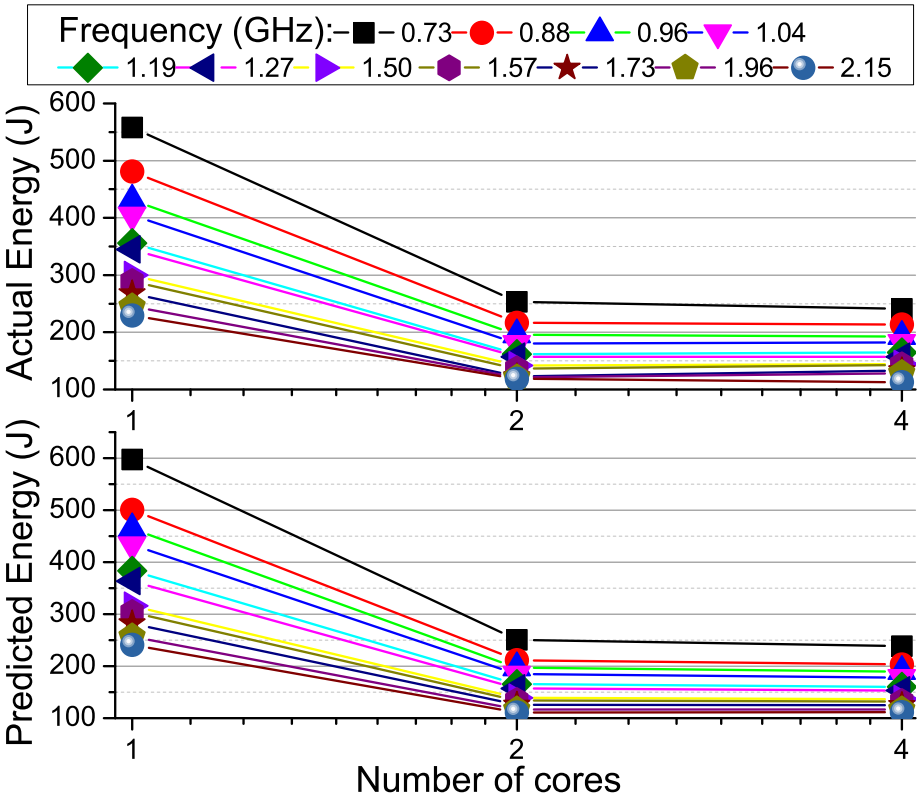


Figure 5.16: Measurement Results Showing Predicted and Actual Values for JPEG Benchmark Running on 1, 2 and 4 CPU Cores. As Shown in the Figure, DVFS May Not Give Better Savings than Adding CPU Cores to the System.

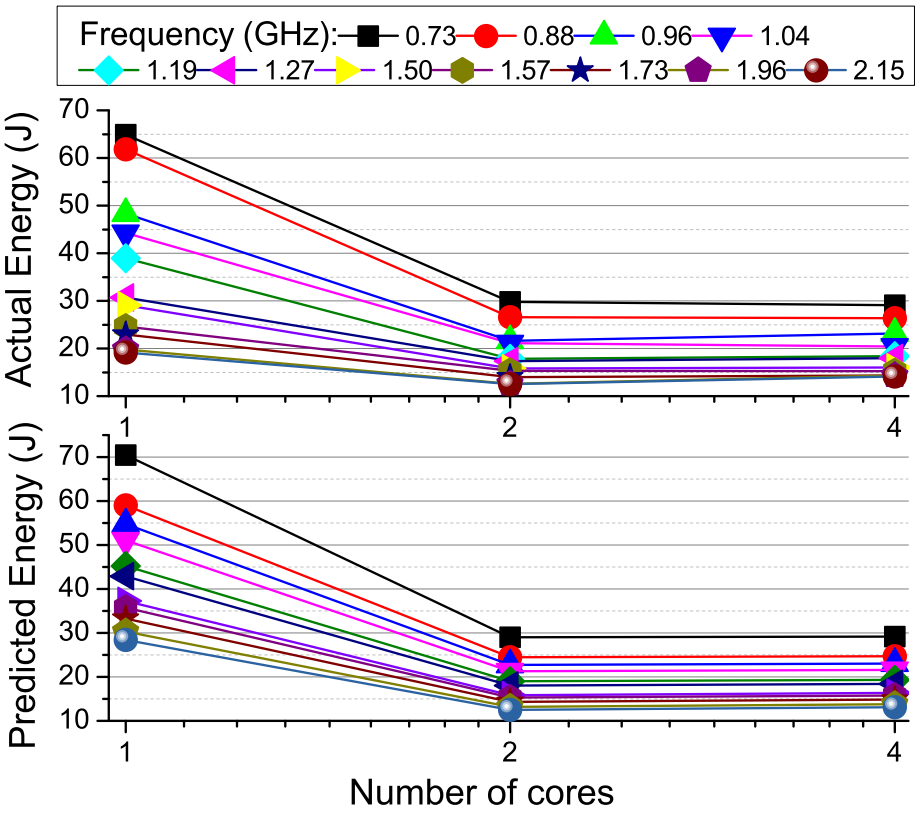


Figure 5.17: Measurement Results Showing Predicted and Actual Values for Basic Math Large Benchmark Running on 1, 2 and 4 CPU Cores. As Shown in the Figure, DVFS May Not Give Better Savings than Adding CPU Cores to the System.

