LIGHTWEIGHT ENERGY ESTIMATION FRAMEWORK FOR SMART-PHONE APPLICATIONS USING STATIC ANALYSIS

University of Malaya

RAJA WASIM AHMAD

FACULTY OF COMPUTER SCIENCE AND
INFORMATION TECHNOLOGY
UNIVERSITY OF MALAYA
KUALA LUMPUR

2017

LIGHTWEIGHT ENERGY ESTIMATION FRAMEWORK FOR SMART-PHONE APPLICATIONS USING STATIC ANALYSIS

RAJA WASIM AHMAD

THESIS SUBMITTED IN FULFILMENT
OF THE REQUIREMENTS
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

FACULTY OF COMPUTER SCIENCE AND
INFORMATION TECHNOLOGY
UNIVERSITY OF MALAYA
KUALA LUMPUR

2017

University of Malaya

UNIVERSITI MALAYA

ORIGINAL LITERARY WORK DECLARATION

Name of Candidate:Raja Wasim Ahmad

Registration/Matrix No.:WHA130017

Name of Degree:Doctorate of Philosphy

Title of Project Paper/Research Report/Dissertation/Thesis ("Lightweight Energy Estimation Framework For Smart-phone Applications Using Static Analysis"):

Field of Study: Mobile Cloud Computing

I do solemnly and sincerely declare that:

(1) I am the sole author/writer of this Work;
(2) This work is original;
(3) Any use of any work in which copyright exists was done by way of fair dealing and for permitted purposes and any excerpt or extract from, or reference to or reproduction of any copyright work has been disclosed expressly and sufficiently and the title of the Work and its authorship have been acknowledged in this Work;
(4) I do not have any actual knowledge nor do I ought reasonably to know that the making of this work constitutes an infringement of any copyright work;
(5) I hereby assign all and every rights in the copyright to this Work to the University of Malaya ("UM"), who henceforth shall be owner of the copyright in this Work and that any reproduction or use in any form or by any means whatsoever is prohibited without the written consent of UM having been first had and obtained;
(6) I am fully aware that if in the course of making this Work I have infringed any copyright whether intentionally or otherwise, I may be subject to legal action or any other action as may be determined by UM.

Candidate's Signature   Date

Subscribed and solemnly declared before,

Witness's Signature   Date

Name:
Designation:

University of Malaya

ABSTRACT

Recently, the preference of users has shifted the computational platform to resource constrained smart-phone devices as users prefer to work while on the go. The shift of information access paradigm on smart-phone devices demand high functionality applications to enrich user experience. However, increasing applications functionality requires more smart-phone resources. As a result, smart-phone battery consumption increases. Smart-phone application energy estimation investigates energy consumption behavior of smart-phone applications at diversified granularity levels when it is run on the smart-phone device. Traditional energy estimation schemes consider smart-phone component's power measurement or code analysis methods for energy estimation of smart-phone applications. Code analysis based methods use energy cost of operations within an application to estimate energy consumption. However, smart-phone applications are non-deterministic in nature. Therefore, traditional code analysis based energy estimation schemes run the smart-phone application to record the execution paths in offline mode to estimate its energy consumption. However, running application on hardware platform inefficiently utilizes underlying hardware resources that lead to extended estimation time and energy estimation overhead. To overcome this issue, this study proposes a lightweight 2-tier static analysis based energy estimation framework to minimize high energy overhead of dynamic analysis based energy estimation methods. The proposed framework, called Static analysis based lightweight energy estimation framework (SA-LEEF), proposes storage location analyzer, ARM-IS energy profile as service, and weighted probability based execution paths estimation to handle non-deterministic nature of smart-phone applications. Moreover, the proposed framework considers the energy overhead due to cache eviction during concurrent programs execution on the smart-

University of Malaya

phone device to present more realistic application execution environment for energy estimation. It also considers user system interaction to input required data during application execution on the smart-phone device to improve the energy estimation accuracy. The proposed framework empowers application developers to estimate energy consumption at source code line, functions, execution paths, and application granularity. The proposed study has performed experiments on Google Nexus One smart-phone device to highlight the effectiveness of SA-LEEF framework. The experiments revealed that SA-LEEF has minimized energy estimation time of dynamic analysis methods by 98% for benchmark applications. In terms of energy overhead, SA-LEEF consumes up to 97% less energy than dynamic analysis based energy estimation method. The accuracy of SA-LEEF is up to 88% compared to external physical measurement method. It is also noticed that SA-LEEF consumes 58% less CPU and 97% lower RAM storage during energy estimation of a smart-phone application. SA-LEEF assist developers investigating energy consumption behavior of their application at earlier development stages as it estimates energy consumption based on fine granular instruction energy cost.

University of Malaya

ABSTRAK

Keutamaan pengguna telah beralih platform pengiraan untuk sumber peranti telefon pintar dikekang sebagai pengguna lebih suka bekerja di luar pejabat. Selain keperluan platform pengkomputeran, permintaan pengguna untuk fungsi permohonan tinggi juga semakin meningkat. Walau bagaimanapun, peningkatan fungsi aplikasi memerlukan lebih banyak sumber telefon pintar untuk memperkayakan pengalaman pengguna. Hasilnya, bateri telefon pintar kenaikan kadar penggunaan. Smartphone tenaga permohonan anggaran menyiasat tingkah laku penggunaan tenaga aplikasi telefon pintar di granularity yang pelbagai apabila ia berjalan pada telefon pintar sumber dikekang. skim anggaran tenaga tradisional mempertimbangkan telefon pintar pengukuran komponen kuasa atau analisis kod kaedah untuk anggaran tenaga aplikasi telefon pintar. Analisis kod anggaran tenaga berasaskan menggunakan kos tenaga operasi dalam permohonan untuk menganggarkan penggunaan tenaga. Walau bagaimanapun, aplikasi telefon pintar yang bukan bersifat deterministik dalam alam semula jadi. Oleh itu, berdasarkan skim anggaran tenaga analisis kod tradisional menjalankan aplikasi telefon pintar untuk merakam laluan pelaksanaan dalam mod luar talian untuk menganggarkan penggunaan tenaga. Walau bagaimanapun, permohonan yang berjalan pada peranti telefon pintar tidak cekap menggunakan sumber perkakasan asas yang membawa kepada anggaran masa yang panjang dan overhed anggaran tenaga. Untuk mengatasi isu ini, kajian ini mencadangkan dua peringkat statik kerangka anggaran berdasarkan analisis ringan untuk mengurangkan overhed tenaga tinggi berdasarkan kaedah anggaran tenaga analisis dinamik. rangka kerja yang dicadangkan, yang dipanggil SA-LEEF, mencadangkan lokasi penyimpanan penganalisis, ARM-ISA profil tenaga perkhidmatan, dan kebarangkalian wajaran berdasarkan jalan pelaksanaan anggaran untuk mengendalikan

University of Malaya

bukan bersifat berketentuan aplikasi telefon pintar. Rangka kerja yang dicadangkan memberi kuasa kepada pemaju aplikasi untuk menganggarkan penggunaan tenaga di garisan kod sumber, fungsi, laluan pelaksanaan, dan tahap permohonan butiran. Kami melakukan eksperimen masa nyata pada Google Nexus peranti Satu telefon pintar untuk menyerlahkan keberkesanan SA-LEEF. Eksperimen menunjukkan bahawa SA-LEEF telah dikurangkan tenaga masa anggaran kaedah analisis dinamik dengan 98 % untuk aplikasi penanda aras. Dari segi overhead tenaga, SA-LEEF menggunakan sehingga tenaga 97 % kurang daripada analisis dinamik anggaran tenaga berasaskan. Ketepatan SA-LEEF adalah sehingga 88 % tepat berbanding kaedah pengukuran fizikal luaran. Ia juga mendapati bahawa SA-LEEF menggunakan 58 % kurang CPU dan kapasiti RAM 97 % lebih rendah semasa anggaran tenaga aplikasi telefon pintar. Sebagai menganggarkan tenaga di peringkat kod berbutir halus, SA-LEEF membantu pemaju dalam menyiasat tingkah laku penggunaan tenaga permohonan mereka di pelbagai peringkat pembangunan awal.

University of Malaya

ACKNOWLEDGEMENTS

ALLAH Almighty provides the courage, knowledge, and resources to every human being in this world. I am thankful to ALLAH Almighty for blessing me in every form of human quality such that I have reached this point of life and completed my Ph.D. thesis.

I am thankful to my supervisors, family, and friends who have supported and encouraged me through difficult times of life. I am highly thankful to my supervisors, Prof. Dr. Abdullah Gani and Assoc. Prof. Dr. Siti Hafizah Ab Hamid, who took me under their guidance and provided me their support for the completion of my Ph.D. A special big thanks to Dr. Anjum Naveed for his kind guidance and continuous support. I am also thankful to Prof. Joel J. P. C. Rodrigues and Assoc. Prof. Dr. Sajjad A. Madani for their valuable inputs.

I would also like to thank my fellow lab mates from whom I have consulted on various technical issues. In particular, I would like to thank Dr.Muhammad Shiraz, Dr. Saied Abolfazli, Dr.Junaid Shuja, Misbah Liaqat, Raja Sehrab Bashir, Abdullah Yousafzei, Ibrar Yaqoob, Khubaib Amjad Alam, and Zakira Inayat. My family has been a great source of inspiration for me. I am also thankful for their support and prayers. My university colleagues have been the great source of encouragement. In particular, I would thank Irfan Tarar, Sardar Nauhman Aslam, Tahir Maqsood, Faisal Rehman, Raja Adeel Ahmad, and Saad Mustafa for their prayers and support. I am also thankful to the FSKTM support staff at WISMA RND and University for lending their support and resources.

Last but not the least, I am thankful to the BrightSpark Program for financially supporting me throughout my Ph.D. tenure. Their strict KPI's were a blessing in disguise that enabled me to publish multiple Tier-1 articles as the first author.

University of Malaya

TABLE OF CONTENTS

University of Malaya

University of Malaya

University of Malaya

University of Malaya

University of Malaya

LIST OF FIGURES

University of Malaya

University of Malaya

LIST OF TABLES

University of Malaya

LIST OF APPENDICES

University of Malaya

CHAPTER 1: INTRODUCTION

1.1 Introduction

Nowadays, with the proliferation of portable devices, the energy efficient system design has become a must to meet requirement for recent resource constrained smart-phone devices. The ever growing smart-phone user's demands encourage application developers to enrich the legacy applications that as a result increase its computational and communication cost. The increase in computational and communication cost of an application noticeably augments its battery consumption rate. Among all smart-phone applications, on-demand video (Abolfazli, Sanaei, Gani, Xia, & Yang, 2014), multi-agent based distributed games, pedestrian tracking (Wang, Nevatia, & Yang, 2015), and context-aware advertisement services (Autili, Cortellessa, Di Benedetto, & Inverardi, 2015) are the most energy consuming services. The inherent features of these services such as portability, ubiquity, offline usability, and context sensing, considerably increase the energy demands of processors when it is executed on smart-phone devices (Murmuria, Stavrou, Barbará, & Fleck, 2015; Peltonen, Lagerspetz, Nurmi, & Tarkoma, 2015). To efficiently utilize smart-phone's power budget, software optimization minimizes inter-components interaction among smart-phone modules to increase battery's lifetime. Software optimization exploits energy estimation methods to identify the battery critical part within a smart-phone application (Aleti, Buhnova, Grunske, Koziolek, & Meedeniya, 2013; Hao, Li, Halfond, & Govindan, 2013; R. W. Ahmad, Gani, Hamid, Xia, & Shiraz, 2015; Q. Lu et al., 2016).

Smart-phone application energy estimation identifies energy demand of an application. Energy demands of an application highly depend on the degree of interaction among smart-phone components during application execution (R. W. Ahmad

University of Malaya

et al., 2015; Barroso & Hölzle, 2007). Smart-phone components such as CPU, Wi-Fi, memory, blue-tooth, and network radio, are the utmost energy consuming components (Anand, Manikopoulos, Jones, & Borcea, 2007). The power consumption for each smart-phone components mainly depend on, (a) power consumption at idle state, (b) usage level, (c) number of power states, (d) transitions among power states, and (e) power consumption associated with each transition. Within a smart-phone device, activities of an application trigger smart-phone components to perform a specific task that as a result consumes battery charge (Dementyev, Hodges, Taylor, & Smith, 2013; Hamzaoui, Benzekri, Grimaud, Berrajaa, & Azizi, 2016). Smart-phone application energy estimation weighs application energy consumption at unlike granularity levels such as source code line, routines, threads, and processes. Fine granular source code line energy cost based estimation creates an opportunity for application developers to investigate application energy consumption at earlier development stages (Bhattacharya, Blunck, Kjærgaard, & Nurmi, 2015; Hans, Burgstahler, Mueller, Zahn, & Stingl, 2015). This research work addresses the challenge of estimating energy consumption of smart-phone applications based on the energy cost of operations within a smart-phone application.

This chapter describes theoretical energy estimation framework and states motivations for the proposed work. It presents the statement of the problem, states the research objectives, and highlights the research methodology to carry proposed research. The organization of this chapter is as follows. Section 1.2 discusses some preliminary background on smart-phone application energy consumption estimation. Section 1.3 highlights main motivations to carry out proposed research. Section 1.4 describes the statement of the problem and presents issues related to the highlighted problem. Section 1.5 states objectives of the proposed research whereas Section 1.6 presents the proposed methodology. Section 1.8 sketches the layout of this thesis.

University of Malaya

1.2 Background

Technological advancements in Information Technology (IT) sector has shaped a new computing environment where the user prefers to perform their social, entertainment, and business activities while on the go. The new computing platform, called mobile computing, considers smart-phone and mobile devices to offer services ubiquitously. Smart-phones are evolved from personal digital assistants (PDA) and perform many of the functions of desktop computers. Smart-phone devices are equipped with high storage capacity and computational power to run resource moderate applications (Koudounas, n.d.; Pejovic & Musolesi, 2015). However, due to high demands of smart-phone users, resource requirements of many of emerging smart-phone applications (context aware services) are increased. Emerging smart-phone applications enrich the user experience by frequently triggering smart-phone sensors such as GPS, compass, accelerometer, and wireless radios to offer desired context-aware services (A. U. R. Khan, Othman, Xia, & Khan, 2015). The small size, wireless access medium, resource constraints, and mobile nature of smart-phone devices require the lightweight design of applications. Smart-phone application energy estimation assists application developers to develop lightweight applications.

Traditional energy estimation methods follow dynamic analysis approaches to estimate energy consumption of a smart-phone application. A dynamic analysis approach exploits power models for software operations or smart-phone components to estimate energy consumption of an application. A power model consists of a set of parameters that affects the total energy consumption of smart-phone application (Kjærgaard & Blunck, 2011; Do, Rawshdeh, & Shi, 2009). For smart-phone components power measurement based energy estimation, dynamic analysis approach runs the application on the smart-phone device to profile power state of smart-

University of Malaya

phone components. The power state of a smart-phone component influences the power consumption of smart-phone device. The power state of a smart-phone component depends on the type of workload running on a smart-phone device. For instance, Wi-Fi power state is low if the transmitter transmits or receives less than 8 packets in one second. Alternatively, the power state is high if Wi-Fi transmitter transmits or receives more than 15 packets in one second (L. Zhang et al., 2010a). After smart-phone components power state profiling, it employs power models of smart-phone components to estimate energy consumption of target smart-phone application.

Software operations based energy estimation methods (also known as code analysis based methods) consider execution cost of software activities to estimate energy consumption of a smart-phone application. Software operations based energy estimation associates energy cost to each operation within the smart-phone application for energy estimation (Hao et al., 2013; Hao, Li, Halfond, & Govindan, 2012a, 2012b). The execution behavior of smart-phone applications is non-deterministic by nature. To handle this issue, traditional software operations based energy estimation methods run the application on the smart-phone device to record run-time execution behavior (execution paths) of the application for energy estimation. Also, the storage location of instructions and data in multi-level memory hierarchy affects total energy estimation accuracy. Considering heavyweight nature of traditional dynamic analysis based schemes, there is a need to propose a lightweight energy estimation framework for efficient resource utilization.

1.3 Motivation

Nowadays, smart-phones are replacing desktop servers as preferences of the user for computing platform have changed. In last three years, the trend in the rise of the

University of Malaya

number of smart-phone users has rapidly increased as shown in Fig. 1.1. It is estimated that in 2016 smart-phone users in the world have reached to 1.82billions. The main motivation for significant growth in smart-phone usage is, (a) easy accessibility, (b) improved usability, and (c) attractive application features. The smart-phone market is generating a significant amount of revenue due to increase in the fame of smart-phone devices. According to the info-graphic report, due to rise in application downloading rate, by 2017 the smart-phone application market will generate 77$ billion worth of revenue (Chaffey, 2016). However, despite this tremendous hype in smart-phone popularity, still, their usage is affected by the design of the battery. Smart-phone devices require frequent battery charging and consume a significant amount of world electricity budget. According to Barry Fischer report (Fischer, 2012), the amount of energy to charge iPhone 5 smart-phones in the world is equivalent to the total energy usage of 54,000 US households for one year. This report stated that iPhone 5 electricity demand per year is 3.5kWh-4.9KWh. Also, the monetary cost to charge one iPhone 5 smart-phone device per year is 0.41$. Energy estimation is one of the ways that help to effectively utilize a smart-phone battery to augment device battery lifetime to minimize the total electricity budget of the world.

Smart-phone application energy estimation provides the basis for green computing within smart-phone devices. During application development process, developers usually consider maintainability, complexity, usability, and understandability as the performance measurement metrics for their applications. It is estimated that 80% of the application developers are unaware of green software development strategies (Pang, Hindle, Adams, & Hassan, 2015). Energy estimation of smart-phone applications provide feedback to the application developers to re-consider their application design for effective battery resource usage. Smart-phone applica-

University of Malaya

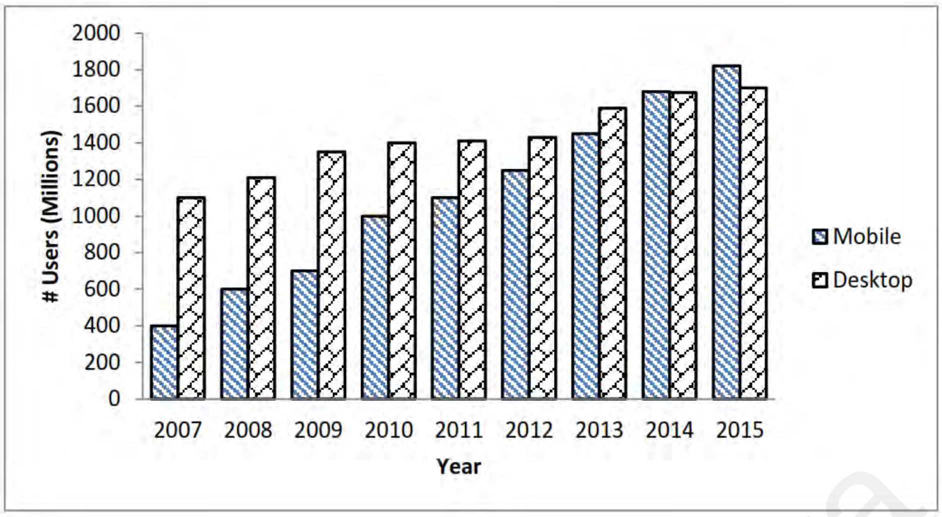

Figure 1.1: World Wide Mobile Phone Users Growth

tion energy estimation facilitates to: (a) identify resource critical rogue applications, (b) diagnose smart-phone energy consumption, (c) estimate per-application energy usage, and (d) optimize application design (R. W. Ahmad et al., 2015).

Dynamic analysis based energy estimation runs application on a smart-phone device to estimate its energy consumption. However, the performance overhead of dynamic analysis based energy estimation is very high. Dynamic analysis approach is impractical to use especially when estimation is applied to minimize energy consumption of an application. For instance, mobile cloud computing (MCC) is one of the methods to minimize the total energy consumption of an application. MCC frameworks consider energy estimation as one of the metrics to decide execution platform for the mobile application (A. R. Khan, Othman, Madani, & Khan, 2014; Abolfazli et al., 2014; Rajan, Noureddine, & Stratis, 2016). However, running mobile application on the smart-phone (dynamic analysis) to estimate its energy consumption for execution platform estimation affects the performance of MCC frameworks.

1.4 Statement of the Problem

Several traditional schemes are proposed to estimate energy consumption of smart-phone applications (L. Zhang et al., 2010a; Hao et al., 2012a; D. Li, Hao, Halfond,

University of Malaya

& Govindan, 2013; Ding, Xia, Zhang, Zhao, & Ma, 2011). These schemes consider power measurement of smart-phone components or energy cost of software operations (source code) to estimate energy consumption of a smart-phone application. The smart-phone components power measurement based estimation considers power models for smart-phone components to estimate energy consumption of an application. Alternatively, software operations energy cost based estimation considers energy cost of activities within smart-phone application to estimate its energy consumption.

Traditional smart-phone components power measurement based energy estimation schemes consider dynamic analysis approach to estimate energy consumption of an application. Smart-phone components power consumption highly depends on utilization level of smart-phone components such as Wi-Fi, CPU, LCD, and blue-tooth (Do et al., 2009; M. Dong & Zhong, 2010; Ding et al., 2011). Being opting dynamic analysis estimation approach, the traditional smart-phone components power measurement method runs the application on the smart-phone device to estimate its energy consumption. Dynamic analysis approach estimates the power state of smart-phone components during application execution on a smart-phone device for energy estimation. However, estimating energy consumption of an application by running it on a hardware platform inefficiently utilizes underlying resources of a smart-phone device. As a result, it considerably prolongs energy estimation time of energy estimation schemes. Due to inefficient resource utilization, the energy estimation overhead of the estimation schemes is very high.

Type of operations within source code significantly impacts to the total energy consumption of a smart-phone application. Software operations energy cost based estimation considers the cost of each instruction within the application for its energy estimation. Applying software operation's energy cost based estimation method on

University of Malaya

sequential flow based applications is straightforward (Tiwari, Malik, & Wolfe, 1994; Mehta, Owens, & Irwin, 1996; Vasilakis, 2015). In contrast, the execution behavior of smart-phone applications is non-deterministic. Existing energy estimation methods physically run the application to log the execution paths in offline execution mode. Based on the execution paths and software operation's energy cost, energy consumption of application is estimated. However, due to dynamic analysis of smart-phone application (Hao et al., 2013; D. Li et al., 2013), the resources of smart-phone devices are inefficiency utilized that leads to prolonged energy estimation time. The prolonged energy estimation time results in high energy overhead. Also, existing estimation methods do not consider storage location of source code instructions that inaccurate estimation accuracy.

Traditional energy estimation methods follow dynamic analysis approach for energy estimation. Traditional energy estimation methods physically run target smart-phone application on the smart-phone device for its energy consumption estimation. During application running on a smart-phone, an energy estimation method profiles execution behavior of target smart-phone application for its energy estimation. As a result, the resources of the smart-phone device are inefficiently utilized that leads to high energy estimation time and overhead. The total energy overhead comprises overhead of energy estimation tool and target smart-phone application.

University of Malaya

1.5 Statement of the Objectives

To overcome the issues of traditional dynamic analysis based energy estimation schemes, this study aim for lightweight static analysis based energy estimation framework. The proposed framework estimates energy consumption of smart-phone application based on the analysis of assembly code of smart-phone application and energy cost profile for Acorn RISC Machines Instruction Set (ARM-IS) for ARM-7 architecture. The objectives of this research are as follows.

- To review existing state-of-the-art smart-phone application energy estimation schemes to gain insights to their performance limitations.
- To investigate the overhead of dynamic analysis based energy estimation schemes to reveal inefficiencies in existing methods while estimating energy consumption.
- To propose a system to estimate energy consumption of assembly based instructions within ARM-ISA and to perform impact analysis of storage location on instruction's energy consumption.
- To design and develop a lightweight energy estimation framework that proposes weighted probability based application execution flow estimation and static cache analysis method to estimate energy consumption of smart-phone applications.
- To evaluate the proposed energy estimation framework for energy estimation time, overhead, resource consumption, and accuracy in comparison to state-of-the-art dynamic analysis based energy estimation schemes.

1.6 Proposed Methodology

Fig. 1.2 highlights the methodology followed to conduct this research. The phases of this research include literature review, analysis of the problem, design of static analysis based energy estimation framework, and evaluation of framework.

This study critically reviewed existing state-of-the-art smart-phone application energy estimation schemes to highlight their strengths and weaknesses. It classify existing energy estimation schemes based on the proposed thematic taxonomies. Existing energy estimation schemes are classified into smart-phone components power

University of Malaya

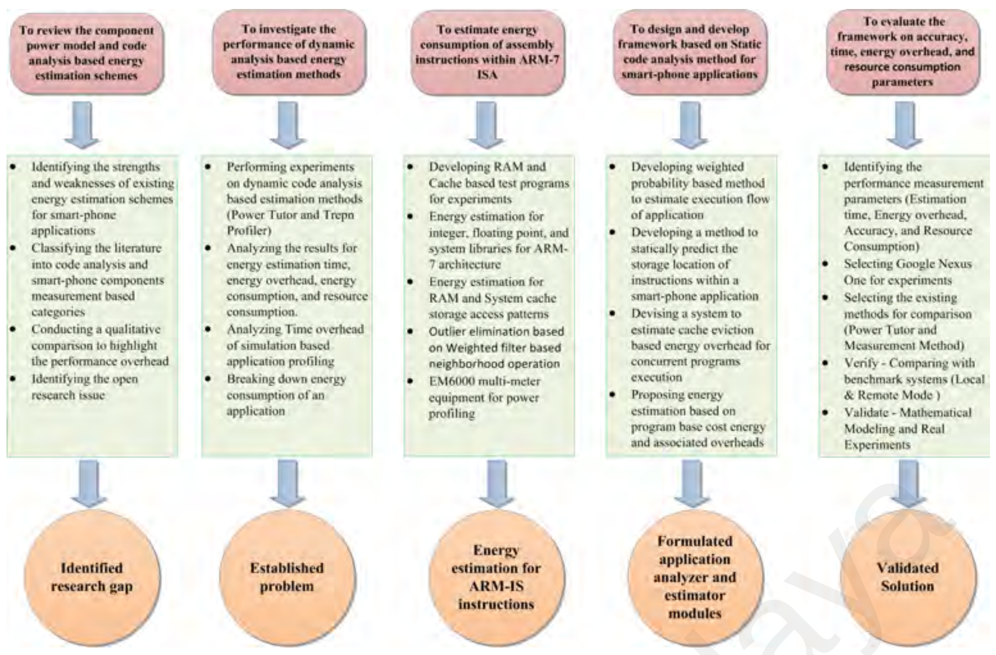
To review the component power model and code analysis based energy estimation schemes
To investigate the performance of dynamic analysis based energy estimation methods
To estimate energy consumption of assembly instructions within ARM-7 ISA
To design and develop framework based on Static code analysis method for smart-phone applications
To evaluate the framework on accuracy, time, energy overhead, and resource consumption parameters
Identifying the strengths and weaknesses of existing energy estimation schemes for smart-phone applications
Classifying the literature into code analysis and smart-phone components measurement based categories
Conducting a qualitative comparison to highlight the performance overhead
Identifying the open research issue
Performing experiments on dynamic code analysis based estimation methods (Power Tutor and Trepn Profiler)
Analyzing the results for energy estimation time, energy overhead, energy consumption, and resource consumption.
Analyzing Time overhead of simulation based application profiling
Breaking down energy consumption of an application
Developing RAM and Cache based test programs for experiments
Energy estimation for integer, floating point, and system libraries for ARM-7 architecture
Energy estimation for RAM and System cache storage access patterns
Outlier elimination based on Weighted filter based neighborhood operation
EM6000 multi-meter equipment for power profiling
Developing weighted probability based method to estimate execution flow of application
Developing a method to statically predict the storage location of instructions within a smart-phone application
Devising a system to estimate cache eviction based energy overhead for concurrent programs execution
Proposing energy estimation based on program base cost energy and associated overheads
Identifying the performance measurement parameters (Estimation time, Energy overhead, Accuracy, and Resource Consumption)
Selecting Google Nexus One for experiments
Selecting the existing methods for comparison (Power Tutor and Measurement Method)
Verify - Comparing with benchmark systems (Local & Remote Mode)
Validate - Mathematical Modeling and Real Experiments
Identified research gap
Established problem
Energy estimation for ARM-IS instructions
Formulated application analyzer and estimator modules
Validated Solution

Figure 1.2: Research Methodology

measurement based estimation and code analysis based energy estimation. Based on the thematic taxonomies, existing energy estimation schemes are compared to highlight the commonalities and variances among them. The issues in this domain of research that affect the performance of existing energy estimation schemes are highlighted.

In the second phase, the research problem is investigated by analyzing the performance of dynamic analysis based energy estimation schemes. A set of benchmark applications are used to investigate energy estimation overhead, time, resource consumption, estimation time, and accuracy of existing methods. The impact of increasing data size of an application on the performance of dynamic analysis based energy estimation is also highlighted. The energy consumption rate by different constructs of an application is also discussed. The overhead of estimating low architecture level system details such as an estimation of cache miss/hit rate is also highlighted. The proposed study has considered Power Tutor and Trepn Profiler dynamic analysis based energy estimation methods for analyzing the problem. Also, results of the measurement-based method (consisting of multi-meter, external mo-

University of Malaya

bile battery, and resistors) are used as ground truth values for highlighting the accuracy of existing energy estimation schemes.

In the third phase of this research, energy consumption cost for different set of instructions within ARM-ISA for ARM-7 is estimated. This study estimated energy consumption for arithmetic, logical, data movement, branch, conditional, LOAD/STORE, and system library routines. A set of test programs to estimate energy consumption for assembly based ARM-IS instructions are proposed. Energy cost for both integer and floating point operations is estimated. Furthermore, this study has considered both system cache and RAM to highlight the impact of storage location on the energy consumption of an instruction. During test program execution on smart-phone, background processes inaccurate the results. This study has opted weighted filter based neighborhood operation to analyze and remove the outliers to improve the accuracy. EM6000 multimeter equipment is chosen to estimate power consumption during test program execution on smart-phone. It has considered test program execution time, average power consumption, and size of test program to estimate energy consumption of a single ARM assembly instruction.

In the fourth phase of this research, a lightweight framework for energy estimation of smart-phone applications is proposed. The proposed framework called static analysis based lightweight energy estimation framework (SA-LEEF) analysis assembly source code of an application and uses ARM-IS energy cost profile (objective 3) to estimate energy consumption of the smart-phone application. It also considers the associated overheads during energy estimation process. The execution behavior of smart-phone applications is non-deterministic by nature. In traditional energy estimation schemes, the execution flows and storage locations for instructions within a smart-phone application are determined based on dynamic analysis of application. SA-LEEF has proposed a weighted probability based method to analyze the data

University of Malaya

sets for branch statements to estimate the execution flows of an application. To handle instructions storage location issue, it has proposed a static cache distance based method to analyze ARM and thumb instructions to estimate the storage access location of instructions. The proposed static cache distance method considers cache size, size of cache lines, and the distance between chunk of application code to predict cache hit or miss. For repetitive statements, it has opted loop slicing method to estimate the loop's bounds within the target smart-phone application. It has proposed a method that considers base power cost while two tenants are running and the cache eviction time to estimate the concurrent program execution energy overhead. To increase the usability feature, SA-LEEF offers ARM-IS energy cost profile as a service for a particular ARM architecture. The proposed energy estimation model for SA-LEEF considers base cost energy (source code energy cost), user-system interaction cost, concurrent programs execution overhead due to cache eviction, and ARM-IS energy profile access cost for energy estimation. Due to static analysis methodology, the proposed framework significantly minimizes estimation time and energy estimation overhead of existing dynamic analysis methods.

In the last phase, the significance of proposed framework is evaluated based on real experimentation using a set of benchmark applications. For different benchmark applications, SA-LEEF is evaluated for the android smart-phone device (Google Nexus One). Also, the behavior of different modules of the proposed framework is verified based on a set of benchmark applications. The performance behavior of proposed static analysis based energy estimation framework is analyzed from the perspective of its energy consumption overhead, estimation time, resource consumption, and accuracy. Based on a set of standard benchmark applications with varying workloads and code sizes, the empirical data is collected. Findings of proposed framework to the existing dynamic analysis based energy estimation methods

University of Malaya

are compared to validate the lightweight nature of proposed framework. Lastly, SA-LEEF is qualitatively compared to existing code analysis based energy estimation frameworks to highlight its differentiating features. It has also discussed the infeasibility for comparing SA-LEEF to code analysis based energy estimation schemes.

1.7 Scope

This study estimates energy consumption of smart-phone applications based on code analysis estimation method. It statically analysis smart-phone application to identify execution flows and storage locations for energy estimation. The scope of this research work is limited to the energy estimation of Sd-card I/O based smart-phone applications. The Sd-card I/O based applications such as audio song player does not require user interaction during its execution. Therefore, the energy estimation of interactive smart-phone applications such as mobile gaming is not considered in this study. The proposed framework considers obj dump of the target smart-phone application for its energy estimation. Therefore, the energy estimation of the applications whose obj dump cannot be generated are out of this research scope. The native smart-phone application code is written in C or C++ languages. The obj dump of native smart-phone applications is usually generated using objdump Linux utility. Therefore, this study is valid for the energy estimation of native smart-phone applications. Also, the proposed research profile energy cost for ARM-7 architecture. Therefore, the energy estimation of smart-phone applications for other ARM architectures is not considered in this research.

1.8 Layout of Thesis

This thesis is structured as a set of chapters as depicted in Fig. 1.3. It is divided into seven chapters including introduction, literature review, problem analysis, proposed lightweight energy estimation framework, evaluation, results and discussion, and

University of Malaya

conclusions. The detailed description and ingredients of each chapter are as follows.

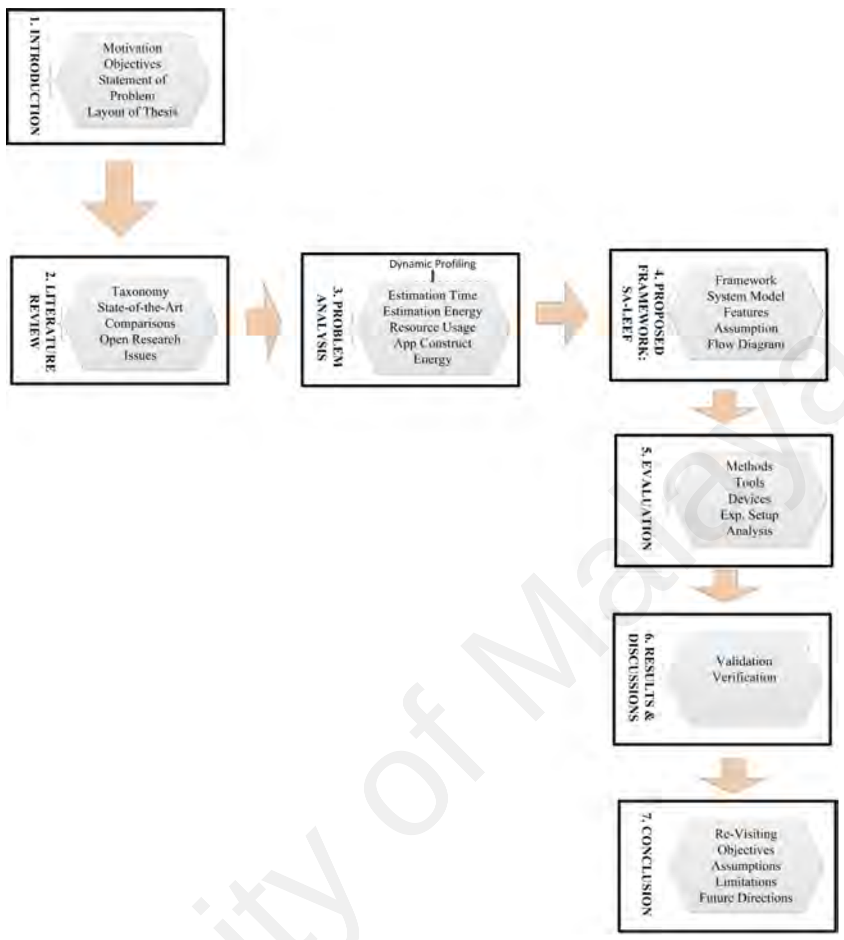

Figure 1.3: Thesis Organization

Chapter 2 presents a through discussion on state-of-the-art smart-phone application energy estimation schemes. It proposes thematic taxonomies to classify existing literature into code analysis and smart-phone components power measurement based energy estimation schemes. It qualitatively analysis existing energy estimation schemes to highlight the common and dissimilar features in them. It presents several open research issues in this domain of research and also discusses the issues that are considered for this research work.

Chapter 3 investigates the issues of dynamic analysis based energy estimation schemes in term of their high energy estimation overhead, prolonged estimation time, high resource consumption, and limited accuracy. It has considered Trepn

University of Malaya

Profiler and Power Tutor energy estimation tools for the analysis. Both of these tools are highly cited in the research industry. Results of the Measurement-based method is used as ground truth value to report the accuracy of dynamic analysis based energy estimation methods. It also investigates high profiling overhead while estimating low architecture level details of smart-phone devices.

Chapter 4 proposes a 2-tier static analysis based framework for energy estimation of native smart-phone applications. The proposed framework consists of four key modules including ARM instruction Energy Profiler, Application Analyzer, Application Energy Estimator, and Collaborator. It discusses flow diagram and system model of the proposed framework. It highlights distinguishing features of proposed framework. Moreover, it discusses assumptions and constraints considered while designing this framework.

Chapter 5 discusses data collection methods for the evaluation of proposed framework. It discusses tools, data collection techniques, devices, and underlying hardware modules to evaluate energy estimation framework. For the instruction energy profiling, the proposed framework has considered external physical measurement method. A set of benchmark applications are considered to evaluate modules of proposed framework.

Chapter 6 validates proposed framework by comparing results of system model for the proposed framework to the results of its empirical evaluation. It compares finding of proposed framework to Power Tutor and Measurement-based energy estimation methods for two scenarios including (a) Standard benchmark code size case and (b) Application data size case. Moreover, it has compared the performance of SA-LEEF to Power Tutor for two of its operational modes including Local mode and Remote mode.

Chapter 7 concludes this research work by re-visiting the research objectives.

University of Malaya

It highlights main contributions of this research. It discusses findings, highlights significance of proposed framework, and states limitation in proposed work to put forward future research directions.

University of Malaya

CHAPTER 2: ENERGY ESTIMATION SCHEMES FOR SMART-PHONE APPLICATIONS

This chapter briefly discusses the essence of smart-phone applications, reviews the state-of-the-art smart-phone application energy estimation schemes, and presents thematic taxonomies for classification of literature. It presents several open research issues that affect the performance of existing energy estimation schemes.

This chapter is organized into seven main sections. Section 2.1 discusses some basic terminologies and necessary background to define fundamental concepts of smart-phone application energy estimation. Section 2.3 presents a thematic taxonomy for smart-phone components power measurement based energy estimation schemes. It reviews and compares existing state-of-the-art energy estimation schemes to highlight the commonalities and variances among them. Section 2.4 discusses code analysis based smart-phone application energy estimation followed by a detailed thematic taxonomy, extensive literature review, and in-depth comparisons of existing energy estimation schemes. Section 2.5 discusses main features of smart-phone application performance analysis tools. Section 2.6 discusses open research issues and challenges that hinder designing optimized energy estimation methods for smart-phone applications. Section 2.7 presents a brief summary on energy estimation to analyze the most important issues in energy estimation research domain. Lastly, Section 2.8 concludes the whole chapter and presents an overview of findings.

University of Malaya

2.1 Background

Nowadays, smart-phones have become an integral part of one's life due to high dependency on them. This section defines a smart-phone device and highlights main features of smart-phone applications. It overviews smart-phone application energy estimation methods. The majority of the contents for this chapter are taken from our published articles in ISI index journals as highlighted in the footnote of this page[1] [2].