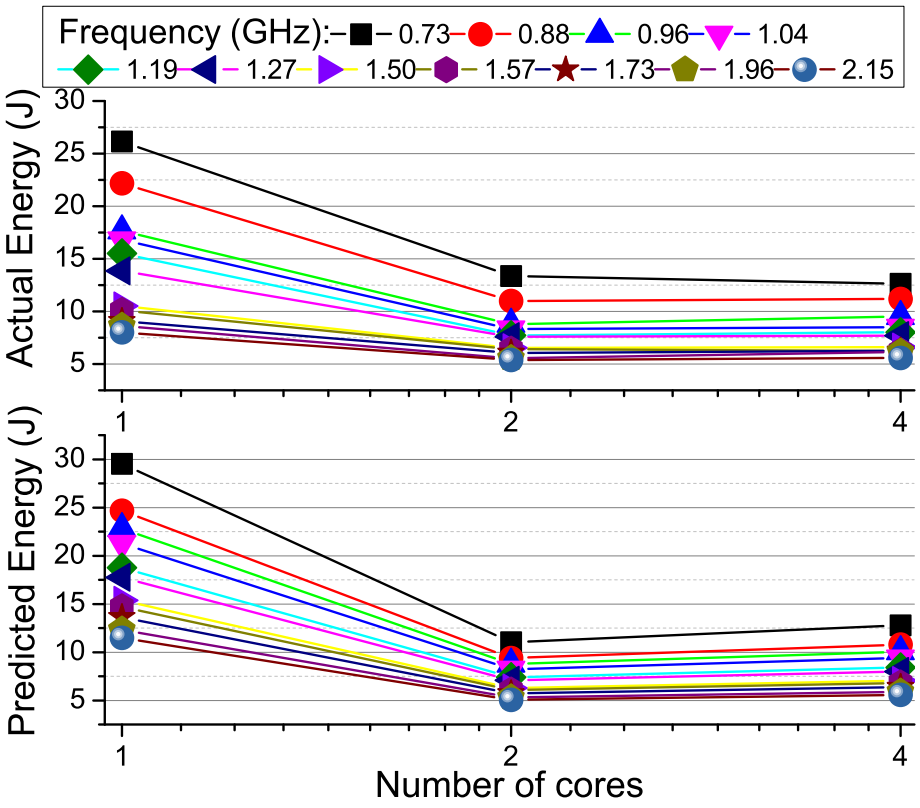


Figure 5.18: Measurement Results Showing Predicted and Actual Values for Patricia Benchmark Running on 1, 2 and 4 CPU Cores. As Shown in the Figure, DVFS May Not Give Better Savings than Adding CPU Cores to the System.

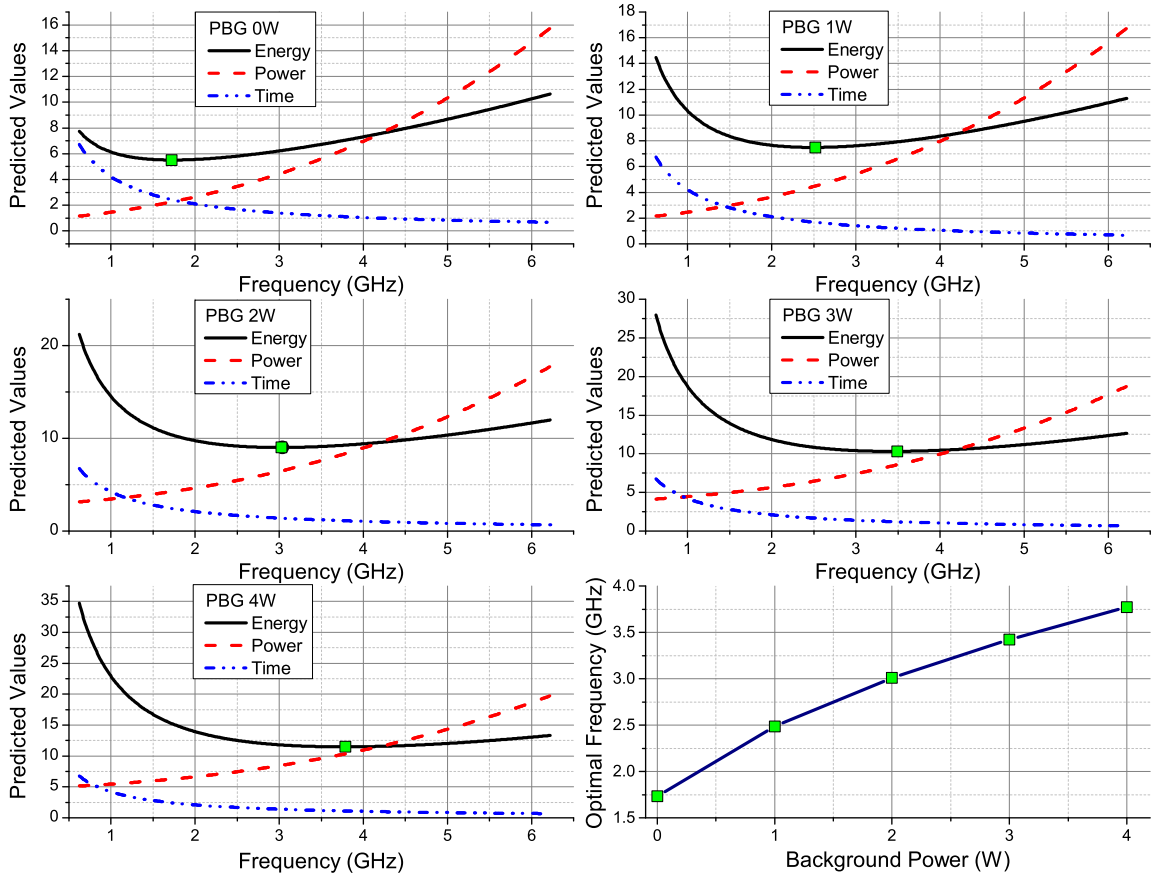


Figure 5.19: Measurement Results Showing Predicted Energy, power, and time values for Basic Math Large Benchmark Running on 4 CPU Cores with increasing Background Power. The optimal frequency of operation keeps increasing at a diminishing rate as background power is increased.