2.1.1 Smart-phone Device

A smart-phone is a cellular device that performs many of the functions of a desktop computer and offers internet access (Wi-Fi,3G), touch screen interface, local and remote data storage, and operating system capable of running downloaded applications. It integrates capabilities of a cell phone to more common features of a hand-held computer or PDA to enrich the services for the user. Due to size limitation, smart-phone carries low-speed CPU and limited capacity of RAM storage. Moreover, smart-phones are equipped with lithium battery with less than the 3000mAh capacity that lasts for a few hours when the network and computations intensive applications run on it (Abolfazli et al., 2014; R. W. Ahmad et al., 2015).

The hype in popularity of smart-phones has remarkably increased due to their unlimited applications in various computing domains such as education (Thornton & Houser, 2005), management information system (Wu & Wang, 2005), and health-care monitoring (Pascu, White, Beloff, Patoli, & Barker, 2016; Joos et al., 2016). Smart-phone applications have inherited main features of rich internet applications to enrich user experience. Recent smart-phone applications offer code portability, asynchronous communication, quick responsiveness, and extensive functionality. It embodies context aware and multi-tier services to entertain smart-phone users (PCs, 2008; Abolfazli et al., 2014). However, in-lining with increasing functionality, these

[1]Ahmad, Raja Wasim, Abdullah Gani, Siti Hafizah Ab Hamid, Feng Xia, and Muhammad Shiraz. "A Review on mobile application energy profiling: Taxonomy, state-of-the-art, and open research issues." Journal of Network and Computer Applications 58 (2015): 42-59.

[2]Ahmad, Raja Wasim, Abdullah Gani, Siti Hafizah Ab Hamid, Mohammad Shojafar, Abdelmuttlib Ibrahim Abdalla Ahmed, Sajjad A. Madani, Kashif Saleem, and Joel JPC Rodrigues. "A survey on energy estimation and power modeling schemes for smartphone applications." International Journal of Communication Systems (2016).

University of Malaya

applications heavily uses light sensors, GPS, compass, and accelerometers, to perform the desired task. As a result, sensors deplete battery charge when application triggers them to perform required task (Gavalas & Economou, 2011; Abolfazli et al., 2014; Plaza, MartíN, Martin, & Medrano, 2011; X. Li & Gallagher, 2016).

2.1.2 Smart-phone Application Energy Estimation

Energy of an application is estimated based on its average power consumption and execution time. Alternatively, the power consumption states the rate at which energy is consumed (Tiwari et al., 1994). Smart-phone application energy estimation considers power models for smart-phone components or software operations to estimate energy consumption of a smart-phone application. Smart-phone components power model based energy estimation uses state of charge (SOC) estimation methods such as coulomb counting and terminal voltage methods to forecast energy consumption of an application as highlighted in Fig. 2.1. Alternatively, software operation's cost based estimation method (also known as code analysis based estimation) considers base cost energy of source code instructions to estimate energy consumption of the smart-phone application. Section 2.1.2.1 and section 2.1.2.2 briefly discusses energy estimation methodologies for smart-phone applications.

University of Malaya

2.1.2.1 State of Charge Energy Estimation

State of charge represents total remaining charge capacity inside a smart-phone battery. It reflects the performance of a battery. Accurate SOC estimation improves battery life, prevents over-discharging, and assists application developers to adopt rational control strategies to save energy. SOC for smart-phone battery is estimated based on coulomb counting (Shin et al., 2013; Sepasi, Ghorbani, & Liaw, 2014) and terminal voltage method (C. Jiang, Taylor, Duan, & Bai, 2013; H. He, Zhang, Xiong, Xu, & Guo, 2012; Gholizadeh & Salmasi, 2014). Coulomb count-

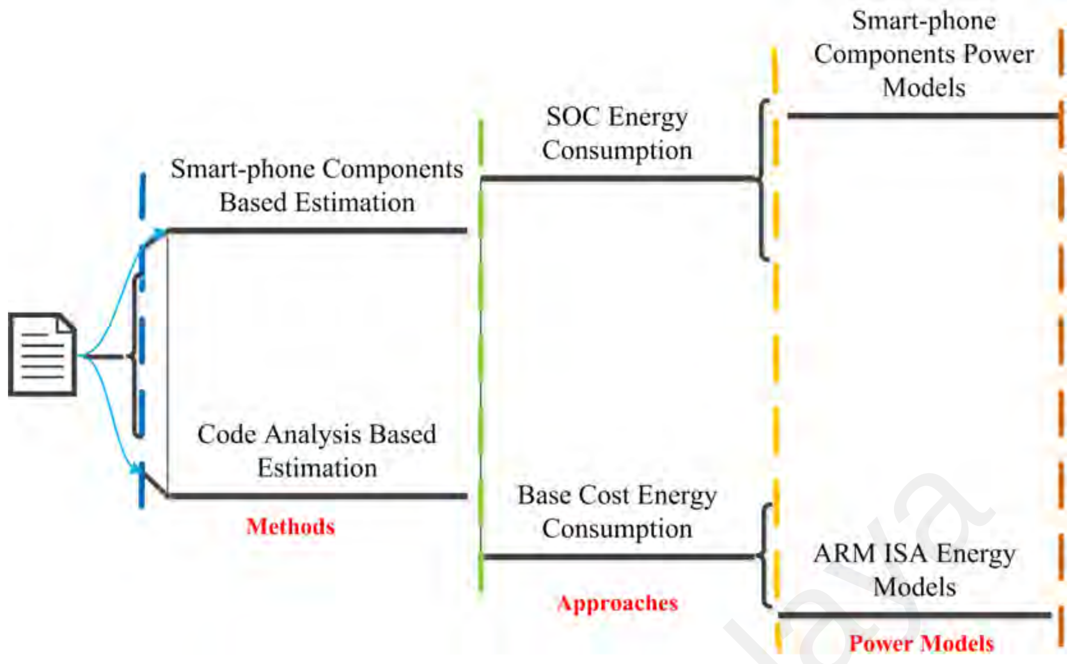
Smart-phone Components Power Models
SOC Energy Consumption
Smart-phone Components Based Estimation
Code Analysis Based Estimation
Methods
Base Cost Energy Consumption
Approaches
ARM ISA Energy Models
Power Models

Figure 2.1: Overview of Smart-phone Application Energy Estimation Methods

ing estimates SOC based on an accumulative current drop by directly accessing the current sensor within smart-phone devices. The accuracy of coulomb counting method is highly affected by numerous external and internal factors such as battery aging, temperature, and charging/discharging rate (L. Lu, Han, Li, Hua, & Ouyang, 2013; Zhong, Zhang, He, & Chen, 2014; Hoque, Siekkinen, Khan, Xiao, & Tarkoma, 2015). Alternatively, terminal voltage method estimates SOC based on voltage drop because of the internal impedance of battery during its discharging. Both coulomb counting and terminal voltage methods are implemented in battery fuel gauge hardware module of the smart-phone battery. For energy estimation of an application, operating system (OS) considers android power APIs to access the fuel gauge (Y. He, Liu, Zhang, & Chen, 2013; Hu, Youn, & Chung, 2012; W.-Y. Chang, 2013).

2.1.2.2 Base Cost Energy Estimation

Smart-phone application performs a series of activities during its execution on a smart-phone device. Base cost energy of an activity truly depends on the circuitry

University of Malaya

it triggers during application execution on the smart-phone device. Within an application, the energy cost of each operation such as variable declaration, assignment, and arithmetic operations is fixed. In comparison to coarse granular operations, the energy cost of a fine granular operation is much lower. The energy cost of each instruction within Acorn RISC Machines Instruction Set (ARM-IS) is dissimilar and highly depends on cycles per instruction (CPI) for a particular instruction. For a single instruction, energy consumption also depend on the type of op-code, number of operands in instruction address part, operands type, and architecture of smart-phone device. For instance, base cost energy for ARM assembly based LOAD instruction is higher than ADD instruction because memory-based operations are expensive than simple Arithmetic and Logical Unit (ALU) based operations. Base cost energy of a program is estimated based on the energy cost of the source code of the application (Jayaseelan, Mitra, & Li, 2006; Allcott & Greenstone, 2012; Hao et al., 2013; Sinha, Ickes, & Chandrakasan, 2003; Tiwari et al., 1994).

2.2 Static Analysis vs. Dynamic Analysis

Static analysis is a program analysis method that debugs an application based on source code examination without running it on underlying hardware platform. Application developers consider static analysis of their applications to uncover the bugs. Static analysis approach is used for debugging application at its earlier development stages. It helps to understand the structure of the code for better software quality control (Emanuelsson & Nilsson, 2008; Pistoia, Chandra, Fink, & Yahav, 2007). The programers use static analysis to identify the variables with undefined values, unreachable code, syntax violation, and security vulnerabilities. As static analysis is concerned to the structure of the code, therefore it is mainly used to identify the logical errors within a piece of code (Ayewah, Hovemeyer, Morgenthaler, Penix, &

University of Malaya

Pugh, 2008). To identify the uninitialized variables in a program, static analysis considers flow graph and examines how variables are used. In a graph, the nodes represent the program main units whereas arcs represents the dependency among the modules of the program. Moreover, static analysis helps to estimate worst case execution time of a program. Loops are the most energy consuming element within an application. Static analysis opts loop slicing and abstract interpretation (AI) techniques to estimate upper bounds of a loop. The loop slicing method extracts the set of variables and the statements within a program that must be considered for loop bound estimation. AI is a program analysis method that estimates the possible set of values for variables at different points of the program (Maroneze, Blazy, Pichardie, & Puaut, 2014; Lisper, 2014; Sun & Cassé, 2016). Also, static analysis based cache prediction helps to estimate the storage location of code within an application for high level program constructs based on graph theory (Guan, Yang, Lv, & Yi, 2013; Grund, 2012).

Dynamic analysis evaluates a program by executing it on a physical hardware machine in real-time mode. The main purpose of dynamic analysis is to examine the code for error finding. In comparisons to static analysis, dynamic analysis is performed after program development. To be effective, dynamic analysis requires the target application to be executed with multiple set of use cases (Gosain & Sharma, 2014; Cornelissen, Zaidman, Van Deursen, Moonen, & Koschke, 2009). There are several tools available for dynamic analysis of a program. For instance, tools including ClearSQL, Daikon, Dmalloc, and Cenzic follows dynamic analysis approach for program understanding (Nimmer & Ernst, 2001; Mandeep, 2008; Watson, 2004). CleanSQL is a code illustration tool for PL/SQL. Alternatively, Cenzic and Daikon supports security vulnerability identification and invariant detection within a program, respectively. Alternatively, Dmalloc helps application developers to check the

University of Malaya

memory allocation and leaks for their applications. Valgrand (Seward, Nethercote, & Fitzhardinge, 2004) is another dynamic analysis based tool that analysis applications to estimate execution flows, memory leaks, instructions count, and cache hit/miss rate. The disadvantage of dynamic analysis is the requirement to run application for its testing. However, as compared to static analysis, coverage issue problem is well handled by dynamic analysis tools. Moreover, another advantage of dynamic analysis is its support to identify vulnerabilities in code that is false negative in case of static analysis. Also, dynamic analysis can be done over and over for extensive testing. However, dynamic analysis requires sufficient hardware resources that make it costly compared to static analysis.

University of Malaya

2.3 Smart-phone Components Power Measurement Based Energy Estimation

Smart-phones of the modern day come with wide variety of components embedded in them. Typical smart-phone components include CPU, Wi-Fi, memory, 3G, GPS, compass, LCD, touch screen, and speaker. Smart-phone applications simultaneously utilize several components to offer a rich experience to its users. During application execution on a smart-phone device, each component consumes unlike power depending on its execution states. Total energy consumption of an application is estimated based on the power consumption of smart-phone components during its execution on the smart-phone device. The power consumption of a smart-phone component is estimated based on its power model constructed using physical measurements or self-metering methods as discussed below.

Fig. 2.2(a) highlights a physical measurement based power measurement setup. In Fig. 2.2, measuring hardware devices such as multi-meter, power-meter, or ammeter captures voltage and current drop across battery terminals to estimate power consumption of the target application. Alternatively, self-metering method (Fig. 2.2(b))

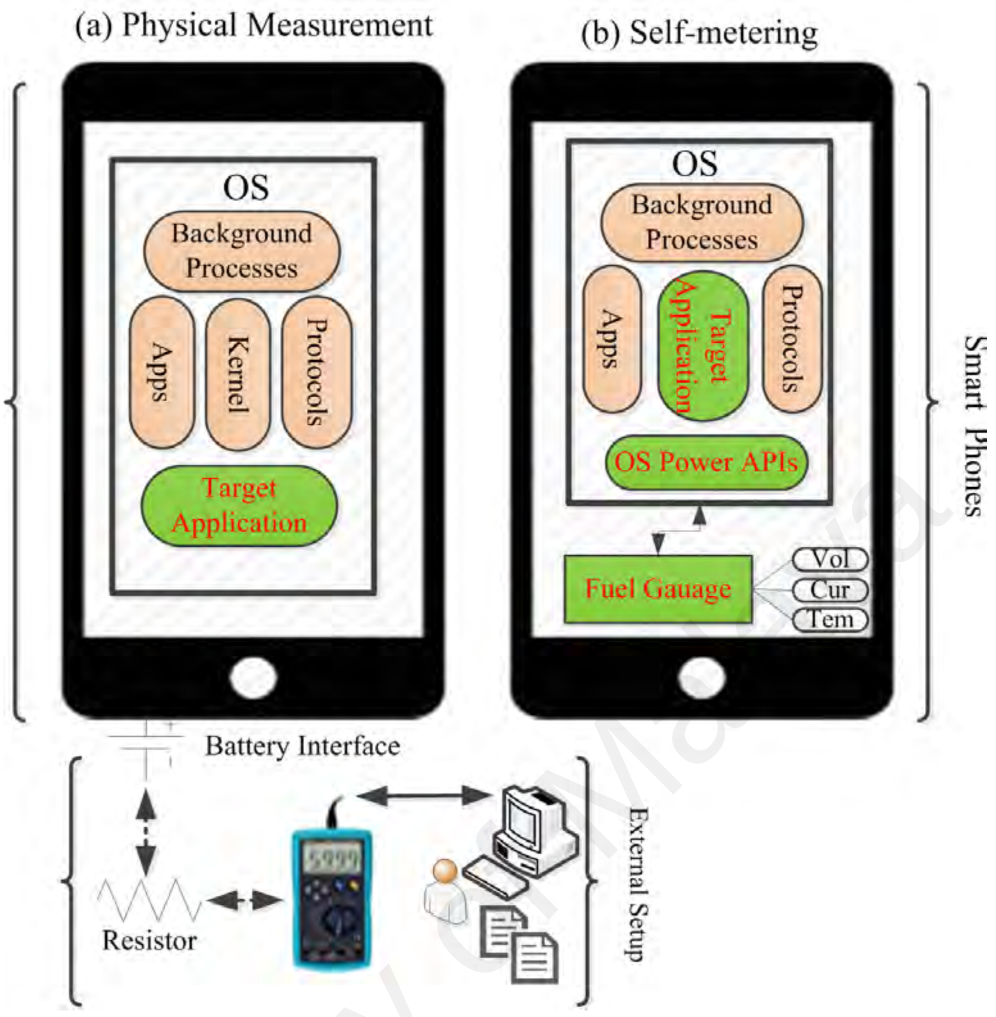

Figure 2.2: Overview of Physical Measurement and Self-metering based Energy Estimation

uses smart battery interface of the smart-phone device for power measurement of the target application.

Section 2.3.1 presents a taxonomy for classification of existing smart-phone components power measurement based energy estimation schemes.

2.3.1 Taxonomy of Smart-phone Components Power Measurement Based Energy Estimation Schemes

Fig. 2.3 presents a thematic taxonomy to classify existing state-of-the-art smart-phone components power measurement based energy estimation schemes. The parameters to classify state-of-the-art energy estimation schemes include power model type, methodology, granularity, training mode, power modeling approach, and ob-

University of Malaya

Table 2.1: Comparison of Self-metering and Measurement based Estimation

Parameter	Physical Measurement	Self-metering
Accuracy	High	Low
Power Source	Uses an external power source for hardware equipment during energy profiling process	Uses smart-phone's battery power for application energy profiler
Setup	It interfaces smart-phone battery to the power meter	APIs to access the battery draining rate, temperature, and total remaining battery power
Support	System level power consumption monitoring	Individual hardware component and system level
Dependency	Accuracy depends on hardware sampling rate	Accuracy depends on Fuel guage sensor's accuracy
Sampling Rate	Sampling rate as offered by hardware equipment	Power sampling rate is as offered by underlying OS
Structure	It uses physical tools like power meters, multi-meter, amm-meter	Android APIs, pre-constructed component power models
Overhead	Do not suffer from feedback loop	Suffers from feedback loop as profiling application too runs on the same smart-phone device
Scalability	Not scalable	Scalable

jective function.

Attributes of *Power Model Type* parameter states whether an estimation model has followed external power measurement or self-metering based estimation methods to construct power models for smart-phone components. External power measurement method employs power estimation tools (e.g., multimeter and power meter) to obtain voltage and current drop across sense resistor attached to the power rail of the smart-phone battery. Alternatively, the self-metering method employs SOC estimation methods to estimate energy consumption of an application or smart-phone device.

Attributes of *Methodology* parameter defines the power modeling approach based on the kind of input variables the model uses. Utilization-based methodology co-relates smart-phone components power usage to its resource consumption rate. Utilization-based methods exploit hardware performance counters (HPCs) to model power consumption of smart-phone sub-components such as CPU fetch cycles, cache hit rate, pipeline stalls, and number of cache hits/miss. The utilization-

University of Malaya

based methodology is effective for capturing the linear relationship between resource and energy consumption of the hardware component being modeled. Event driven methodology considers the non-linear energy consumption behavior (e.g., Wi-Fi tail energy) of smart-phone components for accurate energy estimation.

Energy estimation *Granularity* states the level at which an estimation model finds energy consumption of the application. Attributes of granularity for energy estimation includes smart-phone application, web browser, smart-phone device, function, process, and device components. For instance, Power Tutor estimates energy consumption at the application and smart-phone component granularity levels.

For power modeling of smart-phone components in lab setting environment, training measurements are collected to build power models for smart-phone components. For training measurements, each smart-phone component is set to execute at allof its possible execution states where rest of the components are put to their lowest execution modes. Attributes of *Training Mode Locality* parameter describes the platform where the analysis on the smart-phone component's power training profile is performed to construct power models for smart-phone components. The smart-phone component's power training analysis is carried either at the mobile device or on desktop/cloud servers. Trepn Profiler considers on-device training mode locality for power modeling and application energy estimation.

Power Modeling Approach highlights method opted to analyze energy consumption behavior of smart-phone components. The attributes of this parameter include deterministic or statistical modeling designs. Deterministic power modeling paradigm opts power state machine (PSM) to estimate energy consumption of an operation. PSM based estimation considers, (a) power consumption at each system state, (b) number of transitions among states, and (c) power consumption during each transition to estimate energy consumption of a mobile component. Alterna-

University of Malaya

tively, statistical power model design considers statistical models such as regression analysis to investigate non-linear power consumption behavior of smart-phone applications.

Objective Function parameter states main aim of energy estimation schemes. Energy estimation methods aim to automate energy estimation process, minimize estimation overhead, and to propose power models for smart-phone devices.