Chapter 6

CONCLUSION AND FUTURE DIRECTIONS

In this work, we develop a detailed energy optimization framework for heterogeneous mobile platforms and an experimental methodology to validate our models. The framework can be used for optimizing the total energy of a given hardware configuration. To the best of our knowledge, we are first to propose a framework that can be used for both, architectural exploration and dynamic management of resources. For example, on the one hand, computer architects can find the optimal number of CPU cores that are required to speedup a particular functionality, like voice recognition. On the other hand, application developers can find the optimal CPU frequency, they must use, to minimize energy consumption of their application, like gaming. We evaluate, that our framework can be used to give reasoning about optimization with or without introducing new resources. Also, our evaluations using mobile applications show that adding new resources, in particular more energy efficient accelerators, improves the platform energy well beyond what power and performance scaling can achieve alone. The experimental results verify our framework for scaling and addition of more CPU cores. Note, it is crucial to use a state-of-art hardware platform for the experiments to give reliable results with lesser effort. This is because, it is time consuming to implement models for all resources and make them work together seamlessly in a simulator. Moreover, the simulations would provide only an approximation and will not show the outlier behaviour seen in actual experiments. Apart from the main contributions, we also present a voltage-frequency table that can be used by researchers, who employ cubic power models for DPM controllers. Furthermore, we provide a theoretical approach to implement our framework at run time using Key Performance Indicators (KPIs) with the help of an illustrative example.

Future work: Here, we present a number of natural extensions to our work,

- We need to perform experimental validation of our framework for the case of adding an accelerator equivalent to adding CPU cores. For example, it can be shown that adding an energy efficient resource on a mobile platform will significantly improve energy savings than scaling existing CPU alone or adding homogeneous CPU core(s). Note, we could only analytically validated this result at present because, all IPs are fixed in the current platform [52]. In future, we plan to use an ARM® big.LITTLE™ architecture based ODROID-XU+E development board [27] and/or FPGAs in which it is possible to change IPs. We did not use these boards to start with because, the profiling tools for Odroid board and FPGAs are not as advanced as that of Qualcomm's MDP.

- We plan to use a more detailed tracing tool [21]. The anomalies highlighted in Chapter 5, particularly, the case of benchmarks running on one core are due to lack of good time tracing tool across all the resources. In future, we also plan to develop an algorithm that can dynamically calculate the response time of application at resource level.

- A natural extension to the theory discussed in this work on using KPIs is its implementation on a mobile platform. We plan to implement a dynamic power management scheme by modifying the on-demand governor [48].

- Discussion on area cost in this work is performed qualitatively between accelerators and CPU cores. We will focus on more detailed analysis of area cost and include it in the objective function.

REFERENCES

[1] B. M. Al-Babtain, F. J. Al-Kanderi, M. F. Al-Fahad, and I. Ahmad. A Survey on Amdahl's Law Extension in Multicore Architectures. *International Journal of New Comp. Arch. and their App.*, 3(3):30–46, 2013.

[2] G. M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of ACM spring joint computer conference*, pages 483–485, 1967.

[3] Android. Android debug bridge. `http://developer.android.com/tools/help/adb.html`.

[4] R. Z. Ayoub, U. Ogras, E. Gorbatov, Y. Jin, T. Kam, P. Diefenbaugh, and T. Rosing. Os-level power minimization under tight performance constraints in general purpose systems. In *Proc. of the Intl. Symp. on Low-power Electronics and Design*, pages 321–326, 2011.

[5] A. Bartolini, M. Cacciari, A. Tilli, L. Benini, and M. Gries. A virtual platform environment for exploring power, thermal and reliability management control strategies in high-performance multicores. In *Proceedings of the 20th symposium on Great lakes symposium on VLSI*, pages 311–316, 2010.

[6] D. Bertsekas. *Constrained optimization and Lagrange multiplier methods*. Opt. and neural comp. 1996.

[7] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood. The gem5 simulator. *SIGARCH Comput. Archit. News*, 39(2):1–7, Aug. 2011.

[8] N. L. Binkert, R. G. Dreslinski, L. R. Hsu, K. T. Lim, A. G. Saidi, and S. K. Reinhardt. The m5 simulator: Modeling networked systems. *IEEE Micro*, 26(4):52–60, 2006.

[9] 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 *Networks on Chip (NoCS), 2012 Sixth IEEE/ACM International Symposium on*, pages 35–42, 2012.

[10] A. Carroll and G. Heiser. An Analysis of Power Consumption in a Smartphone. In *USENIXATC*, pages 21–21, 2010.

[11] A. S. Cassidy and A. G. Andreou. Beyond Amdahl's Law: An Objective Function That Links Multiprocessor Performance Gains to Delay and Energy. *IEEE Transactions on Computers*, 61(8):1110–1126, 2012.

[12] J. Ceng, W. Sheng, J. Castrillon, A. Stulova, R. Leupers, G. Ascheid, and H. Meyr. A high-level virtual platform for early mpsoc software development. In *Proceedings of the 7th IEEE/ACM international conference on Hardware/software codesign and system synthesis*, pages 11–20, 2009.

[13] W. O. Cesário, D. Lyonnard, G. Nicolescu, Y. Paviot, S. Yoo, A. A. Jerraya, L. Gauthier, and M. Diaz-Nava. Multiprocessor SoC Platforms: a Component-based Design Approach. *IEEE Design & Test of Computers*, 19(6):52–63, 2002.

[14] H.-C. Chang, A. Agrawal, and K. Cameron. Energy-aware computing for android platforms. In *Energy Aware Computing (ICEAC), 2011 International Conference on*, pages 1–4, 2011.

[15] S. Cho and R. G. Melhem. Corollaries to amdahl's law for energy. *Computer Architecture Letters*, 7(1):25–28, 2008.

[16] S. Cho and R. G. Melhem. On the interplay of parallelization, program performance, and energy consumption. *IEEE Transactions on Parallel and Distributed Systems*, 21(3):342–353, 2010.

[17] Y. Cho, N. Chang, C. Chakrabarti, and S. Vrudhula. High-level power management of embedded systems with application-specific energy cost functions. In *Proceedings of the 43rd annual Design Automation Conference*, pages 568–573. ACM, 2006.

[18] B.-G. Chun and P. Maniatis. Augmented smartphone applications through clone cloud execution. In *HotOS*, volume 9, pages 8–11, 2009.

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

[20] Digby. Mobile Industry Stats@ONLINE. `http://digby.com/mobile-statistics/`.

[21] eLinux. Ftrace. `http://elinux.org/Ftrace`.

[22] S. Eyerman and L. Eeckhout. Modeling critical sections in amdahl's law and its implications for multicore design. *SIGARCH Comput. Archit. News*, 38(3):362–370, June 2010.

[23] GNU. Bash Reference Manual@ONLINE. `http://www.gnu.org/software/bash/manual/bashref.html`.

[24] O. Google. Android Jelly Bean @ONLINE. `http://www.android.com/versions/jelly-bean-4-2/`.

[25] M. Guthaus, J. Ringenberg, D. Ernst, T. Austin, T. Mudge, and R. Brown. Mibench: A free, commercially representative embedded benchmark suite. In *Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop on*, pages 3–14, Dec 2001.

[26] 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 *Workload Characterization (IISWC), 2011 IEEE International Symposium on*, pages 81–90. IEEE, 2011.

[27] Hard Kernel. Odroid. http://www.hardkernel.com/.

[28] X. he Sun and L. Ni. Another view on parallel speedup. In *Supercomputing '90., Proceedings of*, pages 324–333, Nov 1990.

[29] M. D. Hill and M. R. Marty. Amdahl's Law in the Multicore Era. *IEEE Computer*, 2008.

[30] A. Hodjat and I. Verbauwhede. Interfacing a high speed crypto accelerator to an embedded CPU. In *Proceedings of IEEE, Asilomar Conference on Signals, Systems and Computers*, 2004.

[31] Y. Huh. Future direction of power management in mobile devices. In *Solid State Circuits Conference (A-SSCC), 2011 IEEE Asian*, pages 1–4, Nov 2011.

[32] Intel Corporation. Intel© Atom™ Processor $Z2760$. http://ark.intel.com/products/70105.

[33] B. Juurlink and C. H. Meenderinck. Amdahl's law for predicting the future of multicores considered harmful. *SIGARCH Comput. Archit. News*, pages 1–9, May 2012.

[34] D. Kadjo, U. Ogras, R. Ayoub, M. Kishinevsky, and P. Gratz. Towards Platform Level Power Management in Mobile Systems. *To appear in International IEEE SoC (System-on-Chip) Conference, Sept. 2014*.

[35] K. Kim, D. Shin, Q. Xie, Y. Wang, M. Pedram, and N. Chang. Fepma: Fine-grained event-driven power meter for android smartphones based on device driver layer event monitoring. In *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014*, pages 1–6, March 2014.

[36] 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 *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*, pages 469–480. IEEE, 2009.

[37] G. Martin. Overview of the mpsoc design challenge. In *Proceedings of the 43rd annual Design Automation Conference*, pages 274–279, 2006.

[38] G. Metri, W. Shi, M. Brockmeyer, and A. Agrawal. Batteryextender: An adaptive user-guided tool for power management of mobile devices. 2014.

[39] A. Morad, T. Morad, L. Yavits, R. Ginosar, and U. Weiser. Generalized MultiAmdahl: optimization of heterogeneous multi-accelerator SoC. *IEEE Comp. Arch. Letters*, 2012.

[40] T. Mudge. Power: A First-class Architectural Design Constraint. *Computer*, 34(4):52–58, 2001.

[41] 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*, pages 111–116. IEEE, 2007.

[42] Netscape Communications Corporation, Mozilla Foundation. JavaScript @ONLINE. https://developer.mozilla.org/en-US/docs/Web/JavaScript.

[43] N. K. Nithi and A. J. de Lind van Wijngaarden. Smart Power Management For Mobile Handsets. *Bell Labs Technical Journal*, 15(4):149–168, 2011.

[44] Nomura and Gartner. Smarphone Guide @ONLINE. http://images.businessweek.com/bloomberg/pdfs/nomura_smartphone_poster_2012.pdf.

[45] U. Y. Ogras, R. Z. Ayoub, M. Kishinevsky, and D. Kadjo. Managing mobile platform power. In *Proceedings of the International Conference on Computer-Aided Design*, pages 161–162. IEEE Press, 2013.

[46] U. Y. Ogras, R. Marculescu, D. Marculescu, and E. G. Jung. Design and Management of Voltage-Frequency Island Partitioned Networks-on-Chip. *Very Large Scale Integration (VLSI) Systems, IEEE Trans. on*, 17(3):330–341, 2009.

[47] OriginLab, Northampton, MA. Origin. http://www.originlab.com/.

[48] V. Pallipadi and A. Starikovskiy. The Ondemand Governor. In *Proceedings of the Linux Symposium*, volume 2, pages 215–230, 2006.

[49] D. Pandiyan, S.-Y. Lee, and C.-J. Wu. Performance, energy characterizations and architectural implications of an emerging mobile platform benchmark suite - mobilebench. In *Workload Characterization (IISWC), 2013 IEEE International Symposium on*, pages 133–142, Sept 2013.

[50] A. Pathak, Y. C. Hu, and M. Zhang. Where is the energy spent inside my app?: fine grained energy accounting on smartphones with eprof. In *Proceedings of ACM EU conf. on Computer Systems*, pages 29–42, 2012.

[51] A. Pathak, Y. C. Hu, M. Zhang, P. Bahl, and Y.-M. Wang. Fine-grained power modeling for smartphones using system call tracing. In *Proceedings of ACM conf. on Computer systems*, 2011.

[52] Qualcomm Inc. Qualcomm Snapdragon 805 MDP/T device. https://developer.qualcomm.com/mobile-development/development-devices/snapdragon-mobile-development-platform-mdp/snapdragon-805-mdp-tablet.

[53] Qualcomm Inc. Qualcomm Trepn profiler. https://developer.qualcomm.com/mobile-development/increase-app-performance/trepn-profiler.