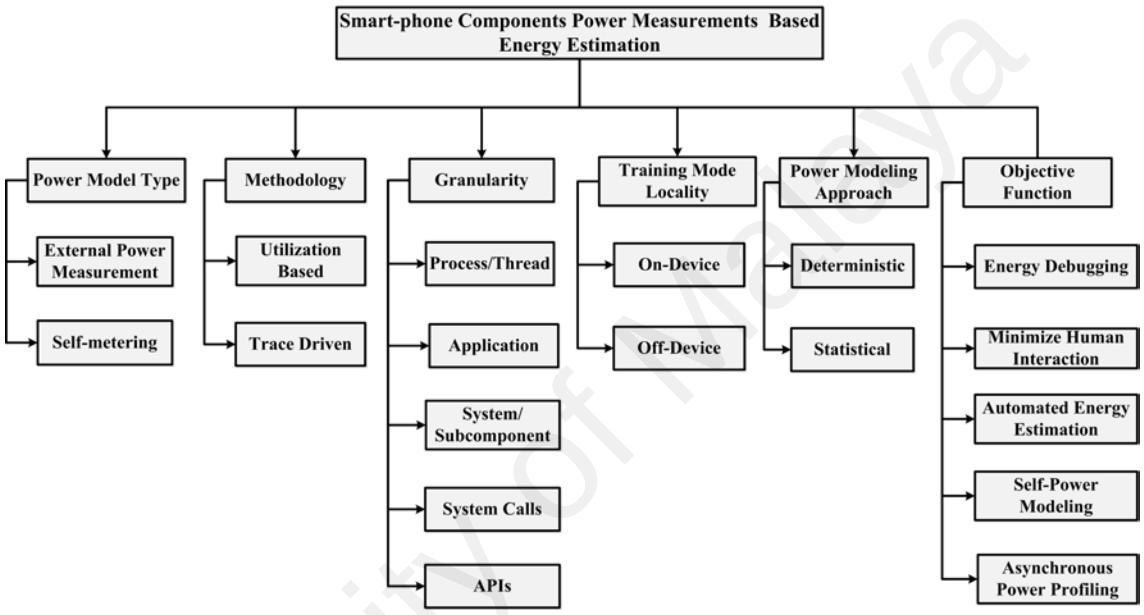

Figure 2.3: Taxonomy of Smart-phone Components Power Measurement Based Energy Estimation Schemes

2.3.2 Review of Smart-phone Components Power Measurements Based Energy Estimation Schemes

This section presents a review on state-of-the-art smart-phone components power measurements based energy estimation schemes. Section 2.3.2.1 discusses physical measurement based energy estimation schemes; whereas, Section 2.3.2.2 reviews self-metering based energy estimation schemes.

2.3.2.1 Physical Measurement Based Energy Estimation

The knowledge of where and how energy is consumed within a smart-phone assists designing resource efficient smart-phone applications. DUT (Carroll & Heiser, 2010)

University of Malaya

is a physical measurement based power modeling scheme that break-down energy consumption of a smart-phone into its sub-components. It has considered lab setting environment to construct power models for smart-phone components. The design of DUT includes mobile under test, data acquisition (DAQ) board, and a sense resistor to construct models for smart-phone components. To investigate power consumption behavior, a sense resistor (residing at power rail of the target mobile component) captures voltage and current drop using DAQ module. DAQ is a high rate power profiling tool that captures power profile of an application executing for a short period of time (milli-seconds). DUT analyzes power profile in an offline execution mode to propose power models for smart-phone components. For instance, Eq. 2.1 presents power model for energy estimation of audio and video services. Alternatively, Eq. 2.2 models energy consumption for call and emailing services. In the mentioned equations t represents total activity time whereas PBL defines the back-light power of the smart-phone device. DUT has proposed models for free-runner smart-phone that can be replicated easily as its design files are freely available. However, the proposed scheme has not modeled 3G radio as free-runner smart-phone does not have the support for it.

University of Malaya

$$E_{Audio} = 0.32 \times t, E_{Video} = (0.45W + PBL) \times t \tag{2.1}$$

$$E_{Call} = 1.05W \times t, E_{Email} = (0.61W + PBL) \times t \tag{2.2}$$

The energy consumption of a smart-phone device increases when a user scroll the screen to perform some task. During smart-phone components power modeling, interaction with smart-phone to adjust settings in-accurate the estimation readings owing to sudden rise in LCD power consumption rate. To suppress the estimation error due to interaction with smart-phone during power profiling, A. C. Rice et al. (A. C. Rice & Hay, 2010) has proposed a framework that downloads (online mode) scripts to generate annotated traces for smart-phone applications energy consumption assessment. Proposed framework consists of a desktop server (for timestamped power recording) and power measurement tool (power capturing). Smart-phone hosted module runs script on the smart-phone device and uses power measurement tool to profile and transfer traces to the remote server. To assure the strict synchronization between the smart-phone device and remote cloud server, the framework uses a test client application that formulates synchronization pulse. During the offline phase it processes power profile using regression analysis to construct power models. Authors considered a network study to examine the power consumption of 2G/3G radios. It was concluded that 2G is more power consuming than 3G. The advantage of proposed framework is its versatility to support non-invasive tasks, batch job processing, automated test generation, and execution. However, recent smart-phones are equipped with OLED screens which increase battery charge consumption rate when the color of GUI is brighter. The proposed model can be re-configured to work for OLED based smart-phone screens too.

A similar to A. C. Rice et al. (A. C. Rice & Hay, 2010) work, but with more detailed analysis has been reported in (A. Rice & Hay, 2010) to analyze the behavior of smart-phone power consumption using hardware instrumentation. The proposed framework consists of hardware equipment and software module. The former measures voltage drop across a sense resistor to estimate power consumption whereas the latter acquires test scripts from a remote server to generate the execution log during application execution on smart-phone device. In this study it was observed that Wi-Fi network power consumption is a function of network traffic, connec-

University of Malaya

tion establishment method, data transmission rate/size, network protocol type, and sender buffer size. The power consumption of an application during network activity is costly when considering the DHCP protocol (dynamic host configuration protocol) compared with static address assignment policy. However, access point position, attenuation pattern, and types of radio significantly affect battery power consumption, which is not considered in this scheme. The profiling module also does not consider the asynchronous behavior of network interfaces, such as Wi-Fi. Furthermore, it requires a stable network connection to download the test script to analyze power consumption behavior. The effects of the distance between the smart-phone and the Wi-Fi access point on the total power consumption of the Wi-Fi component can improve the estimation accuracy of proposed model.

The asynchronous nature of I/O operations during application execution complicates the process of identifying what is responsible for an I/O activity because of rapid context switching during application execution. Power Memo (Tsao et al., 2012) is a measurement-based energy profiler that overcomes the aforementioned issue by adding the process identifier (PID) to the socket data structure to precisely identify the process that has generated or received the packets. Power Memo estimates application power consumption at process and function granularity levels. The Power Memo architecture deploys the required system modules at host and target sides. The host side implements the GUI module which, (a) acts as a control center and access the data-acquisition card (DAQ), (b) emulates mobility using a single attenuator module, and (c) maps power measurements to calculate the total energy for each system activity. On the target side, the kernel module uses kernel probes (K-probes) and user space probes (U-probes) to support static and dynamic profiling. Compared with static profiling, dynamic profiling does not require the source code of the application to generate a report. The kernel module logs system

University of Malaya

activities and other necessary parameters in a file before transferring the log to the user space for energy consumption estimation using *National Instruments PCI-6115* data-acquisition board. However, the accuracy of I/O energy estimation is affected by noise pattern on channels, type of radio, distance to the access point (in case of Wi-Fi), and the available bit rate that has not been focused by the Power Memo.

Thiagarajan et al. (Thiagarajan, Aggarwal, Nicoara, Boneh, & Singh, 2012) proposed a framework to investigate the power consumption behavior of smart-phone based web browser. The design of the proposed energy estimation scheme consists of a server, a smart-phone, and a multi-meter module. The server entity control and manages the smart-phone and multimeter equipment during the power profiling process. In the first step, the server module communicates to web browser profiler on the smart-phone to repeatedly load a specific URL. During the subsequent stages, the profiler measures energy consumption of web browser using a multi-meter hooked with smart-phone battery while the browser renders the web pages. The software part of the proposed scheme includes web browser profiler and android browser. During browsing activity on smart-phone, the profiler caches the web page elements and monitors 3G/Wi-Fi signal strength, data transfer rate, and page loading time to estimate the power consumption. The web browser loads web pages either in “with cache” or “no cache” mode. The authors concluded that the cascade style sheet and java scripts are expensive elements while accounting browser energy. The critical aspect of the proposed scheme is this that the sampling rate of the chosen multi-meter was low that lead to estimation inaccuracy. Also, the proposed study has considered energy estimation of only a few web pages; however, the proposed model can be extended by considering complete web session energy estimation.

F. Jiang et al. (F. Jiang, Zarepour, Hassan, Seneviratne, & Mohapatra, 2015) conducted experiments to investigate and compare the power consumption of three

University of Malaya

input modalities including soft key, speech to text, and swype. Authors performed primary and secondary experiments to investigate these three modules with and without user related context information. To capture voltage and current drop, the shunt resistor is attached to the battery terminal to capture readings during activity on the mobile phone. It was concluded that for short size messages (14-30 characters) soft tex-ting is most energy efficient. However, for long messages, soft key outperforms than other two modalities in terms of power consumption. Also, power consumption for different applications such as email via a web browser, video streaming, online games, news/weather, and emailing, has been investigated with non-textual online/offline activities. This study helped to find out how battery life of the smart-phone can be increased. However, apart from the input modalities, numerous other factors such as bugs in the code, Wi-Fi locks, and aging factor of smart-phone batteries also affects battery's power consumption rate which is overlooked in this study while classifying three input modalities.

2.3.2.2 Self-metering Based Energy Estimation

Unsupervised power profiling assists to reduce human efforts during smart-phone components power modeling. PowerProf (Kjærgaard & Blunck, 2011) follows on-demand power modeling paradigm that re-constructs smart-phone power models when the hardware or software is updated. It considers Genetic algorithm (GA) to propose power models for smart-phone components to represent their dynamic power consumption behavior. PowerProf considered smart battery interface (fuel gauge sensors) for capturing current and voltage drop through android built-in power APIs. Power model for each smart-phone component has considered four power states to represent power consumption behavior with limited search space to speed up power estimation process. Moreover, to optimize mobile battery consumption, PowerProf

University of Malaya

offers offline training mode to perform resource expensive execution there. However, during the training phase, it overlooks inter smart-phone components dependency that affects estimation accuracy. Also, as PowerProf has considered four power states for a smart-phone component; therefore, estimation accuracy is limited.

Power Tutor follows off-device power state finding for online power model generation process. On-device power model generation do not require an external hardware device to construct power models for smart-phone components. The proposed framework discussed in (L. Zhang et al., 2010a), called Power Booter, consists of two components including training and model construction. Training phase profiles power consumption of each mobile component using built-in smart battery interfaces; whereas, model construction module applies linear regression to find a relationship between resource usage and power consumption to construct power models. The overhead of power booter is very high as training phase is both resource and time consuming process. Also, Power Booter has exploited voltage discharge curve for voltage/current estimation. However, voltage discharge curve based estimation is not scalable as it varies with time and age of the battery. The advantage of this method is its adaptability as it does not depend on external hardware devices for power modeling.

S. Dong et al. presented Se-same (M. Dong & Zhong, 2010), an energy estimation framework, to propose a self-modeling approach for constructing high-rate and accurate smart-phone components power model. Se-same consists of collector, model modeling, and model constructor modules, to estimate application energy consumption. Among all modules, collector module being located within the kernel collects power and utilization logs based on advanced configuration profiling interface (ACPI) interface. Alternatively, model modeling module generates high-rate smart-phone components power model based on prediction transformation compo-

University of Malaya

nent (PTC). The estimation overhead of Se-same is very limited as it exploited external physical server to execute resource expensive tasks. However, the acceptability of Se-same is very low as it considers only those components which are visible to OS for energy estimation.

Smart energy monitoring system (SEMO) (Ding et al., 2011) is a tool that continuously monitor and analyzes the energy consumption of smart-phone applications to rank them based on their energy consumption rate. The design of SEMO consists of three key modules: inspector, recorder, and analyzer. The inspector module monitors battery health status, voltage level, total battery charge remaining, and temperature of the smart-phone device during application execution. It notifies the application users when the device is about to reach its critical condition (e.g., low power, low temperature, over-utilization, and voltage level). The recorder is responsible for collecting various battery and program-related parameters such as execution time t, power consumption during time t, and a number of applications running during time t. The analyzer module utilizes the profile of the recorder and the power-remaining historic curve of the mobile battery to estimate the energy consumption rate of an application. SEMO assists application developers in ranking smart-phone applications based on energy consumption behavior to optimize power hungry applications. The critical aspect of the SEMO framework is its low-rate to capture power readings (1 sample per second). Thus, it has a high probability of missing many high-power consumption statistics. This framework does not consider the effect of the components that are invisible to the running OS.

Application energy profiling experiences the unique challenge of asynchronous power consumption behavior, wherein the state of the smart-phone component remains unchanged even beyond the lifetime of the entity that triggers it. Eprof (Yetim, Malik, & Martonosi, 2012), a fine-granular energy estimation tool, has resolved the

University of Malaya

asynchronous power consumption behavior issue of smart-phone components by employing the last-trigger accounting policy. Based on power state analysis, Eprof identifies the key reasons for asynchronous power consumption behavior, which include component/application tail power state, persistent power state wake-locks, and exotic components within smart-phones. The tail energy (e.g., disk, Wi-Fi, 3G, and GPS) of smart-phone components represent the state in which the activities of one entity (e.g., function, thread) push the smart-phone component to a high power state and the component remains in that state even after termination of the entity. Persistent state wake-lock occurs because of aggressive CPU screen sleeping policies as legacy OS exports wake-lock APIs to ensure that the components of the smart-phone are awake during application execution. Exotic components such as GPS, camera, and accelerometer, consume a significant amount of battery power because they are activated and deactivated by distinct entities. The advantage of Eprof is its capability to map the energy consumption of a smart-phone component at diversified granularity levels such as process, subroutine, thread, and system call. However, Eprof overlooked energy consumption by OS policies and poor software design.

P-top (Do et al., 2009) estimates the energy consumption of an application at process granularity. The design of P-top consists of four vital components: energy profiler daemon, in-memory data, display utility, and API-kit. The energy profiler daemon runs within the OS kernel and records the resource utilization level for several smart-phone components including CPU, hard disk, and network connectivity. The energy consumed by each smart-phone component during a specified time interval is calculated through power models, as presented in Eq. 2.3 and Eq. 2.4, prior to transferring it to the in-memory module for temporary storage. In the presented models, the parameters t_j, n_k, E_k, and P_{end} represent process execution

University of Malaya

time, count of transition between CPU states, energy consumption at a particular state, and power consumed during data sending via Wi-Fi link, respectively.

$$E_{CPU} = \sum_{a} p_j \times t_j + \sum_{k} n_k \times E_k \tag{2.3}$$

$$E_{NET} = t_{send} \times P_{send} + t_{receive} \times P_{receive} \tag{2.4}$$

The display utility uses collected execution log (from the in-memory module) to generate a detailed report that highlights the power consumption of several mobile components. P-top offers the API interface to enable application developers to access the power log of their desired processes. Consequently, application developers can optimize application scheduling and lightweight application design with the help of API interface module. This tool is helpful because it is embedded as an OS service. However, numerous factors such as GPS and LCD brightness, are disregarded during energy profiling. The asynchronous power features of mobile components are also overlooked.

Table 2.2 relates aforementioned state-of-the-art energy estimation schemes based on a set of parameters highlighted in thematic taxonomy. Table 2.3 presents a set of smart-phone components that existing schemes have considered during energy estimation of applications. Also, in Table 2.3 the hardware model and mobile type chosen by existing schemes during power consumption profiling are reported. The detail discussion on the aforementioned tables is presented in Section 2.2.3.

University of Malaya

2.3.3 Comparison of Smart-phone Components Power Measurement Based Energy Estimation Schemes

This section compares state-of-the-art smart-phone components power measurement based energy estimation schemes to highlight the commonalities and variances among existing schemes. The parameters to compare existing energy estimation schemes include power model type, methodology, granularity, training mode locality, power modeling approach, objective function, and design as shown in Table 2.2. Given below is an analysis of existing schemes based on each of aforementioned parameters.

2.3.3.1 Power Model Type

Attributes of power model type represents whether an estimation scheme has considered external physical measurement or self-metering based energy estimation paradigm. Existing energy estimation schemes such as DUT (Carroll & Heiser, 2010), Rice and Hay (A. C. Rice & Hay, 2010), Netw (A. Rice & Hay, 2010), Power Memo (Tsao et al., 2012), Browser (Thiagarajan et al., 2012), and Skype (F. Jiang et al., 2015), considered external measurement based power estimation mode. Alternatively, state-of-the-art energy estimation schemes such as PowerProf (Kjærgaard & Blunck, 2011), Power Booter (L. Zhang et al., 2010a), Se-same (M. Dong & Zhong, 2010), P-top (Do et al., 2009), Eprof (Yetim et al., 2012), and SEMO (Ding et al., 2011), have chosen self-metering based energy estimation mode. External physical measurement based energy estimation models are more accurate compared to self-metering based energy estimation due to error prone nature of fuel gauge's voltage and current sensors. However, external physical measurement based estimation requires lab setting environment to conduct experiments. The accuracy of external physical measurement based modeling highly depend on, (a) sample rate of power monitoring devices such as a multimeter, power meter, ammeter, (b) device

University of Malaya

Table 2.2: Comparisons of Smart-phone Components Power Measurements Based Energy Estimation Schemes

Reference	Power Model Type	Methodology	Power Log Source	Granularity	Training Mode Locality	Power Modeling Approach	Objective Function	Design
(Carroll & Heiser, 2010)	External Power Measurement	Utilization Based	Sense Resistor	Mobile Component and Mobile System	Off-Device	N/A	Where Mobile Battery Used?	Heavy weight
(A. C. Rice & Hay, 2010)	External Power Measurement	Trace Driven	Sense Resistor	Mobile Apps	N/A	N/A	Automated Energy Profiling	Heavyweight
(A. Rice & Hay, 2010)	External Power Measurement	Trace Driven	Sense Resistor	Mobile Apps	N/A	Deterministic	Automated energy Profiling	Heavyweight
(Tsao et al., 2012)	External Power Measurement	Trace Driven	V/I Clamps	Process and Function	Off-Device	N/A	Asynchronous Behavior of Wi-Fi Power Consumption	Heavyweight
(Thiagarajan et al., 2012)	External Power Measurement	N/A	Sense Resistor	Web Browser	Off-Device	N/A	Web Component Rendering Energy	Heavyweight
(F. Jiang et al., 2015)	External Power Measurement	Utilization Based	Sense Resistor	Skype, Text, Wype	Off-device	N/A	Characterize Power Consumption of Three Input Modes	Heavyweight
(Kjærgaard & Blunck, 2011)	Self-metering	Trace Driven	Fuel Gauge	API, Mobile Apps	Off-Device	Statical	Unsupervised Power Profiling	Heavyweight
(L. Zhang et al., 2010a)	Self-metering	utilization Based	Fuel Gauge	Mobile Apps	On-Device	Statistical	Online Power Model Generation	Heavyweight
(M. Dong & Zhong, 2010)	Self-metering	Trace Driven	Fuel Gauge	Application and System	Off-Device	Statistical	Self Power Model Construction	Heavyweight
(Ding et al., 2011)	Self-metering	N/A	N/A	Application	N/A	N/A	Smart Energy Monitoring	Heavyweight
(Yetim et al., 2012)	Self-metering	Trace Driven	Fuel Gauge	Application, System calls	Off-Device	Statistical	Energy Bug Finding	Heavyweight
(Do et al., 2009)	Self-metering	Trace Driven	Fuel Gauge	Process	N/A	N/A	Process Level Estimation	Heavyweight

University of Malaya

disruption rate during the training phase, (c) and calibration of the device under test. Existing energy estimation schemes considered various types of power measurement equipment to capture voltage and current drop during experiments such as PCI-6229 DAQ, PCI-MIO-16E-4, PCI-6115 DAQ, and PCI-MIO-16E-4 to predict smart-phone/application energy consumption. The accuracy of self-metering based energy estimation schemes depends on, (a) error rate of smart battery's voltage/current sensors, (b) fuel gauge's sensor updating rate and (c) mobile power API's access rate. Self-metering based estimation solutions do not require extra external hardware equipment to capture power readings as they use API to access voltage and current sensors. In terms of overhead, self-metering based estimation is more expensive than external physical measurement based method as estimation tool has to run on the smart-phone device to record the time-stamped power profile in parallel to an application under test. In terms of monitory cost, self-metering based estimation is cheep compare to external measurement based estimation.