[54] A. Raghavan, Y. Luo, A. Chandawalla, M. Papaefthymiou, K. P. Pipe, T. F. Wenisch, and M. M. Martin. Computational sprinting. In *IEEE, High Performance Computer Architecture (HPCA)*, 2012.

[55] R. Rao, S. Vrudhula, C. Chakrabarti, and N. Chang. An optimal analytical solution for processor speed control with thermal constraints. In *Proceedings of the 2006 international symposium on Low power electronics and design*, pages 292–297. ACM, 2006.

[56] A. Roy, S. M. Rumble, R. Stutsman, P. Levis, D. Mazières, and N. Zeldovich. Energy management in mobile devices with the cinder operating system. In *Proceedings of the sixth conference on Computer systems*, pages 139–152. ACM, 2011.

[57] S. Sharifi, D. Krishnaswamy, and T. S. Rosing. PROMETHEUS: A Proactive Method for Thermal Management of Heterogeneous MPSoCs. *IEEE Transactions on CAD of IC and Systems*, 2013.

[58] H. Shen and F. Pétrot. Using amdahl's law for performance analysis of many-core soc architectures based on functionally asymmetric processors. In *Proceedings of the 24th International Conference on Architecture of Computing Systems*, ARCS'11, pages 38–49, Berlin, Heidelberg, 2011. Springer-Verlag.

[59] W.-T. Shine and C. Chakrabarti. Low-power scheduling with resources operating at multiple voltages. *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on*, 47(6):536–543, 2000.

[60] A. Shye, B. Scholbrock, and G. Memik. Into the wild: studying real user activity patterns to guide power optimizations for mobile architectures. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 168–178. ACM, 2009.

[61] A. Shye, B. Scholbrock, G. Memik, and P. A. Dinda. Characterizing and modeling user activity on smartphones. 2010.

[62] N. Vallina-Rodriguez and J. Crowcroft. Energy management techniques in modern mobile handsets. *IEEE Communications Surveys & Tutorials*, (99):1–20, 2012.

[63] W3C and WHATWG. HyperText Markup Language @ONLINE. `http://www.w3.org/html/`.

[64] D. H. Woo and H.-H. Lee. Extending Amdahl's Law for Energy-Efficient Computing in the Many-core Era. *IEEE Computer*, 2008.

[65] E. Yao, Y. Bao, G. Tan, and M. Chen. Extending amdahl's law in the multicore era. *SIGMETRICS Perform. Eval. Rev.*, 37(2):24–26, Oct. 2009.

[66] E. Yao, Y. Bao, G. Tan, and M. Chen. Extending Amdahl's law in the multicore era. *ACM SIGMETRICS Perf. Eval. Review*, 2009.

[67] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced CPU energy. In *Proceedings of IEEE, Comp. Science*, 1995.

[68] C. Yoon, D. Kim, W. Jung, C. Kang, and H. Cha. Appscope: Application energy metering framework for android smartphone using kernel activity monitoring. In *USENIX Annual Technical Conference*, pages 387–400, 2012.

[69] J. Zhuo and C. Chakrabarti. Energy-efficient dynamic task scheduling algorithms for dvs systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 7(2):17, 2008.

[70] J. Zhuo, C. Chakrabarti, and N. Chang. Energy management of dvs-dpm enabled embedded systems powered by fuel cell-battery hybrid source. In *Proceedings of the 2007 international symposium on Low power electronics and design*, pages 322–327. ACM, 2007.

[71] T. Zidenberg, I. Keslassy, and U. Weiser. MultiAmdahl: How Should I Divide My Heterogenous Chip? *IEEE, Computer Architecture Letters*, 2012.

APPENDIX A

DATA MEASUREMENT METHOD

One can follow this step-by-step procedure to obtain the data from the tablet:

1. Switch on/ reboot the tablet.
2. Check that no other applications are running in background, this is to ensure that we have less overhead while measuring power.
3. Connect the tablet to the PC over Wi-Fi using ADB [3].
4. Through ADB shell, hot-plug the device, change governors and frequency of the CPU cores. For setting to a particular frequency write to `max_scaling_freq` variable while `performance` governor is in effect.
5. In Trepn profiler enable "Show Deltas" and "Aquire Wakelock while Profiling" options and select the appropriate data points such as CPU cores, Graphics, WLAN/BT/FM, SD-Card, Codec/Wireless/GPS, GPU to be logged-in.
6. Ensure that no more than six data points are selected to keep overhead of Trepn profiler low.
7. Remove the charger/ USB cable before power profiling because it may result in inaccurate power data.
8. Now switch OFF the Wi-Fi, ADB connection will break as a result of this.
9. Start the Trepn profiler through GUI on tablet and wait for about 10 seconds (randomly selected number).
10. Turn OFF the display by pressing the power button and wait for 10 seconds.
11. Turn ON the display by again pressing the power button and wait for 10 seconds.
12. Connect to Wi-Fi and connect the tablet to PC using ADB.
13. Run the benchmark and note the start and stop time using Trepn profiler or we can use `time` command in shell to get more detailed information.
14. Now wait for the benchmark to stop and then turn OFF the display and wait for 10 more seconds.
15. Turn ON the display and wait for 10 seconds.
16. Stop profiling and save the collected data in .CSV file.

APPENDIX B

RAW DATA

Table B.1: Raw Data Table for Basic Math Large Benchmark Running with 4 CPU Cores Online. The Table Lists the Frequency, Response Time, Power and Energy Consumed in the System.