2.3.3.2 Methodology

Attributes of methodology parameter describes whether an estimation scheme has chosen utilization based method or trace driven power modeling approach to estimate smart-phone application energy usage. State-of-the-art energy estimation schemes such as DUT (Carroll & Heiser, 2010), Power Booter (L. Zhang et al., 2010a), and Skype (F. Jiang et al., 2015) has considered utilization based power modeling for energy estimation of smart-phone applications. Alternatively, Rice and Hay (A. C. Rice & Hay, 2010), Network, Power Memo (Tsao et al., 2012), Browser (Thiagarajan et al., 2012), PowerProf (Kjærgaard & Blunck, 2011), Sesame (M. Dong & Zhong, 2010), P-top (Do et al., 2009), Eprof (Yetim et al., 2012), and SEMO (Ding et al., 2011), has considered event driven power model construction methodology to estimate application energy consumption. Utilization based type of power modeling requires low power sample rate of hardware equipment to access architectural level details such as cycles, cache, and other HPCs parameters. Moreover, utilization based models lack in handling non-linear relationship of the parameters. Alternatively, event based power models consider system calls, hardware/software operating modes, and APIs, to capture voltage/current drop. The advantage of event driven to utilization based power modeling is their capability to model non-linear relationship among power model entities. Moreover, the event

University of Malaya

driven based approach is more suitable to identify the energy bugs within smartphone applications.

2.3.3.3 Power Log Source

Power log source parameter describes the hardware equipment opted as a source to collect current and voltage drop readings. Existing state-of-the-art energy estimation schemes have either considered sense resistor or fuel gauge hardware sensors to access current and voltage drop. Estimation schemes such as DUT (Carroll & Heiser, 2010), Rice and Hay (A. C. Rice & Hay, 2010), Netw (A. Rice & Hay, 2010), Power Memo (Tsao et al., 2012), and Skype (F. Jiang et al., 2015), has considered a sense resistor to collect current and voltage drop during activity on mobile phone to construct power models. Alternatively, Power Booter (L. Zhang et al., 2010a), PowerProf (Kjærgaard & Blunck, 2011), Se-same (M. Dong & Zhong, 2010), Eprof (Yetim et al., 2012), and P-top (Do et al., 2009), has exploited fuel gauge sensors to log voltage and current drop using built in voltage/current sensors. Sense resistor based methodology uses an extra external resistor placed at mobile component's power supply rail to monitor voltage/current drop. The accuracy of this category of estimation schemes highly depends on the resistance of the sense resistor. Sense resistor based methodology is helpful when evaluating power consumption behavior of single mobile component; however, they are not accurate while monitoring the behavior of an application in terms of mobile components power usage. This is due to the fact that collectively instrumenting every component of the mobile phone for application energy estimation increases the complexity of circuits and total impedance. Alternatively, fuel gauge based estimation does not require external hardware resources and uses built in OS's power management APIs to collect voltage/current drop. However, fuel gauge based methods do not directly estimate

University of Malaya

per-mobile component energy. For a single mobile component power capturing, it switches off all the mobile components except one under analysis. However, some of the components reside on the same physical hardware and hinders fuel gauge based estimation method to measure current/voltage accurately (i.e., blue-tooth and Wi-Fi).

2.3.3.4 Granularity

Attributes of granularity parameter describes the extent to which an energy estimation scheme assesses energy consumption. Estimation schemes considered estimation at diversified granularity levels such as smart-phone application, function, thread, smart-phone device, web browser, and smart-phone components. Among all granularity levels, fine granular level estimation is more accurate. However, fine granularity level estimation (mobile-component, source code line, path) is resource expensive and also requires extensive profiling of target application/component for a long period of time (Carroll & Heiser, 2010), (Yetim et al., 2012), (Kjærgaard & Blunck, 2011)). Alternatively, coarse granularity based estimation ((A. C. Rice & Hay, 2010), (A. Rice & Hay, 2010), (F. Jiang et al., 2015), (Do et al., 2009)) requires few system resources, (b) limited energy profiling time, and (c) low resource monitoring. In comparison to software level granularity, estimating energy consumption at smart-phone components level such as Wi-Fi, LCD, Radio, CPU (DUT (Carroll & Heiser, 2010)) requires, (a) extensive profiling, (b) per-component power profile isolation, (c) and offline power trace analysis (regression analysis) to extract coefficient of power models. Using external measurement based method, estimating power consumption of an individual hardware component is inaccurate as external measurement based devices capture energy consumption at the system level. Moreover, estimating energy consumption at fine granular function level requires that

University of Malaya

the power capturing tool should offer high sample rate to cope with low execution time of fine granular operations. For instance, EM6000 multi-meter captures three power samples per second (chapter 3).

2.3.3.5 Training Mode Locality

Training phase is the most resource rigorous process during smart-phone components power modeling. In this phase, it estimates coefficient of mobile phone power models. Training phase is performed either on mobile-device or off-device. On-device training incurs high estimation overhead and quickly depletes mobile battery charge. Alternatively, off-device training augments device battery lifetime as it schedules power model's co-efficient finding process on the remote cloud or desktop systems. However, the drawback of off-device based computation is the high dependency on the external hardware. Existing state-of-the-art energy estimation schemes such as DUT (Carroll & Heiser, 2010), Power Memo (Tsao et al., 2012), Skype (F. Jiang et al., 2015), Eprof (Yetim et al., 2012), PowerProf (Kjærgaard & Blunck, 2011), and Se-same (M. Dong & Zhong, 2010), has scheduled resource intensive tasks on the desktop servers to reduce estimation overhead. Alternatively, Power Booter (L. Zhang et al., 2010a) has not relied on the external hardware for the processing of resource expensive tasks.

2.3.3.6 Power Modeling Approach

Attributes of power modeling approach parameter defines what approach an estimation scheme has considered while constructing power models for smart-phone components. Attributes of power modeling approach are divided into deterministic and statistical power modeling categories. Netw (A. Rice & Hay, 2010) followed deterministic approach to construct power models. Alternatively, numerous energy estimation schemes such as Se-same (M. Dong & Zhong, 2010), Power

University of Malaya

Booter (L. Zhang et al., 2010a), and PowerProf (Kjærgaard & Blunck, 2011), followed statistical modeling approach (i.e., linear regression) to find the coefficient of power models. However, using a statistical approach, power modeling is resource expensive unless resource expensive tasks are scheduled on off-device cloud servers. Deterministic based power modeling approach is only valid to investigate power modeling of smart-phone components. A deterministic approach based power models estimate energy consumption based on, (a) energy per state of the mobile component, (b) number of transitions between states, (c) and energy per transition among system states. Nevertheless, finding all this information consumes a significant amount of battery charge due to time agnostic training phase. Statistical methods are only applicable for software component power estimation and cannot be used for hardware component's power consumption estimation.

2.3.3.7 Objective Function

Objective function defines the core motivation for each of smart-phone application energy estimation scheme. State-of-the-art energy estimation schemes have targeted, (a) automatic power model generation, (b) low overhead based estimation, (c) asynchronous network component behavior monitoring, (d) digging loop holes within application, (e) self-power model construction, (f) web components rendering cost estimation, and (g) investigating the resource expensive activity within mobile phone among talk, text, and whype operations.

2.3.3.8 Design

Design parameter states the resource consumption behaviour of existing energy estimation schemes. Smart-phone components power measurement based energy estimation schemes are heavyweight by their design. The main reason for the heavyweight design of existing smart-phone components power measurement based

University of Malaya

Table 2.3: Comparison of Existing Energy Estimation Schemes based on the Power Models they Built for Smart-phone Components

	Smart-phone Components									
Reference	CPU	Wi-Fi	LCD	2G/3G	Blue-tooth	Compass	Smart-phone Model	Power Capturing Tool	Estimation Accuracy	Estimation Overhead
(Carroll & Heiser, 2010)	YES	YES	YES	NO	YES	NO	HTC Dream, Nexus one	PCI-6229 DAQ	NA	NA
(A. C. Rice & Hay, 2010)	NO	YES	NO	YES	NO	NO	T-Mobile (HTC) G1	PCI-MIO-16E-4	NA	NA
(A. Rice & Hay, 2010)	NO	YES	NO	YES	NO	NO	Magic, Hero	PCI-MIO-16E-4	NA	NA
(Tsao et al., 2012)	NO	YES	NO	NO	NO	NO	NA	EPA-400BMG	NA	380 sec
(Thiagarajan et al., 2012)	NO	NO	NO	YES	NO	NO	ADP2	Agilent 34410A	NA	NA
(F. Jiang et al., 2015)	NO	YES	YES	YES	NO	NO	Samsung galaxy S3	NI-USB 6008	91.69 %	NA
(Kjærgaard & Blunck, 2011)	YES	YES	NO	YES	NO	YES	N7, N8, C7	Not Applicable	97%	NA
(L. Zhang et al., 2010a)	YES	YES	YES	YES	NO	NO	HTC dream	Not Applicable	80%	NA
(M. Dong & Zhong, 2010)	YES	NO	NO	NO	NO	NO	N85, N900	Not Applicable	95%	NA
(Ding et al., 2011)	NO	NO	NO	NO	NO	NO	ZTE-X876	Not Applicable	NA	NA
(Yetim et al., 2012)	YES	YES	NO	YES	NO	NO	N/A	Not Applicable	94%	NA
(Do et al., 2009)	YES	YES	NO	YES	NO	NO	T42 laptop	Not Applicable	2W "error"	3% CPU

schemes is the dynamic analysis approach for estimation. Dynamic analysis engages resources of a smart-phone for a longer period of time and as a result, it makes the estimation tool heavyweight.

Table 2.3 highlights the smart-phone components power models constructed by existing energy estimation schemes. It states which of the energy estimation scheme has proposed power models for which of the smart-phone component. It also highlights the claimed accuracy and energy estimation overhead for existing energy estimation schemes. Table 2.4 compares state-of-the-art energy estimation schemes based on their quantitative findings. Moreover, it showed a set of benchmarks that are considered by the state-of-the-art energy estimation schemes during experimentation. For instance, DUT (Carroll & Heiser, 2010) reported that during run of Equake on HTC dream mobile phone, blue-tooth consumed 44.9 mW power. Similarly, Netw (A. Rice & Hay, 2010) states that during an idle state of a smart-phone with LCD set to its full brightness-level (LCD) magic phone consumes 1-1.5W power (on average).

University of Malaya

Table 2.4: Quantitative Findings of Existing Energy Estimation Schemes

Reference	Benchmark(s)	Key Finding (s)
(Carroll & Heiser, 2010)	Equake, Vpr, Gzip, Mcf	[Component Energy]:- Blue-tooth= 44.9 mW
(A. C. Rice & Hay, 2010)	TCPdump, Libpcap	[Network Energy]:-3G idle= 0.6 W 2G idle=0.7 W, Wi-Fi idle=0.9W
(A. Rice & Hay, 2010)	TCPdump	[Mobile Energy]:-Idle full-brightens= 1-1.5 W, Idle dim-light=0.3-0.7 W, Standby=0.1-0.4 W
(Tsao et al., 2012)	NA	[Apps Energy]:-Unmodified JPEG= 23.3 J Modified JPEG= 15.91 J
(Thiagarajan et al., 2012)	Aol, picasa, live.com	[Network Energy]:-3G Upload= 25 J for 26 KB 3G download= 15 J for 26 KB
(F. Jiang et al., 2015)	Oracle-modality	[Apps Energy]:-Apple rendering= 455 J, Wikipedia=35 J Picasa=15 J, Yahoo= 18.5 J
(Kjærgaard & Blunck, 2011)	NA	[Modalities Energy]: Swype= 0.57 J/char, type= 0.11 J/char Skype= 0.45 J/char
(L. Zhang et al., 2010a)	Synthetic benchmark	0.145 Error for 95% percentile
(M. Dong & Zhong, 2010)	JBenchmark , 3DMarkMobile	NA
(Ding et al., 2011)	NA	accuracy=95% for 1 estimation/second accuracy=88% for 1 estimation/10ms
(Yetim et al., 2012)	fft, filterbank, fmradio, matrixmult, audio-beam-former	NA

2.4 Code Analysis Based Smart-phone Application Energy Estimation Schemes

Code analysis based smart-phone application energy estimation schemes consider operation's energy cost to estimate energy consumption of an application. For a smart-phone application, each operation consumes unlike resource consumption; therefor, each operation has dissimilar CPU execution-time and power-consumption cost. This section presents a thematic taxonomy and analysis current state-of-the-art code analysis based energy estimation schemes.

2.4.1 Taxonomy of Code Analysis Based Energy Estimation Schemes

Fig. 2.4 presents a thematic taxonomy to classify existing code analysis based energy estimation schemes based on a set of parameters including abstraction level, instruction profiling method, program structure analysis, estimation overhead, profiled instruction type, and estimation granularity.

Attributes of *Abstraction Level* parameter defines whether an energy estimation scheme has considered architectural details or non-architectural designs for energy

University of Malaya

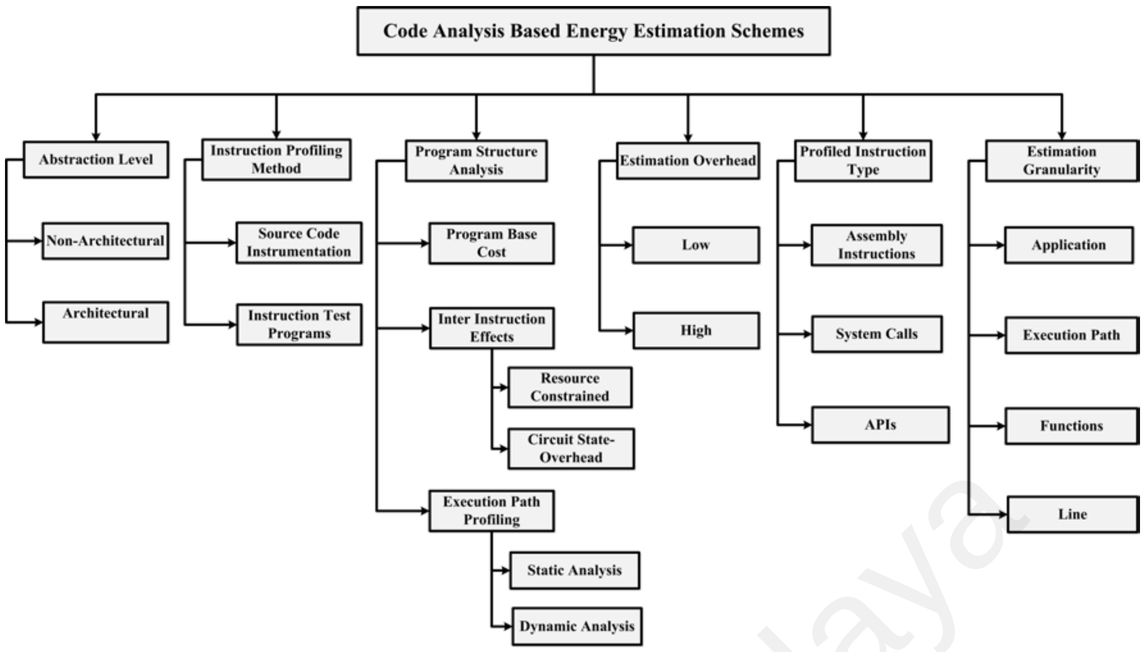

Figure 2.4: Taxonomy of Code Analysis based Energy Estimation Schemes

modeling of source code instructions. Attributes of abstraction level parameter consists of architectural details and non-architectural. In the case of architectural details, hardware performance counters (HPC) are used to construct instruction power model. However, for the non-architectural attribute, instruction power consumption is estimated based on the external power measurement tools such as multi-meter, power meter, and ammeter.

Instruction Profiling Method states the monitoring approach to capture power profile for instructions power modeling. The attributes of profiling method are classified into source code instrumentation and test program (s) categories. Source code instrumentation refers to the process of logging the performance of an application (power consumption) at different execution points. For instance, to find the energy cost of a function in an application, the entry and exit points of the target function is marked to record the timestamped power profile for that function. Test program refers to a set of statements performing the desired task when executed on smart-phone device. For instance, test program to examine the network connectivity embodies a set of statements continuously pinging target IP address.

University of Malaya

Attributes of *Program Structure Analysis* parameter defines the basic ingredients of an application that an estimation scheme has considered during energy estimation. Program's base cost represents energy cost of a program based on the individual base-cost energy of each instruction within the smart-phone application. The energy consumption of application truly depends on the path that an application takes at dynamic execution time. To find the execution path, smart-phone applications are instrumented prior to running on the mobile phone to record execution paths (offline mode).

Estimation Overhead highlights the energy consumption rate of energy estimation tool to assess energy usage-rate of smart-phone applications. Based on the architectural complexity of estimation tools, estimation overhead is attributed as low or high. Low value attribute defines that proposed scheme follows lightweight estimation design. Whereas, high attribute states that proposed scheme does not effectively utilize underlying smart-phone resources to estimate application energy consumption.

The source code of the application is of diverse types i.e., C, Java, or assembly, depending on the design of an application. Based on the nature of source code, type of an instruction (whose energy consumption is to be modeled) is different. *Profiled Instruction Type* parameter defines the type of instruction which is profiled to estimate application energy consumption. The attributes of profiled instruction type include assembly instructions, system calls, and APIs, as shown in Fig. 2.4. An assembly instruction presents the lowest system details when interacting with smart-phone. On the other hand, some of the code based analysis schemes has considered power profiling at system calls or API level.

Estimation Granularity describes the level at which energy estimation tool estimates energy consumption of an application. Estimation granularity includes ap-

University of Malaya

plication, line, path, and routines within an application. Application level energy estimation is most trivial as it requires considering every chunk of application's source code. The line represents the high level source-code line that is considered during profiling process (e.g., compound assignment statement, prefix, and increment statement). Execution path is also a complex entity as execution of a particular path depends on the data set in conditional statement.

Cycle Level energy estimation (Table 2.5) represents per CPU cycle energy consumption and is computed based on Eq. 2.5. In Eq. 2.5, V_{DD} represents the supplied voltage and t represents clock-time.

$$E_{cycle} = V_{DD} \times \int_0^t i(t)dt \tag{2.5}$$

Look-up table based energy estimation does not simulate architectural details for mobile application energy estimation. Rather, it uses already profiled architectural details to estimate application energy consumption. However, size of the look-up table greatly impacts estimation time (Table 2.5).

2.4.2 Review of Code Analysis Based Energy Estimation Schemes

This section briefly discusses code analysis based energy estimation schemes. Based on the design of energy estimation schemes, state-of-the-art code analysis based estimation schemes are classified into two categories. The categories of code analysis based estimation includes external setup based instruction power profiling and simulation based instruction power profiling.

2.4.2.1 External Setup Based Instruction Power Profiling

External setup based instruction power profiling paradigm exploits external hardware based laboratory setting environment to construct power models for instruc-

University of Malaya

tions. In this section, external setup based instruction power profiling based energy estimation schemes are discussed to highlight their advantages and shortcomings.

To estimate power consumption of embedded software, Tiwari et al. (Tiwari et al., 1994) has proposed an instruction power cost estimation model. The power cost of an assembly based instruction, called base cost energy, is estimated by creating and running test programs comprising several instances of the same instruction executing within a tight loop to capture current drop across the battery terminals. However, inside the application instruction execution order impacts estimation accuracy due to circuit state overhead. To handle the circuit state overhead, inter instruction effects of a few instructions is modeled. The proposed framework identifies basic building blocks within application followed by per block energy estimation to forecast energy demands of the application. The advantage of the proposed technique is simplicity and applicability towards identifying application energy consumption at earlier development stages. However, the proposed technique requires per CPU cycle energy consumption to model power cost of each assembly based instruction. However, finding low architectural details is computationally expensive and also it requires high sample rate based power meters. Also, in comparison to sequential based embedded applications considered in this study, the non-deterministic behavior of mobile application poses numerous challenges that require rigorous program analysis to estimate energy consumption. The proposed scheme has considered embedded sequential flow based application for estimation.