Frequency (MHz)	Time (sec)				Battery Power (mW)	Energy with PBG (J)	Average CPU Absolute Power (mW)	Average CPU Delta Power (mW)	Total CPU power (mW)	Energy without PBG (J)
	Real	User	System	CPU						
729.60	5.73	5.16	0.41	5.56	5076.16	29.07	1063.78	193.88	1257.66	7.20
883.20	4.91	4.42	0.38	4.80	5373.88	26.39	1139.96	131.86	1271.82	6.24
960.00	4.13	3.78	0.31	4.08	5612.98	23.18	1239.51	239.21	1478.72	6.11
1036.80	3.77	3.50	0.26	3.77	5412.78	20.41	1278.35	215.00	1493.36	5.63
1190.40	3.38	3.11	0.26	3.37	5439.03	18.38	1349.09	269.71	1618.80	5.47
1267.20	3.12	2.87	0.23	3.10	5750.17	17.94	1519.47	240.44	1759.92	5.49
1497.60	2.64	2.38	0.24	2.62	6074.84	16.04	1669.36	385.26	2054.61	5.42
1574.40	2.53	2.29	0.19	2.49	6012.69	15.21	1715.26	431.65	2146.91	5.43
1728.00	2.32	2.12	0.18	2.30	6181.49	14.34	1810.67	428.25	2238.92	5.19
1958.40	2.11	1.89	0.18	2.08	6786.21	14.34	1934.20	714.63	2648.83	5.60
2150.40	1.95	1.78	0.14	1.91	7256.14	14.12	2221.74	764.15	2985.89	5.81

Table B.2: Raw Data Table for Basic Math Large Benchmark Running with 2 CPU Cores Online. The Table Lists the Frequency, Response Time, Power and Energy Consumed in the System.

Frequency (MHz)	Time (sec)				Battery Power (mW)	Energy with PBG (J)	Average CPU Absolute Power (mW)	Average CPU Delta Power (mW)	Total CPU power (mW)	Energy without PBG (J)
	Real	User	System	CPU						
729.60	5.98	5.15	0.38	5.53	4978.72	29.77	902.47	151.90	1054.37	6.31
883.20	5.3	4.43	0.35	4.78	5013.32	26.57	958.88	217.50	1176.39	6.23
960.00	4.27	3.81	0.27	4.08	5060.48	21.61	1052.63	219.10	1271.74	5.43
1036.80	4.08	3.47	0.30	3.77	5184.56	21.15	1089.93	209.35	1299.29	5.30
1190.40	3.40	3.09	0.25	3.34	5244.19	17.83	1114.66	239.43	1354.09	4.60
1267.20	3.14	2.85	0.27	3.12	5527.63	17.36	1304.81	363.72	1668.53	5.24
1497.60	2.70	2.38	0.23	2.61	5843.25	15.78	1393.49	377.93	1771.42	4.78
1574.40	2.57	2.29	0.20	2.49	5969.50	15.34	1488.72	383.75	1872.48	4.81
1728.00	2.33	2.10	0.18	2.28	6000.51	13.98	1499.30	449.71	1949.01	4.54
1958.40	2.06	1.87	0.19	2.06	6110.59	12.59	1610.34	583.92	2194.26	4.52
2150.40	2.00	1.74	0.18	1.92	6258.99	12.52	1757.55	522.32	2279.87	4.56

Table B.3: Raw Data Table for Basic Math Large Benchmark Running with 1 CPU Core Online. The Table Lists the Frequency, Response Time, Power and Energy Consumed in the System.

Frequency (MHz)	Time (sec)				Battery Power (mW)	Energy with PBG (J)	Average CPU Absolute Power (mW)	Average CPU Delta Power (mW)	Total CPU power (mW)	Energy without PBG (J)
	Real	User	System	CPU						
729.60	13.52	5.19	0.38	5.57	4800.61	64.90	845.51	34.81	880.32	11.90
883.20	12.86	4.45	0.34	4.79	4805.11	61.79	860.00	54.70	914.71	11.76
960.00	9.83	3.76	0.37	4.13	4919.26	48.36	940.54	56.30	996.84	9.80
1036.80	8.96	3.55	0.23	3.78	4942.36	44.28	996.59	99.90	1096.49	9.82
1190.40	7.47	3.03	0.29	3.32	5215.86	38.96	1013.31	138.26	1151.58	8.60
1267.20	5.88	2.79	0.31	3.10	5223.34	30.71	1171.79	127.20	1298.99	7.64
1497.60	5.5	2.44	0.23	2.67	5294.54	29.12	1256.45	137.39	1393.85	7.67
1574.40	4.69	2.3	0.2	2.50	5241.98	24.58	1290.46	135.08	1425.54	6.69
1728.00	4.36	2.07	0.22	2.29	5256.37	22.92	1345.56	177.37	1522.93	6.64
1958.40	3.64	1.9	0.14	2.04	5427.29	19.76	1405.85	213.82	1619.66	5.90
2150.40	3.38	1.76	0.13	1.89	5658.65	19.13	1496.70	186.44	1683.13	5.69

Table B.4: Raw Data Table for JPEG Benchmark Running with 4 CPU Cores Online. The Table Lists the Frequency, Response Time, Power and Energy Consumed in the System.

Frequency (MHz)	Time (sec)				Battery Power (mW)	Energy with PBG (J)	Average CPU Absolute Power (mW)	Average CPU Delta Power (mW)	Total CPU power (mW)	Energy without PBG (J)
	Real	User	System	CPU						
729.60	46.71	40.58	5.96	46.54	5154.74	240.78	993.85	168.82	1162.66	54.31
883.20	39.84	33.69	5.73	39.42	5365.12	213.75	1141.74	206.01	1347.74	53.69
960.00	35.86	30.59	4.72	35.31	5376.06	192.79	1253.88	171.37	1425.24	51.11
1036.80	32.83	28.07	4.46	32.53	5554.66	182.36	1281.28	194.61	1475.90	48.45
1190.40	28.95	24.90	3.82	28.72	5682.91	164.52	1353.48	228.25	1581.73	45.79
1267.20	26.55	23.49	3.45	26.94	5925.09	157.31	1531.74	269.30	1801.04	47.82
1497.60	23.06	19.93	2.72	22.65	6266.53	144.51	1675.62	400.69	2076.31	47.88
1574.40	22.15	19.06	2.83	21.89	6462.37	143.14	1725.97	400.48	2126.45	47.10
1728.00	19.93	17.09	2.75	19.84	6686.54	133.26	1814.32	552.06	2366.38	47.16
1958.40	18.54	15.35	2.73	18.08	6920.20	128.30	1939.96	631.17	2571.13	47.67
2150.40	17.01	14.30	2.56	16.86	6632.21	112.81	2307.53	705.69	3013.22	51.25

Table B.5: Raw Data Table for JPEG Benchmark Running with 2 CPU Cores Online. The Table Lists the Frequency, Response Time, Power and Energy Consumed in the System.