To investigate the power consumed by various activities involved in ARM instruction execution, Bazzaz et al. (Bazzaz, Salehi, & Ejlali, 2013) proposed a model to breakdown an instruction's power consumption into its sub activities. The proposed model calculates ARM-IS power consumption based on the aggregate sum of energy consumed by processor cores, memory-controller, and static RAM. The

University of Malaya

modeled parameters include register bank flip flops, the number of shift operations, instruction weight, and the hamming distance between instructions. Moreover, the proposed scheme modeled inter instruction effect for a set of instructions due to resource constrained nature of mobile phones. To capture the power consumption at mobile component granularity, the proposed approach (similarly to (Tiwari et al., 1994)) places a low-resistance resistor (1 ohm) at the power supply line of micro-controller. The proposed processor energy model for a single instruction is presented in Eq. 2.6, Eq. 2.7, and Eq. 2.8. In the mentioned equations, $E_{Control(code)}$ and E_{flash} represent the energy consumed in the memory controller and caused by the code in flash memory, respectively. Alternatively, E_{IF}, E_{ID}, and E_{EX}, represent the energy consumed during fetch, decode, and execute stages of CPU pipelines. E_{stall} represents the energy consumption during pipeline staling.

$$E_{fetch} = E_{Control(code)} + E_{flash} + E_{IF} \tag{2.6}$$

$$E_{decode} = E_{ID} \tag{2.7}$$

$$E_{execute} = E_{EX} + E_{Control(data)} + E_{Flash(data)} + E_{stall} \tag{2.8}$$

The proposed approach is useful when lower architectural details are required to optimize design of an application. However, to find energy for each mobile component, rest of the mobile components are switched off to minimize effect of estimation noise that as a result requires extensive offline regression analysis. Also, the proposed approach places 1 ohm resistor at each component power rail to capture current drop that makes proposed methodology suitable and adaptable only for lab experiments.

University of Malaya

A similar study to Bazzle et al. (Bazzaz et al., 2013) has been proposed by Khoshbakht et al. (Khoshbakht & Dimopoulos, 2015) to analyze ARM IS power consumption behavior when varying type of data within store memory operations. The proposed scheme enhanced Intel Pin cycle accurate estimation tool to instrument the store instructions. It was concluded that with more bit transitions (number of '1's) the energy consumption of store operation surges significantly. However, this scheme lack in considering effect of changing the storage location of data i.e., cache and memory. It also does not relate power consumption behavior of store operation to memory access pattern such as aligned, stride, and random.

To consider the complex nature of mobile applications, Shuai et al. (Hao et al., 2012b) proposed an estimation framework called eCalc to estimate energy consumption based on source code power profile of application. For instruction cost model construction, eCalc exploited an instrumented test program to profile power consumption of system calls, APIs, return function statement, and array creation process. In comparison to (Khoshbakht & Dimopoulos, 2015) and (Bazzaz et al., 2013), eCalc considered execution path of the application by instrumenting and running mobile application on mobile phone using application's uses cases to mark the execution path. The estimator module of eCalc exploited execution path and instruction power cost profile to estimate application energy consumption. The authors reported that eCalc is accurate up to 91.5% ground truth value. The proposed framework does not require any cycle accurate estimation simulator to profile instruction's energy consumption. However, path finding method of eCalc is both time consuming and resource expensive as it has considered dynamic code analysis based path profiling. Also, eCalc estimates energy consumption of only those applications whose source code is available.

A similar but more detailed study to eCalc (Hao et al., 2012b) has been pro-

University of Malaya

posed in (Hao et al., 2013) to combine per-instruction energy consumption and program analysis for mobile application energy estimation. However, in comparison to eCalc (Hao et al., 2012b), the proposed work has considered both path dependent and independent cost functions to improve estimation accuracy. The energy cost of path dependent instructions not only depend on type of instruction and state of mobile component, but it also relies on the weight of an argument such as number of packets transmitted on WAN interface and waiting for the state of the component. Also, the proposed framework annotated the application with energy cost to assist application developers in optimizing smart-phone application with limited efforts.

Li et al. (D. Li et al., 2013) combined hardware based and program analysis to estimate energy demands for an application's source code instructions. Proposed approach in comparison to (Hao et al., 2013) and (Hao et al., 2012b) has considered energy bugs that affect the accuracy of power models when profiling energy consumption of source code instructions (high level). This work too instrumented application and ran it on a smart-phone to capture time-stamped power log in parallel to execution path recording. Regression analysis on logged time-stamped power profile is performed to find per API energy consumption with high accuracy. Based on the execution paths, energy demands of the target application are identified. However, the instrumentation and running application requires a lot of system resources in addition to overhead imposed by instrumentation instructions.

Elens (Hao et al., 2012a) is a lightweight energy estimation framework as it does not depend on smart battery sensors for power estimation. The training phase of Elens captures application's execution paths when tested against test-case suites. Workload generator module is responsible generating load on the mobile phone to test application in a particular execution environment for a test stetting. It generates load based on the input of description document and software-artifacts

University of Malaya

modules. The analyzer module considers execution paths and SEEP based source code energy estimation to assess application energy consumption. The drawback of this scheme is its infeasible assumption that per-instruction power profile for any mobile everywhere is available.

Motivated by the work of Shuai et al. (Hao et al., 2013) and Hao et al. (Hao et al., 2012a), Reyhaned et al. (Jabbarvand, Sadeghi, Garcia, Malek, & Ammann, 2015) proposed EcoDroid, a system to rank mobile applications by estimating application energy consumption across several test cases. EcoDroid estimated application energy consumption using power prole for a set of system calls while automatically generating the test cases for path generations. However, EcoDroid considered power modeling at system call level and overlooked ne granular assembly instruction based energy estimation that gives more insights into the application's power consumption behavior.

Jayaseelan et al. (Jayaseelan et al., 2006) has proposed a method to estimate application energy consumption for ARM architecture. In comparison to (Hao et al., 2012b), (Hao et al., 2013), and (Jabbarvand et al., 2015), the proposed study has considered cache analysis of the application too during energy estimation. During application execution, the data and instructions are fetched from the main memory or cache depending on the frequency of memory access. The proposed scheme integrates cache analysis (simulator based) to the analysis of the application for worst case energy estimation. The proposed scheme lack in considering the impact of cache hierarchy and lines to cache hit/miss rate.

To profile energy consumption of low-level ARM-ISA, Sinha et al. (Sinha et al., 2003) considered Brutus SA-1100, a verification design platform, to link it with a desktop server (serial link). Moreover, the power supply to ARM core (residing within Brutus SA-1100) is externally provided through variable voltage supplier

University of Malaya

source. For power measurement, ammeter was used to log current readings. For each instruction, a test program comprising a set of same instances of the target instruction was designed. The proposed scheme concluded that branch instructions are most expensive ones among all instructions. The proposed scheme is accurate as it separates leakage current during the estimation process. However, the cost of the equipment is very high and also the proposed work does not consider execution path analysis.

Another approach proposed by Shao et al. (Shao & Brooks, 2013) has considered per-instruction energy consumption for *Xeo Phi* processor based on high performance counters (HPC). The proposed scheme considered the effect of the multi-threaded multi-processing environment to the accuracy of instructions energy consumption. The proposed framework accepts parameters including, performance counters, EPIs, instruction, and operand counts, to find the energy of a single instruction. The EPI is calculated based on the micro-benchmark designed to configure different cores. The reported estimation accuracy ranges from 1% to 5%. However, collecting performance counter creates dependency on some external tool (i.e., intel's VTune tool) to investigate benchmarks to collect required counter variables.

University of Malaya

2.4.2.2 Simulation Based Instruction Power Profiling

External setup based instruction power profiling method requires external power measurement tool to record timestamped power profile of an instruction test program to construct instruction power model. However, simulation based application's energy estimation does not require external power measurement tool. Rather, it creates target architecture environment inside a system to measure their behavior on different input data-sets. This section briefly describes application energy estimation literature that considered instruction power modeling based on HPCs simulated over different simulation tools.

Stanley et al. (Stanley-Marbell & Hsiao, 2001) proposed and implemented a cycle accurate simulator called *Myrmigki* to investigate power dissipation rate during instruction execution. The distinguishing features of Myrmigki include dynamic voltage frequency scaling (DVFS), per-cycle architectural configuration, and clock speed setting by modeling CPU cores, off-chip memory, and shared decode cache. The proposed modeled architecture is a 32-bit RISC architecture and is based on 5 stages pipeline. To estimate power consumption, simulator performs a look-up to access the current drop for the memory and CPU operations to calculate the per-cycle estimate. However, the proposed simulator is lacking in considering inter-instruction effects during application energy consumption estimation.

Vasilios et al. (Konstantakos, Chatzigeorgiou, Nikolaidis, & Laopoulos, 2008) presented an approach that has considered and modeled instruction energy in terms of number of accesses to memory, type of executed instruction, and analog to digital (ADC) conversion, as described in Eq. 2.9, Eq. 2.10, and Eq. 2.11, respectively. In the here mentioned equations, total energy consumed by RAM storage (E_{RAM}) depends on number of reads and write accesses. Also, $E_{micrcontr}$ is attributed in terms of a number of cycles for each instruction. For E_{ADC}, energy is estimated based on number of accesses to memory. The advantage of the proposed model is its flexibility to adjust with any simulator to perform desired operations. However, the proposed work lack in considering the impact of cache hit/miss on the power performance. Also, it does not consider circuit state overhead of assembly based instructions.

$$E_{RAM} = f(\#ReadAccesses, \#WriteAccesses, \#References) \tag{2.9}$$

University of Malaya

$$E_{micr-contr} = f(Ins\#1Cycles, Ins\#2Cycles..Ins\#5Cycles) \tag{2.10}$$

$$E_{ADC} = f(\#Accesses) \tag{2.11}$$

Another simulation based approach proposed by Zhao et al. (Zhao, Guo, Wang, & Chen, 2008) has considered per cycle energy modeling to find instruction's energy cost to track power consumed by function calls and execution paths. However, the authors overlooked the effect of cache hit-rate and miss-ratio to total energy estimation accuracy. Moreover, it overlooked circuit state overhead of the instructions. Alternatively, (Brandolese, Fornaciari, Salice, & Sciuto, 2000) modeled instruction's power cost based on the functional decomposition of tasks a microprocessor performs at the time of instruction test program execution. Authors constructed power models to breakdown instruction's power cost into, fetch, decode, and execute activities. Also, authors considered single and multi-processors to investigate power consumption for each functional unit during application execution.

Chen et al. (Chen, Irwin, & Bajwa, 1998) considered the format of instructions to estimate energy consumption at processor's control unit. It has considered transition signal among co-resident instructions to investigate inter-instruction power consumption effects. The proposed framework considered CPU instructions, DSP instructions, and pre-decoder, to identify the instruction (s) and power consumption behavior while placing instruction in different order. The complexity of the proposed technique is very high as it builds energy tables for all control units that takes reasonably larger time. Therefore, accuracy is traded by the estimation speed. The advantage of this technique is its capability to deal with both synthetic and kernel benchmarks.

University of Malaya

A similar to chen et al. (Chen et al., 1998) work but more detailed simulation has been reported by Chang et al. (N. Chang, Kim, & Lee, 2002) to investigate per cycle energy consumption for the ARM7 processor. The proposed simulator estimates architectural level energy based on the charge transfer within processor's circuit and also it is robust to noise. Simulator accurately estimates per-instruction energy to highlight the power consuming elements within an instruction such as instruction fetch address, op-codes, the number of registers, operand type, and data access. Efficiently optimizing these low level activities helps to optimize application power usage. However, the cache analysis is missing in this study that significantly impacts application performance.

Xiang et al. (Zhou, Guo, Shen, & Li, 2009) has proposed C based instruction's power cost model to estimate application energy consumption. The proposed study improved Embedded strong-ARM energy simulator (EMSIM) to propose *iEMSIM* that has considered improved power models for instruction power consumption as depicted in Eq. 2.12, Eq. 2.13, Eq. 2.14, and Eq. 2.15. Eq. 2.12 demonstrates energy consumption of a single instruction in terms of energy required during fetch, decode, and execute. Alternatively, for a single instruction, eq 2.13 computes total fetch energy based on energy consumed while fetching instruction form memory and a total number of CPU cycles for the fetch state. Eq. 2.13 exploits CPU cycles during which memory was active and energy consumed during decoding process (per cycle) to estimate decode stage energy consumption. Energy consumption for an execute state is accumulative sum of, (a) CPU energy in active and an idle state, (b) accumulated memory energy, (c) accumulated universal asynchronous receiver/transmitter (UART) energy, (d) and energy consumed due to peripheral input/output operations. The proposed simulator is good for analyzing limited instruction based operations. However, considering cache analysis, the designed

University of Malaya

approach is not effective as it overlooked cache analysis support.

$$E_{total} = E_{totalfetch} + E_{totaldecode} + E_{totalexecute} \tag{2.12}$$

$$E_{totalfetch} = E_{mem} \times N_{memcycles} \tag{2.13}$$

$$E_{totaldecode} = E_{decode} \times N_{decodecycles} \tag{2.14}$$

$$E_{tot\_exec} = E_{Proc} + E_{Idle} + E_{Memory} + E_{URT} + E_{Peri} \tag{2.15}$$

In (Mehta et al., 1996), an instruction power profile is proposed that simulates total energy consumption of instructions based on control paths and ALU. The advantage of the proposed framework is limited simulation speed as it requires a small table look up during instruction power estimation. The reported accuracy for instruction energy estimation is 8% for the IRSIM-CAP processor. IRSIM is a switch level circuit to simulate a digital circuit. However, the proposed simulator is only applicable for the desired processor and cannot be adopted widely.

2.4.3 Comparison of Code Analysis Based Energy Estimation Schemes

This section compares code analysis based smart-phone application energy estimation schemes using a set of comparison parameters such as abstraction level, instruction profiling method, estimation granularity, target processors, program structure analysis, estimation overhead, profiled instruction type, estimation granularity, claimed accuracy, and benchmarks as depicted in table 2.5. The following section briefly debates on each of the categories discussed above.

University of Malaya

Table 2.5: Code Analysis Based Smart-phone Application Energy Estimation and Modeling Schemes Comparisons

Ref.	Abstraction Level	Instruction Profiling Method	Program Structure Analysis	Estimation Overhead	Profiled Instruction Type	Estimation Granularity	Target Processor	Claimed Accuracy	Benchmark (s)
(Tiwari et al., 1994)	Architectural Details	Test Programs	Program Base cost Energy and Inter Instructions Effects	N/A	Assembly Based Instructions	Application	Intel 486DX2	N/A	Synthetic
(Bazzaz et al., 2013)	Architectural Details	Source Code Instrumentation	Inter Instructions Effects	N/A	Assembly Based instructions	Application	AT91SAM7	94%	MiBench
(Khoshbakht & Dimopoulos, 2015)	Non-Architectural	Source Code Instrumentation	N/A	N/A	Assembly Based Instructions	N/A	N/A	N/A	Spec2000 CPU
(Hao et al., 2012b)	Non-Architectural	Test Programs	Execution Paths	High	System Calls, APIs	Line,Path, Function	N/A	89%	Qsort, Md5, Matrix
(Hao et al., 2013)	Non-Architectural	Source Code Instrumentation Test Programs	Execution Paths	High	System Calls	Line, Path	N/A	90%	Text-program, Bubble, Sky-Fire
(Stanley-Marbell & Hsiao, 2001)	Architectural Details	N/A	Program Base Cost Energy	N/A	Assembly Based Instructions	Application	N/A	90%	Bubble, Heap, Quick
(Konstantakos et al., 2008)	Architectural Details	N/A	N/A	N/A	Assembly Based Instructions	Application	ARM7TDMI	N/A	N/A
(Zhao et al., 2008)	Architectural Details	N/A	N/A	High	System calls	Execution Paths, Routine	N/A	N/A	N/A
(Brandolese et al., 2000)	Architectural details	N/A	N/A	N/A	Assembly Based Instructions	N/A	i80486	N/A	N/A
(Chen et al., 1998)	Architectural Details	N/A	Inter Instructions Effects	High	Assembly Based Instructions	N/A	16 bit DSP	NA	G711, IIR filter, LMS_B
(N. Chang et al., 2002)	Architectural Details	N/A	Inter Instructions Effects	NA	Assembly Based Instructions	N/A	ARM7TDI	N/A	N/A
(Zhou et al., 2009)	Architectural Details	N/A	N/A	N/A	API	Application	N/A	N/A	N/A
(Jabbarvand et al., 2015)	Non-Architectural	Source Code Instrumentation	Program Base Cost Energy	NA	System Calls	Application	NA	NA	Weather
(Jayaseelan et al., 2006)	Architectural Level	N/A	Inter Instructions Effects, Execution Paths	High	N/A	Application	Power PC	N/A	Isort, fft, Fdct, Des
(Mehta et al., 1996)	Architectural Details	N/A	N/A	N/A	Low	Line	DSP	92%	N/A
(Sinha et al., 2003)	Non-Architectural	Test Programs	N/A	N/A	Assembly Based Instructions	Application, Line	itachi SH-4 microprocessors	97%	Alram, Mutex
(Shao & Brooks, 2013)	Architecture Details	N/A	N/A	NA	Assembly Based Instructions	Application	Xeon Phi	95%	Linpack, Scan
(D. Li et al., 2013)	Non-Architectural	Source Code Instrumentation	Execution Paths	High	System Calls, APIs	Line, Execution Path, Function	N/A	90%	BBC, Sky-Fire

2.4.3.1 Abstraction Level

Abstraction level parameter defines what type of information the instruction power models are made from to estimate smart-phone energy usage. The architectural detail attribute of abstraction level parameter demonstrates that estimation scheme has considered HPCs to construct fine granular power models to estimate energy requirements of a smart-phone application. However, finding architectural details is a slow process and also it requires offline simulation of required architecture to collect HPCs. Instruction power model is based on the number of CPU cycle to

University of Malaya

Table 2.6: Program Structure Analysis Based Comparison of Existing Code Analysis Based Energy Schemes

Reference	Program Base Cost	Inter Instructions Effects		Execution Paths		Cache Analysis
		Resource Constraints	Circuit State Overhead	Dynamic Analysis	Static Analysis	
(Tiwari et al., 1994)	✓	×	✓	N/A	N/A	×
(Bazzaz et al., 2013)	×	×	✓	N/A	N/A	×
(Khoshbakht & Dimopoulos, 2015)	×	N/A	N/A	N/A	N/A	×
(Hao et al., 2012b)	✓	N/A	N/A	✓	×	×
(Hao et al., 2013)	✓	N/A	N/A	✓	×	×
(Stanley-Marbell & Hsiao, 2001)	✓	N/A	N/A	N/A	N/A	✓
(Konstantakos et al., 2008)	N/A	N/A	N/A	N/A	N/A	×
(Zhao et al., 2008)	N/A	N/A	N/A	N/A	N/A	✓
(Brandolese et al., 2000)	N/A	N/A	N/A	N/A	N/A	×
(Chen et al., 1998)	×	✓	×	N/A	N/A	×
(N. Chang et al., 2002)	×	×	✓	N/A	N/A	✓
(Zhou et al., 2009)	N/A	N/A	N/A	N/A	N/A	×
(Jabbarvand et al., 2015)	✓	N/A	N/A	N/A	N/A	×
(Jayaseelan et al., 2006)	×	✓	×	N/A	N/A	✓
(Mehta et al., 1996)	N/A	N/A	N/A	N/A	N/A	×
(Sinha et al., 2003)	N/A	N/A	N/A	N/A	N/A	×
(Shao & Brooks, 2013)	N/A	N/A	N/A	N/A	N/A	×
(D. Li et al., 2013)	×	N/A	N/A	✓	×	×

fetch, decode, and execute a target instruction in addition to power consumption during cache hit/miss. Existing state-of-the-art energy estimation schemes such as (Shao & Brooks, 2013), (Sinha et al., 2003), (Jayaseelan et al., 2006), (Zhou et al., 2009), (Chen et al., 1998), (Tiwari et al., 1994), and (Stanley-Marbell & Hsiao, 2001), has considered architectural details attribute to construct per-instruction energy models. On the other hand, (D. Li et al., 2013), (Sinha et al., 2003), (Jabbarvand et al., 2015), and (Hao et al., 2013), constructed instruction power models based on profiling through extremal power measuring hardware device. Non-architectural based instruction power models collect timestamped power profile for a test program when it is executed on the target processor. However, profiling based instruction power modeling is costly as they require external power measurement tools such as muti-meter, monsoon power meter, and ammeter. Considering architectural based instruction power modeling, finding per cycle power consumption of embedded processors is a challenging task. The accuracy of non-architectural based instruction power modeling depends on the resistance of resistor attached to the battery terminals to capture voltage and current drop.