Frequency (MHz)	Time (sec)				Battery Power (mW)	Energy with PBG (J)	Average CPU Absolute Power (mW)	Average CPU Delta Power (mW)	Total CPU power (mW)	Energy without PBG (J)
	Real	User	System	CPU						
729.60	51.4	40.36	6.4	46.76	4922.54	253.02	899.25	119.97	1019.22	52.39
883.20	42.91	33.62	5.8	39.42	5047.18	216.57	968.50	141.68	1110.18	47.64
960.00	37.93	30.80	4.67	35.47	5150.41	195.36	1051.27	170.14	1221.40	46.33
1036.80	34.90	28.64	4.27	32.91	5172.39	180.52	1083.23	170.47	1253.71	43.75
1190.40	30.41	24.87	4.05	28.92	5310.33	161.49	1162.91	186.39	1349.30	41.03
1267.20	27.76	23.24	3.68	26.92	5655.45	157.00	1315.75	296.97	1612.72	44.77
1497.60	24.16	19.42	3.67	23.09	5867.14	141.75	1439.80	317.57	1757.37	42.46
1574.40	22.93	18.90	3.30	22.20	5942.62	136.26	1471.26	400.40	1871.66	42.92
1728.00	20.90	17.40	2.93	20.33	5878.69	122.86	1520.54	463.23	1983.77	41.46
1958.40	19.03	15.32	2.88	18.20	6357.86	120.99	1632.30	556.24	2188.54	41.65
2150.40	17.66	14.04	2.66	16.70	6719.51	118.67	1754.13	748.15	2502.27	44.19

Table B.6: Raw Data Table for JPEG Benchmark Running with 1 CPU Core Online. The Table Lists the Frequency, Response Time, Power and Energy Consumed in the System.

Frequency (MHz)	Time (sec)				Battery Power (mW)	Energy with PBG (J)	Average CPU Absolute Power (mW)	Average CPU Delta Power (mW)	Total CPU power (mW)	Energy without PBG (J)
	Real	User	System	CPU						
729.60	122.59	41.13	5.76	46.89	4551.00	557.91	832.36	51.68	884.04	108.37
883.20	102.11	34.04	5.43	39.47	4710.84	481.02	856.75	66.45	923.20	94.27
960.00	91.52	30.68	4.48	35.16	4711.93	431.24	941.93	74.54	1016.47	93.03
1036.80	84.55	28.43	4.04	32.47	4787.96	404.82	984.28	108.37	1092.65	92.38
1190.40	73.32	24.93	3.72	28.65	4852.31	355.77	1000.47	119.31	1119.77	82.10
1267.20	67.24	23.31	3.31	26.62	5128.28	344.83	1169.00	133.41	1302.42	87.57
1497.60	57.41	19.55	3.07	22.62	5226.52	300.05	1246.90	177.93	1424.83	81.80
1574.40	54.52	18.82	2.76	21.58	5295.19	288.69	1279.75	208.70	1488.45	81.15
1728.00	49.77	17.16	2.61	19.77	5382.43	267.88	1334.97	203.35	1538.32	76.56
1958.40	44.24	15.36	2.25	17.61	5535.85	244.91	1415.44	219.26	1634.70	72.32
2150.40	40.51	13.75	2.35	16.10	5654.94	229.08	1442.95	262.47	1705.42	69.09

Table B.7: Raw Data Table for Patricia Benchmark Running with 4 CPU Cores Online. The Table Lists the Frequency, Response Time, Power and Energy Consumed in the System.

Frequency (MHz)	Time (sec)				Battery Power (mW)	Energy with PBG (J)	Average CPU Absolute Power (mW)	Average CPU Delta Power (mW)	Total CPU power (mW)	Energy without PBG (J)
	Real	User	System	CPU						
729.60	1.93	1.8	0.07	1.87	6545.81	12.63	2508.34	58.82	2567.16	4.95
883.20	1.67	1.54	0.11	1.65	6706.25	11.20	2596.52	105.38	2701.89	4.51
960.00	1.36	1.27	0.09	1.36	6990.37	9.51	2746.78	158.31	2905.09	3.95
1036.80	1.24	1.17	0.07	1.24	6857.73	8.50	2779.67	198.53	2978.20	3.69
1190.40	1.14	1.07	0.05	1.12	7027.53	8.01	2858.76	177.77	3036.53	3.46
1267.20	1.04	0.95	0.06	1.01	7401.30	7.70	3052.33	174.16	3226.50	3.36
1497.60	0.88	0.78	0.08	0.86	7505.16	6.60	3195.44	314.96	3510.40	3.09
1574.40	0.84	0.78	0.06	0.84	7448.64	6.26	3157.93	366.71	3524.63	2.96
1728.00	0.79	0.72	0.04	0.76	7907.95	6.25	3346.68	431.67	3778.35	2.98
1958.40	0.72	0.64	0.06	0.70	8549.40	6.16	3560.47	513.74	4074.22	2.93
2150.40	0.66	0.60	0.06	0.66	8449.11	5.58	3711.02	943.46	4654.47	3.07

Table B.8: Raw Data Table for Patricia Benchmark Running with 2 CPU Cores Online. The Table Lists the Frequency, Response Time, Power and Energy Consumed in the System.