University of Malaya

2.4.3.2 Instruction Profiling Method

Attributes of this parameter defines the method chosen to estimate per instruction energy consumption. Attributes of instruction profiling method include test programs and source-code instrumentation for instruction power profiling. State-of-the-art smart-phone application energy estimation schemes such as (Tiwari et al., 1994), (Hao et al., 2012b), (Hao et al., 2013), and (Sinha et al., 2003) has considered test programs attribute to find per instruction energy consumption. Alternatively, (D. Li et al., 2013), (Jabbarvand et al., 2015), (Hao et al., 2013), and (Khoshbakht & Dimopoulos, 2015) has considered source-code instrumentation to find per instruction energy consumption. However, finding instruction's energy based on source-code instrumentation method requires high sample rate power capturing tools as low sample rate based tools can miss various low-level details. Also, it requires strict synchronization between time unit and power profile to eliminate any chance of error. Source code instrumentation based profiling method is usually preferred to observe energy usage behavior of instructions written in high-level languages (i.e., system calls, APIs). Test program based power profiling is suitable for instructions written in a low-level language such as assembly programs. Therefore, in comparison to test program based profiling, instrumentation method cannot profile power consumption for the activities whose runtime is very small. Moreover, running instrumentation based application also executes instrumentation code that itself consumes sufficient amount of smart-phone battery charge. In comparison, test program's overhead is low and also they can handle instructions with relatively small execution time. However, considering power profiling a source-code line, source-code instrumentation based profiling is more suitable compared to test program based profiling.

University of Malaya

2.4.3.3 Program Structure Analysis

Considering smart-phone applications, the design of recent applications is very complex due to their non-deterministic execution nature. While estimating smart-phone application energy consumption, the basic constructs considered during energy estimation by different schemes include, (a) program base cost energy ((Stanley-Marbell & Hsiao, 2001), (Tiwari et al., 1994)), (b) inter-dependent behavior of instructions ((N. Chang et al., 2002), (Chen et al., 1998), (Bazzaz et al., 2013), (Tiwari et al., 1994), (Jabbarvand et al., 2015)), and (b) execution path ((D. Li et al., 2013), (Hao et al., 2013), (Jabbarvand et al., 2015)) of the application. Finding base cost energy of the program is simple; however, inter-instruction effect requires checking all the possible combination of the instruction to find the overhead during changing their execution order. Path profiling is a resource expensive process as it runs instrumented application to record the execution path for all possible use cases in offline mode. However, such path profiling is expensive and time-consuming due to dynamic analysis method. Table 2.6 highlights a detailed analysis of program structures considered by various energy estimation schemes. As highlighted in the aforementioned table, the majority of energy estimation schemes has estimated program base cost energy of program. Also, existing energy estimation schemes have considered dynamic analysis approach to estimate execution paths for energy estimation of the smart phone application. Due to dynamic analysis approach, the design of energy estimation approach is heavyweight as it inefficiently exploits resources of a smart-phone device. Existing profiling based energy estimation schemes overlooked cache analysis of data and instructions to estimate energy consumption. Therefore, the estimation accuracy of existing schemes is significantly affected.

University of Malaya

2.4.3.4 Energy Estimation Overhead

A lightweight energy estimation framework efficiently utilizes mobile phone resources while estimating application energy consumption. The energy estimation overhead of existing energy estimation schemes is characterized as low or high based on the architectural design of proposed schemes. Estimation overhead is low when an estimation scheme schedules the resource expensive operations on nearby desktop or cloud servers to save mobile battery charge. In Metha et al. (Mehta et al., 1996) work, estimation overhead is low as this scheme requires a small look-up table to estimate energy consumption of different activities involved during instruction execution. Alternatively, the majority of proposed schemes have not considered their overhead in terms of how much battery power they have consumed during the estimation process. The stat-of-the-art schemes such as, (D. Li et al., 2013), (Jayaseelan et al., 2006), (Zhao et al., 2008), and (Hao et al., 2013), incur high estimation overhead due to dynamic profiling to mark execution paths of the application for all use cases.

2.4.3.5 Profiled Instruction Type and Estimation Granularity

Profiled instruction type parameter defines type of the instruction for which timestamped power profile is captured to estimate energy usage of the smart-phone application. Existing state-of-the-art smart-phone application energy estimation methods have profiled power consumption for, (a) assembly instructions, (b) system calls, (c) and APIs. Among all, assembly based power profile (i.e., (Konstantakos et al., 2008) and (Bazzaz et al., 2013)) gives lowest level of details while considering power consumption from software perspective. Moreover, system calls and APIs level power profiling (i.e., (D. Li et al., 2013; Hasan et al., 2016), (D. Li et al., 2013), (Jayaseelan et al., 2006), (Zhao et al., 2008), (Hao et al., 2013)) is useful when high

University of Malaya

level source code of the application is available to estimate application's energy consumption. However, assembly instruction power profiling gives more detailed and lower level insights of application power consumption behavior. Considering system calls and APIs, to profile source code of the application is required. However, having assembly power profile, the obj-dump of application's executable can be used to estimate application energy consumption (obj-dump of an application is in assembly).

2.4.3.6 Target Processor, Claimed Accuracy, and Benchmarks

ARM-IS power profile for all architectures is different due to high resource heterogeneity in their underlying architectures. Attributes of the target processor parameter represent the processor model that estimation tool has considered during energy consumption estimation for the desired benchmarks. The processor reported in different energy estimation schemes include Intel 486DX2, itachi SH-4 microprocessors, Power PC, ARM7TDI, and DSP. The application benchmarks considered by various energy estimation schemes includes BBC, Sky-fire, Lin-pack, scan, Spec2000 CPU, Mi-Bench, isort, fft, fdct, and DES. It is noticed that accuracy, as reported by the simulator based power models, is higher than the rest. However, simulators are comparatively slow and incur high energy estimation time.

2.5 Performance Analysis Tools for Smart-phone Applications

This section briefly discusses smart-phone performance analysis tools that assist in analyzing smart-phone applications.

University of Malaya

2.5.1 Val-grind

Val-grind[3] is a suite of tools that provides a number of debugging and profiling options to increase correctness and execution speed of a program. It is a dynamic analysis based debugging tool. The main tools in the Val-grind package include, (a) memory faults detector, (b) cache profile, (c) heap profiler, (d) branch predictor, and (e) thread bugs detector. The main debugging options of the Val-grind tool includes memcheck, cachegrind, callgrind, helgrind, massif, sgcheck, and lackey. Among all debugging options, memcheck and lackey are commonly used tools. Memcheck[4] assists to highlight the common errors usually in C/C++ programs that lead to system crashes or unpredictable behavior. It profiles the number of memory leaks occurred during program execution. Alternatively, Lackey[5] debugging tool profiles, (a) number of calls to a function, (b) number of branches taken/not-taken, (c) number of super-blocks entered, (d) the number of guest instructions executed, (e) and exit code of the program.

2.5.2 Trepn Profiler

Trepn Profiler is an on-device power modeling and performance measurement tool for energy estimation of the smart-phone device, samrt-phone component, and application. It constructs power models for smart-phone components based on on-device application profiling and fuel gauge sensor based power measurements. It assists in monitoring CPU utilization and network activity during application execution on the smart-phone device. It analysis performance of all the cores within smart-phone. For network activity, it generates the activity graph to highlight different states of network components such as, Off, Connect, Dormant, Idle, Send, Receive, and Ac-

[3]www.linux.die.net/man/1/valgrind

[4]www.valgrind.org/docs/manual

[5]www.valgrind.org

University of Malaya

tive. Trepn Profiler collects system data once per 100ms and on average consumes 40-05% CPU capacity during execution on smart-phone. Unlike Power Tutor, it follows self-modeling paradigm and exploits SOC estimation methodologies to estimate energy consumption during activity on smart-phone. However, for a device to accurately produce output, mobile development platform hardware instrumentation is required. To estimate utilization level of a hardware component, Trepn Profiler relies on Proc and other system files (Qian & Andresen, 2015; Ben-Zur, 2011).

2.6 Challenges and Issues for Smart-phone Application Energy Estimation

Fig 2.5 presents several research issues in this domain of research that hinders designing a resource efficient highly accurate energy estimation tool for smart-phone applications.

2.6.1 Effects of Battery Aging Factors on Estimation Accuracy

State-of-the-art smart-phone application energy estimation schemes forecast energy consumption of an application based on SOC estimation by considering the ratio of current battery charge capacity to the normal capacity. Normal capacity represents the total storage capacity of the battery and usually, it is defined by the manufacturer of the battery. However, SOC estimation accuracy is limited due to the architectural flaws in their construction. For instance, for each smart-phone model the SOC estimation error is different. SOC estimation methods i.e., Rint model and Counting model, does not truly represents the accurate capacity of the battery. This is due to the fact that these models do not consider the impact of aging, operational history, and size of the battery, during SOC estimation process. Historically, it is proven that for lithium based batteries, discharge-rate varies even with the same battery usage (known as a lithium-ion aging factor) (Hoque et al.,

University of Malaya

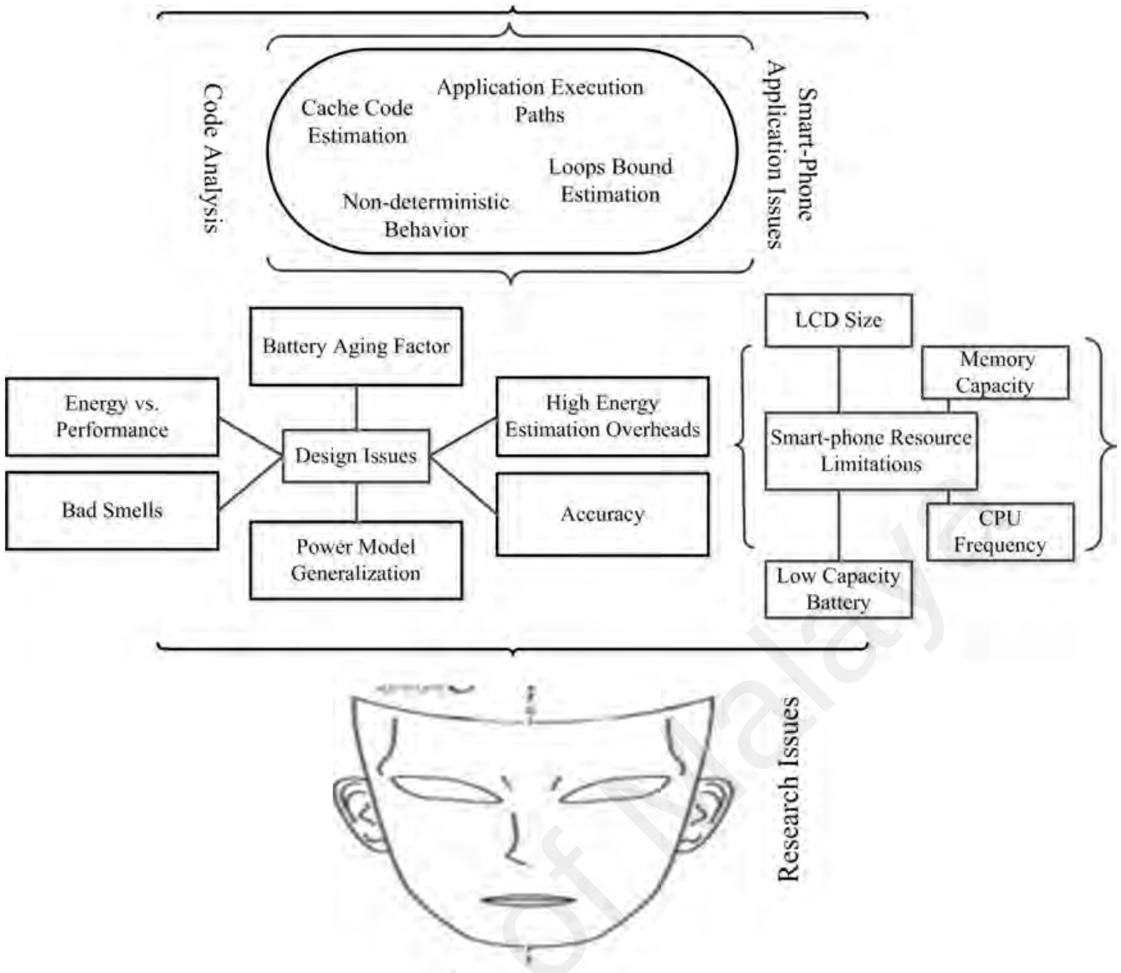

Figure 2.5: Open Research Issues and Challenges

2015; G. Dong, Chen, Wei, Zhang, & Wang, 2016; L. Lu et al., 2013). As a result, state-of-the-art SOC estimation methods report voltage-drop that is higher than its true value. Therefore, power models generated for one type of smart-phone or battery cannot be generalized for every model of batteries. For smart-phone batteries, Rint model based SOC estimation gives 11% estimation error. Also, counting based method incurs 5 to 2% error. Another factor that affects estimation accuracy is the high difference between smart phone's battery interface updating rate and OS's API access rate. For instance, the highest updating rate of smart battery interface is 4Hz, whereas, Linux OS updates the P-state residency for CPU component at 250Hz rate. The rate of the smart battery interface affects the power model generation time (Do et al., 2009; Hoque et al., 2015; Peltonen, Lagerspetz, Nurmi, &

University of Malaya

Tarkoma, 2016). Existing energy estimation schemes that considered built-in smart battery interface for energy estimation does not accommodate the error generated by SOC while reporting smart-phone application energy consumption.

2.6.2 Power-performance Trade-off

Resource hungry high-performance computing based applications consume a significant amount of smart-phone battery when they are executed on resource constrained smart-phone devices. Therefore, it is important for application developers to consider power performance tradeoff while optimizing smart-phone application design. For instance, while executing real-time communication applications, a quick application response time is highly crucial and consumes a significant amount of battery resource. Similarly, for security based applications, quantifying power performance trade-off during encryption decryption process is difficult because this process is highly complex. In particular, each line of application source code consumes a dissimilar amount of energy because of the variance in the number of operations being performed. Subsequently, the application segment that heavily uses smart-phone components drain more battery power because of the dynamic behavior of the application/mobile components. For example, pedestrian tracking applications use GPS module and network radio to calculate and transfer position updates to the monitoring server for continuous observation. In this scenario, increasing the time interval between position updates effects to the usability of application (low accuracy). Therefore, energy profiling designs should consider energy–performance trade-off while estimating and optimizing application energy consumption. For example, applying dynamic voltage frequency scaling (DVFS) increases battery life-time at the cost of application throughput. To date, limited attention has been paid to highlight the effects of applying DVFS optimization during power profiling on

University of Malaya

estimation accuracy for recent smart-phone devices.

2.6.3 Smart-phone Resource Limitations and Energy Bugs

Nowadays, smart-phones are commonly used portable devices due to their high applications in various computing domains. It is estimated that by 2017, smart-phone application market will reach to 77$ billion industry [6]. However, despite this tremendous growth still smart-phone usage is limited due to their resource constrained nature. Usually, smart-phones are equipped with limited battery capacity, limited storage, small size screen, low capacity processors, and highly vulnerable circuits. The average battery life-time of a smart-phone is very limited due to resource hungry nature of smart-phone applications. For instance, a context aware resource rich smart-phone application i.e., proximity based applications, enriches user experience at the cost of quick mobile battery charge depletion. The execution speed of smart-phones is very limited because of, (a) limited CPU clock rate (s), (b) small cache size to host frequently used data, and (c) limited RAM storage capacity (Abolfazli et al., 2014; W. Zhang, Wen, Wu, & Li, 2013). Moreover, smart-phone device's visualization quality is very poor because of the small size of the screen. Considering security perspectives, smart-phone devices are vulnerable compared to desktop servers. This is due to the fact that smart-phone devices offer low-resistance against sensitive vulnerable attacks to save battery charge. Also, high negligence of users during smart-phone usage makes it vulnerable to attacks. In addition to vulnerability issues, smart-phone applications sometime abnormally use smart-phone's battery charge due to energy bugs embedded within applications or hardware. Hardware energy bugs are difficult to track and mainly occur due to, (a) faulty batteries, (b) damaged mobile battery chargers, (c) infected memory

[6]www.entrepreneur.com/article/236832

University of Malaya

cards, (d) and damaged SIM cards (Pathak, Hu, & Zhang, 2011; Oliner, Iyer, Stoica, Lagerspetz, & Tarkoma, 2013; Monte, 2010). For software applications, within an OS changing OS configuration impacts the smart-phone battery power consumption rate. For instance, incorrect SetCPU configuration for kernel's over-clocking leads to a sudden rise in mobile battery power-usage rate. Similarly, infected smart-phone applications (bad smells) and frameworks are difficult to track as energy bugs do not affect the functioning of the application. A "no sleep" bug hinders a smart-phone component to go into a sleep state that as a result depletes smart-phone battery charge. A smart-phone application, with no sleep bugs, acquires a lock on smart-phone component and does not release it for a long period of time. Similarly, "sleep conflict" bad smell is due to a situation where an application acquires a lock on one of the smart-phone component and then CPU moves to sleep state without waiting for the application to release the component.

University of Malaya

2.6.4 Issues of non-deterministic behavior of Smart-phone Applications

This sub-section discusses the issues relating to the non-deterministic nature of smart-phone applications for static analysis based energy estimation (D. Li et al., 2013; Konstantakos et al., 2008).

2.6.4.1 Execution Flow Estimation

The battery usage of a smart-phone application profoundly depends on the execution path it selects at run-time. At run time, application execution path depends on the current input, use case of the application, and application's historic data. Dynamic application profiling based estimation methods instrument application to identify the execution paths by running it in offline mode (D. Li et al., 2013, 2013; Jayaseelan et al., 2006; Zhao et al., 2008; Hao et al., 2013). The energy estimation of the program is estimated based on the energy cost model and execution path marked during dynamic path profiling phase. However, the instrumentation and profiling phases are time consuming and resource expensive. Also, in order to use instrumentation based method, the source code of the smart-phone application should be available. In reality, due to application privacy issues, the source code of every smart-phone application is not available. As a result, dynamic analysis based path profiling has low scalability.

2.6.4.2 Loop Bounds Estimation

Within smart-phone application's source code, some portion repeatedly executes for a fixed number of time depending on the input to the application. Finding a solution to identify and estimate iteration of repetitive portion of code can augment estimation methodology while considering static code analysis based solutions (Blazy, Maroneze, & Pichardie, 2013; Hardy, Puaut, & Sazeides, 2016). The factor that affects loop bound estimation includes (a) initialization, (b) termination, and (c) growth rate of the variable. However, these three parameters are not known always depending on the structure and need of the program. Therefore, a method should be proposed to estimate bounds on loops for energy estimation (using static analysis).

2.6.4.3 Storage Access Estimation

Nowadays, smart phones are equipped with multi-level caches to speed up the application execution time by directly accessing data and instructions from the locally hosted cache. For architectural level details such as cache access rate, miss rate, simulators are used to track memory access pattern for smart-phone applications (Noguchi et al., 2016). However, simulation based solutions are offline and computationally very slow. Rather than opting dynamic code analysis, static code analysis method can be used to estimate memory access pattern of smart-phone application's code.

University of Malaya

2.6.5 Architectural Non-compatibility and High Energy Estimation Overhead

The instruction power profile for a set of instructions for heterogeneous smart-phone architectures is dissimilar because of their resource objectives. For instance, ARM7 is high performance whereas ARM15 is energy efficient architecture (Vasilakis, 2015). Estimation overhead is one of the most important issues in code based estimation domain. Dynamic profiling runs the application to analyze the source program for energy estimation. However, to analyze the program, the smart-phone application is instrumented that requires annotating the application. Also, at execution time, execution of annotation instructions itself consumes a sufficient amount of energy. Dynamic profiling increases estimation time and consumes significant energy.

2.7 Discussion

The recent trend to shift information access paradigm to smart-phone device calls for optimizing legacy applications for effective battery resource usage. Smart-phone application energy estimation creates an opportunity for developers to reconsider their application design at earlier development stages for effective battery resource usage. A smart-phone application energy estimation method either uses smart-phone components power consumption or code analysis based estimation models to forecast application energy consumption. Smart-phone component based power models are not highly accurate as they use SOC estimation to monitor smart-phone application power consumption. However, due to the architectural flaws in mobile battery construction, SOC estimation methodologies do not reports true capacity of mobile battery charge. On the other hand, hardware based mobile application energy estimation solutions are time consuming and resource expensive due to, (a) inter mobile component dependency, (b) the low sample rate of power measurement

University of Malaya

tools, (c) mobile component's wake-locks, and (d) energy bugs within mobile applications. Considering software aspects of mobile applications, code analysis assists estimating power consumption based on power cost model of instructions within application's source code.