Frequency (MHz)	Time (sec)				Battery Power (mW)	Energy with PBG (J)	Average CPU Absolute Power (mW)	Average CPU Delta Power (mW)	Total CPU power (mW)	Energy without PBG (J)
	Real	User	System	CPU						
729.60	2.07	1.8	0.07	1.87	6461.09	13.37	2388.87	91.54	2480.41	5.13
883.20	1.66	1.54	0.1	1.64	6618.46	10.99	2450.61	123.77	2574.38	4.27
960.00	1.33	1.26	0.05	1.31	6598.54	8.78	2531.62	148.15	2679.78	3.56
1036.80	1.24	1.15	0.07	1.22	6725.63	8.34	2579.82	208.01	2787.83	3.46
1190.40	1.14	1.02	0.08	1.10	6771.68	7.72	2606.92	138.42	2745.33	3.13
1267.20	1.05	0.92	0.07	0.99	7230.14	7.59	2787.06	207.98	2995.03	3.14
1497.60	0.90	0.78	0.08	0.86	7280.90	6.55	2868.13	347.89	3216.02	2.89
1574.40	0.87	0.80	0.01	0.81	7361.39	6.40	2916.32	225.22	3141.55	2.73
1728.00	0.79	0.69	0.07	0.76	7671.56	6.06	3004.80	288.96	3293.76	2.60
1958.40	0.72	0.65	0.04	0.69	7708.44	5.55	3111.52	350.24	3461.76	2.49
2150.40	0.68	0.61	0.03	0.64	7932.89	5.39	3183.17	386.16	3569.33	2.43

Table B.9: Raw Data Table for Patricia Benchmark Running with 1 CPU Core Online. The Table Lists the Frequency, Response Time, Power and Energy Consumed in the System.

Frequency (MHz)	Time (sec)				Battery Power (mW)	Energy with PBG (J)	Average CPU Absolute Power (mW)	Average CPU Delta Power (mW)	Total CPU power (mW)	Energy without PBG (J)
	Real	User	System	CPU						
729.60	4.22	1.75	0.09	1.84	6201.25	26.17	2331.98	36.36	2368.34	9.99
883.20	3.58	1.54	0.09	1.63	6195.70	22.18	2331.02	26.56	2357.58	8.44
960.00	2.84	1.25	0.06	1.31	6222.88	17.67	2425.56	52.31	2477.87	7.04
1036.80	2.63	1.17	0.03	1.20	6390.50	16.81	2462.02	43.46	2505.49	6.59
1190.40	2.43	1.04	0.06	1.10	6381.05	15.51	2490.94	32.11	2523.06	6.13
1267.20	2.09	0.95	0.03	0.98	6631.83	13.86	2654.76	103.05	2757.81	5.76
1497.60	1.55	0.79	0.05	0.84	6777.26	10.50	2735.48	102.91	2838.39	4.40
1574.40	1.49	0.78	0.03	0.81	6768.42	10.08	2751.96	104.50	2856.46	4.26
1728.00	1.31	0.70	0.04	0.74	6920.73	9.07	2807.03	122.94	2929.96	3.84
1958.40	1.25	0.63	0.04	0.67	6908.97	8.64	2906.17	216.21	3122.37	3.90
2150.40	1.12	0.60	0.03	0.63	7141.95	8.00	2971.94	217.14	3189.07	3.57

APPENDIX C

SCRIPT FOR HOT-PLUGGING AND CHANGING CPU FREQUENCY

```
#!/bin/bash
# Description:
#
# This script can be used for:
# 1) CPU Hot-plugging - turning off CPUs
# 2) Changing governors
#
# RUN THIS SCRIPT WITH ROOT PERMISSION
# dos2unix is usually required in case the file is edited in MS Windows.
#
# Number of arguments in the command line
if [ "$#" -gt 0 ]
then
    echo "Your command line contains $# arguments"
else
    echo "Your command line contains no arguments"
fi
# Displaying the parameters
echo "Positional Parameters"
echo '$0 = ' $0
echo '$1 = ' $1
echo '$2 = ' $2
echo '$3 = ' $3
echo '$4 = ' $4
echo '$5 = ' $5
echo '$6 = ' $6
# The arguments to this script will be:
#  + CPU0 status - 1 is ON, 0 is OFF
#  + CPU1 status - 1 is ON, 0 is OFF
#  + CPU2 status - 1 is ON, 0 is OFF
#  + CPU3 status - 1 is ON, 0 is OFF
#  + Governor
#  + CPU0 custom frequency in KHz
cpu0_status=$1
cpu1_status=$2
cpu2_status=$3
cpu3_status=$4
new_governer=$5
cpu0_freq=$6
#CPU_STATUS_PATH
C_PATH=/sys/devices/system/cpu
#
# Switch ON or OFF the CPU cores depending on cpuN_status
#
# To transfer the control of the cores to user, stop mpdecision
# This disables overwriting of the governors at each display off state.
stop mpdecision
echo "mpdecision stopped. Now YOU have the power to govern!"
# Writing the cpuN_status
echo $cpu0_status > $C_PATH/cpu0/online
echo $cpu1_status > $C_PATH/cpu1/online
echo $cpu2_status > $C_PATH/cpu2/online
echo $cpu3_status > $C_PATH/cpu3/online
# Checking the cpuN_status
echo "Status of the CPUs in respective order are: "
cat $C_PATH/cpu0/online
```

```
cat $C_PATH/cpu1/online
cat $C_PATH/cpu2/online
cat $C_PATH/cpu3/online
#
# Switch ON or OFF the CPU cores depending on cpuN_status
#
echo $cpu0_freq > $C_PATH/cpu0/cpufreq/scaling_max_freq
# Check the value of CPU 0 frequency
echo "Current CPU0 frequency is "
cat $C_PATH/cpu0/cpufreq/scaling_max_freq
# Change the governor to "performance" such that the CPU always works at
    MAX frequency
echo $new_governer > $C_PATH/cpu0/cpufreq/scaling_governor
# Check for the governor
echo "Current governor is "
cat $C_PATH/cpu0/cpufreq/scaling_governor
# Check again
echo "Checking for frequency and governor again"
sleep 3
echo "Current CPU0 frequency is "
cat $C_PATH/cpu0/cpufreq/scaling_max_freq
echo "Current governor is "
cat $C_PATH/cpu0/cpufreq/scaling_governor
```