Code analysis based energy estimation helps application developers to estimate application energy estimation at diversified granularity levels such as paths, lines, application, and functions. Smart-phone application energy estimation is based on power cost models for either high-level source code instructions or low granular assembly instructions. High level source code power cost models require source code of the application to estimate application energy consumption. However, assembly instructions power profile based estimation uses assembly code of the application for energy estimation. ARM-IS power cost profile is created using external hardware based power capturing tools (profiling based) or using cycle accurate simulators. Compared to profiling based power modeling, cycle accurate simulators are extremely slow. Based on the ARM-IS power cost profile and analysis of smart-phone application energy consumption is estimated. However, program analysis faces various challenges because of non-deterministic behavior of today's smart-phone applications. Current estimation methods exploit dynamic analysis of application to find the execution paths to estimate application energy consumption. However, dynamic analysis requires annotating source code of the application. Annotated code itself consumes a significant amount of energy to record execution paths of the application when executed on smart-phone. Also, dynamic analysis energy estimation schemes are heavyweight as they engages system resources for longer period of time. During application execution, a sufficient amount of energy is consumed that depletes mobile battery charge. As a result, estimation time and overall estimation overhead surges when considering dynamic analysis based smart-

University of Malaya

phone application energy estimation. In addition, for recent smart-phone devices, the inclusion of multi-level cache too increases the opportunity to accurately estimate smart-phone application energy consumption rate by identifying the storage location of source code within an application.

2.8 Conclusions

Smart-phone application energy modeling and estimation is an emerging area of research due to its long lasting applications in various resource critical domains such as MCC, Internet of things (IoT), mobile code optimization, energy bug detection, and hardware performance monitoring. This chapter has extensively reviewed smart-phone application energy estimation schemes to critically analyze them for highlighting their performance limitations. It has proposed thematic taxonomies to classify existing literature into several categories. Based on the thematic taxonomies, it has compared existing schemes to highlight the commonalities and variances among them. Finally, it has presented several open research issues and challenges that need further research to minimize energy estimation overhead.

Existing smart-phone application energy estimation schemes are classified in smart-phone components power measurement and code analysis based estimation categories. Performance overhead of smart-phone components power measurement based estimation is observed very high owing to its high energy estimation time. For smart-phone components power measurements based estimation, energy consumption estimation accuracy highly depends on, (a) fuel gauge updating-rate, (b) OS's API power capturing-rate, (c) SOC estimation error, (d) power measurement tool's accuracy and sample-rate, and (e) inter-dependency among smart-phone components. Code analysis based energy estimation of smart-phone applications too is of limited performance as it employs dynamic analysis method to estimate execu-

University of Malaya

tion paths of an application. Also, it overlooks the role of instruction/data storage location on the total energy estimation accuracy. The accuracy of smart-phone application energy estimation for code analysis methods depend on, (a) ARM-IS power cost profiling accuracy, (b) accurate execution paths estimation, and (c) precise loops bound estimation. Design of dynamic analysis based energy estimation scheme is heavyweight as it engages the resources of smart-phone for longer period of time. Based on the analysis of existing dynamic analysis based estimation schemes, it is concluded that there is a need of proposing a method to minimize the performance overhead of dynamic analysis methods for smart-phone applications energy estimation.

University of Malaya

CHAPTER 3: PROBLEM ANALYSIS

This chapter analyzes the performance overhead of existing dynamic analysis based energy estimation schemes to highlight their high energy estimation time, energy overhead, and resources consumption rate. It also analyzes the performance overhead of simulation based application profiling methods for smart-phone applications. For experiments, it considers standard benchmark applications to evaluate performance of dynamic analysis based energy estimation schemes. The performance analysis will reveal level of severity of estimation overhead associated with existing dynamic analysis based energy estimation schemes.

The rest of this chapter is organized as follows. Section 3.1 discusses the evaluation method, states benchmark applications used for experiments, and discusses differentiating features of selected energy estimation schemes. Section 3.2 discusses energy estimation time analysis for dynamic analysis based energy estimation schemes. Section 3.3 presents energy consumption analysis for chosen benchmark applications using dynamic analysis based energy estimation schemes. Section 3.4 debates on energy overhead for existing dynamic analysis based energy estimation schemes. Section 3.5 highlights resource consumption behavior of dynamic analysis based energy estimation schemes. Section 3.6 presents a detailed analysis on application components energy consumption and system architecture level profiling overhead. Lastly, this chapter provide concluding remarks of the performance evaluation in Section 3.7.

3.1 Experiments

This section briefly discusses the methodology, devices, benchmark applications, and dynamic analysis based energy estimation tools.

University of Malaya

3.1.1 Methodology

This study has performed experiments on a smart-phone device to analyze performance overhead of dynamic analysis based energy estimation schemes. It has selected ARM-7 based smart-phone device for the experimentation. Nexus One is a single core (1.0 GHz processor) device. Dynamic analysis based energy estimation schemes estimate application energy consumption based on power consumption of smart-phone components. Smart-phone components such as Wi-Fi, LCD, CPU, and sensors (e.g., Compass, GPS, accelerometer) contribute to the total energy consumption of a smart-phone. For the problem analysis, the proposed study has considered computing intensive applications. Computing intensive applications do not require user system interaction and smart-phone sensors to perform the required task. Therefore, prior to experiments, all the unnecessary sensors within smart-phone such as accelerometer, GPS, Wi-Fi, 3G Radio, and compass, were turned off to eliminate background noise. Moreover, OLED brightness level was set to the lowest state to avoid the risk of errors in estimation accuracy. In contrast to LCD, power consumption rate of OLED is affected by the color pattern of the text displayed on smart-phone screen. Also, all the unnecessary applications were closed (System and user applications) prior to the experimentation. To avoid the estimation error due to the power state of the battery, all experiments are performed with the smart-phone battery fully charged. This study has conducted each experiment 15 times to suppress the effects of background OS activities such as context switching and DVFS on the accuracy of the results. The average of 15 runs is reported for all results discussed in this chapter. Power Tutor and Trepn Profiler dynamic energy estimation tools are selected for energy estimation of the smart-phone applications. Power Tutor and Trepn Profiler runs in parallel with target smart-phone application

University of Malaya

for energy estimation.

External physical measurement based estimation method was chosen to analyze overhead of dynamic analysis based energy estimation tools. External physical measurement environment consists of a sense resistor attached to the smart-phone battery terminals (Tiwari et al., 1994). In external physical measurement environment, it attaches power measurement hardware device such as power meter and multimeter to the sense resistor to record the time-stamped power profile at desktop server as described in chapter 5 in details (Fig. 5.3). For the experiments, EM6000 multimeter is used to capture voltage and current drop across sense resistor. The resistance of sense resistor greatly impacts the accuracy of captured voltage/current drop; therefore, in this study 1-ohm resistor was used in the circuit. The voltage drop (in mV) across the sense resistor is profiled at the desktop server. As in current case, the resistance of sense resistor is 1 ohm, therefore current drop is exactly same as is the voltage drop (ohm's law $I = V/R$ (Ferry, 2012)).

University of Malaya

During application execution on smart-phone device, power measurement device (EM6000 multi-meter) records the voltage drop at the desktop server for post-processing to estimate energy consumption of the application. Based on the profiled voltage and current readings, Eq. 3.1 estimates application energy consumption (Tiwari et al., 1994).

$$E = P \times T, Where P = I \times V_{CC} \tag{3.1}$$

In Eq. 3.1, P and T parameters represent average power consumption and total execution time of application on smart-phone. For execution time of application, this study has used "time" utility of Ubuntu distribution of Linux to acquire system and user time for smart-phone application. P is estimated based on the voltage

(V_{CC}) and current drop (I) across the sense resistor.

The proposed study has selected energy estimation time, energy overhead, energy consumption, and resource consumption rate as the measures of performance for the dynamic analysis based energy estimation schemes.

3.1.2 Benchmark Applications

This section discusses the main features of benchmark applications selected for performance analysis of dynamic analysis based energy estimation schemes. All of the selected benchmark applications are compute intensive, open source, and operates on system memory. In addition, chosen benchmark applications consist of standard operations such as array creation, loops, function calls, object creation, arithmetic functioning, array index operations, sequence, and selection statements. To ensure the diversity of data set, it has considered non-homogenous benchmark applications in terms of their code size and type of operations for the experiments. In the following sections, the design of each of the benchmark applications is briefly discussed.

University of Malaya

3.1.2.1 NativeWhetstone2

NativeWhetstone2[1] benchmark application calculates the rating of target CPU in million of whetstone instructions per second (MWIPS). It primarily focuses measurement of basic floating point arithmetic and effectively separates results for the procedures in mega floating point per second (MFLOPS) operations. The modules within NativeWhetstone2 triggers processor (s) to perform required task by accessing data from RAM and cache storage. Eq. 3.2 and Eq. 3.3 present the basic array trigonometry operations defined within NativeWehtstone2 benchmark application. In the presented models, x_i, x, and y represents array indexes, integer, and floating

[1]www.roylongbottom.org.uk/android

point numbers, respectively.

$$X_1 = (x_1 + x_2 + x_3 - x_4) \times 0.5, X_2 = (x_1 + x_2 - x_3 + x_4) \times 0.5 \tag{3.2}$$

$$X = t \times arctan(2.0 \times sin(x) \times cos(x)/(cos(x \times y) + cos(xy)1.0)) \tag{3.3}$$

3.1.2.2 LivermoreLoops2

LivermoreLoops2 benchmark calculates instruction execution speed in MFLOPS and comprises of 24 kernels of numerical calculations to perform dissimilar CPU bound operations. Loops within LivermoreLoops2 benchmark composed of floating point numbers to perform floating point operations. The major operations performed by Livermoreloops2 benchmark includes matrix multiplication, Planckian distribution finding, discrete coordinates transport, general linear recurrence equation, and Monte carlo search loop. Linear recurrence equation model is presented in Eq. 3.4. In Eq. 3.4, c_i represents constant real number values, whereas, a_{n-k} represents the sequence of numbers (Batyuk, Schmidt, Schmidt, Camtepe, & Albayrak, 2009).

$$a_n = c_1 \times a_{n-1} + c_2 \times a_{n-2} + c_3 \times a_{n-3} + ...c_k \times a_{n-k} \tag{3.4}$$

3.1.2.3 LinpackSP2

LinpackSP2 (SP denotes single point operations) is a single precision floating point benchmark that outputs MFLOPS for floating point operations. The performance of this benchmark highly depends on $X[i] = X[i] * y[i] + m$ operation where a change in CPU instructions modifies result notably. LinpackSP2 is a compute-intensive bench-

University of Malaya

mark and multiplies two matrices as presented in Eq. 3.5[2]. During the experiments, the matrix sizes are chosen 200×201 for linpackSP2 benchmark.

$$(AB)_{ij} = \sum_{k=1}^{m} A_{ik} \times B_{kj} \tag{3.5}$$

In Eq. 3.5, AB is the target matrix that hosts multiplication result of A and B matrices. Moreover, values of i, k, and j represent row and column indexing of matrices (Batyuk et al., 2009).

3.1.2.4 Synthetic Test Programs

To extensively investigate the performance of CPU, a set of synthetic test programs comprising loops of different sizes performing basic arithmetic operations are designed. For instance, 10000K synthetic test program comprised of a set of statements performing basic arithmetic operations on unsigned integer values encapsulated within nested loops. The name of the test program is chosen based on the size of the loop within each test program. For instance, in the 10000K test program, the body of the loop is executed for 1.0×10^6 number of times. Alternatively, for the other test programs of this category such as 100000k, 1000000k, and 10000000k, it executes body of loop for 1.0×10^7, 1.0×10^8, and 1.0×10^{10} times, respectively.

3.1.2.5 Fast Fourier Transform

Fast Fourier Transform (FFT1) converts an input from original domain to frequency domain or vice versa. FFT1 operates on code for double and single precision fast Fourier transforms of size 1024bytes to 1048576bytes. During execution, this benchmark empowers the user to record the results in the FFT-tests.txt file for offline

[2]www.tomsguide.com

University of Malaya

analysis. FFT1 benchmark contains source code for multiple platforms such as ARM, MIPS, CISC; dynamically it decides which code to execute based on the architecture of the underlying device.

Eq. 3.6 highlights discrete Fourier transform (DFT) to convert time domain data into its equivalent frequency domain. In Eq 3.6, $K = 0 \dots N-1$ series represents the sequence of complex numbers. Alternatively, $X[k]$ defines the k_{th} harmonic number, and, $X[n]$ represents n_{th} input sample.

$$X_k = \sum_{n=0}^{N-1} X_n e^{-i2\times 22/7\times k\frac{n}{N}} \tag{3.6}$$

3.1.3 Dynamic Analysis Based Energy Estimation Tools

This study has selected two highly cited dynamic analysis based energy estimation tools called Power Tutor and Trepn Profiler for analysis. Both of these energy estimation tools run application for energy estimation. A short overview of the main features of selected energy estimation tools is as follows.

3.1.3.1 Power Tutor

Power Tutor follows on-device profiling, off-device modeling, and on-device estimation for energy measurement of smart-phone applications. In the profiling stage, it collects power states of smart-phone components. In the subsequent stages, it exploits power models of smart phone components to estimate energy consumption of smart-phone applications. Eq. 3.7 highlights energy estimation model of Power Tutor energy estimation tool to estimate energy consumption of the smart-phone application. The power coefficient values for smart-phone components such as β_{uh}, β_{ul}, β_{CPU}, β_{br}, and β_{3Gidle}, are computed in lab setting environment; whereas, it captures execution traces of smart-phone components based on android APIs. In

University of Malaya

Eq. 3.7, β parameter represents power coefficient for a particular execution state of smart-phone component. For instance, when CPU operates in high frequency mode ($freq_h$), power coefficient value is 4.34 W (β_{uh}) whereas, it is 3.42 W (β_{ul}) when it operates in low frequency execution mode (frq_l) (L. Zhang et al., 2010b). In the referenced equation, $GPSsl$, $Wi-Fi_l$, $3G_{DCH}$, and $brightness_l$, represent state of GPS, Wi-Fi, 3G, and LCD brightness, during application execution on smart-phone. Alternatively, $3G_{idle}$ and $3G_{FACH}$ represents idle and shared channel accessing states of 3G radio, respectively (L. Zhang et al., 2010b; Ren, Juarez, Sanz, Raulet, & Pescador, 2015).

$$AppE_{PowerTutor} = (\beta_{uh} \times freq_h + \beta_{ul} \times frq_l) \times util + \beta_{CPU} \times CPU_{on} + \beta_{br} \times brightness_l + \beta_{Gon} \times GPSon + \beta_{Gsl} \times GPSsl + \beta_{Wi-Fi-l} \times Wi-Fi_l + \beta_{Wi-Fi-h} \times Wi-Fi_h + \beta_{3Gidle} \times 3G_{idle} + \beta_{3G-FACH} \times 3G_{FACH} + \beta_{3G_DCH} \times 3G_{DCH} \tag{3.7}$$

3.1.3.2 Trepn Profiler

Trepn Profiler is an on-device power modeling and performance measurement tool for energy estimation of the smart-phone device, components, and applications. It constructs power models for smart-phone components based on on-device application profiling and fuel gauge sensor based power measurements. It assists in monitoring CPU utilization and network activity during application execution on the smart-phone device. It analysis performance of all the cores within smart-phone. For network activity, it generates the activity graph to highlight different states of network components such as, Off, Connect, Dormant, Idle, Send, Receive, and Active. Trepn Profiler collects system data once per 100ms and on average consumes 40-05% CPU capacity during execution on smart-phone. Unlike Power Tutor, it

University of Malaya

follows self-modeling paradigm and exploits SOC estimation methodologies to estimate energy consumption during activity on smart-phone. However, for a device to accurately produce output, MDP hardware instrumentation is required. To estimate utilization level of a hardware component, Trepn Profiler relies on Proc system file (Qian & Andresen, 2015; Ben-Zur, 2011).

3.1.3.3 External Physical Measurement

Trepn Profiler and Power Tutor are software-based energy estimation tools. External physical measurement based estimation method uses power measurement hardware device to estimate application energy consumption. External physical Measurement based estimation (also known as measurement method) attaches a small shunt (also called shunt) resistor to the power rail of the target hardware component for smart-phone sub-component energy estimation. Then, it interfaces power monitoring hardware device to sense resistor to record voltage/current drop across the resistor during activity on smart-phone. However, the accuracy of external physical measurement method depends on the resistance of the sense resistor.

3.2 Energy Estimation Time Analysis for Dynamic Analysis Based Estimation Tools

Energy estimation time is a projection of the time required to estimate energy consumption of a smart-phone application. Energy estimation time of dynamic analysis based estimation methods comprised of application profiling and analysis time. Application profiling time is the dominant factor in total estimation time as it depends on the total execution time of application on the smart-phone device. In this section, NWS, LPS, LML, and FFT1 are the acronyms used to represent NativeWhetstone2, LinpackSP2, LivermoreLoops2, and FFT1 benchmark applications.

Fig. 3.1 presents energy estimation time for a set of benchmark applications us-

University of Malaya

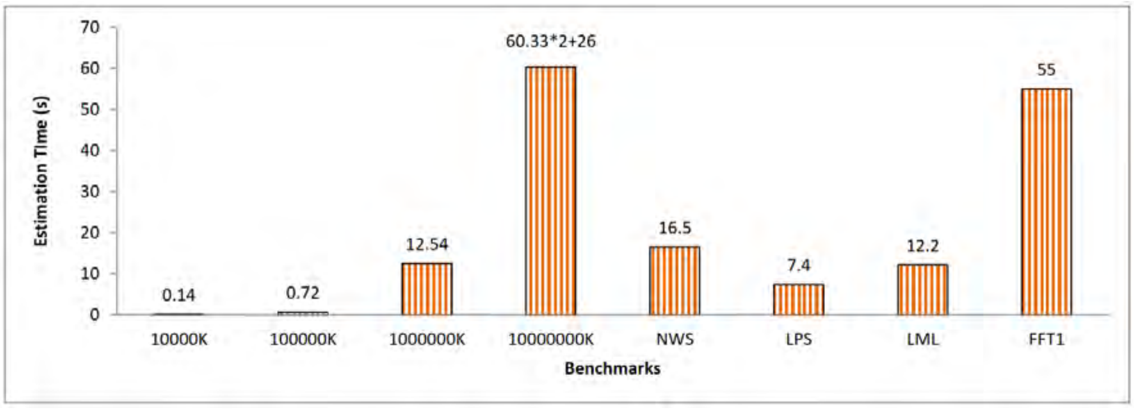

Figure 3.1: Estimation Time Analysis of Benchmark Applications

ing dynamic analysis based energy estimation tools. Energy estimation time of an application using dynamic analysis based energy estimation tool is estimated based on its wall clock execution time. For energy estimation time analysis, each benchmark application is ran 15 times on the smart-phone and the average of collected data is reported in this section. The average energy estimation time for 10000K, 100000K, and 1000000k is observed 0.14s, 0.72s, and 12.54s, respectively. Alternatively, for NativeWhetstone2, LinpackSP2, LivermoreLoops2, and FFT1, average energy estimation time is observed 16.5s, 7.4s, 12.2s, and 55s, respectively. The wall clock execution time of 10000K test program is minimal; therefore, energy estimation time of 10000K is very limited. Alternatively, the wall clock execution time of 10000000k is observed highest among all benchmark applications. The energy estimation time of 10000000k is noticed 146.33 seconds. The main reason of high estimation time is the large number of instructions CPU executed while running the 10000000k test program. Also, the energy estimation time of FFT1 is noticed high as this benchmark calculates Fourier transform of arrays of size 1048574, 524288, 1048576, 528288, and 1048576 elements. Energy estimation time for each of the benchmark application depends on the type and number of instructions executed within the application. For large data size applications such as 10000000k and FFT1, the more CPU clock cycles are required execution. As a result, estimation

University of Malaya

time of these application is very high. The descriptive statistics is applied to show the variance in findings of energy estimation time. Standard deviation from mean for all the test runs for test programs including 10000K, 100000K, 1000000k, and 10000000K, is noticed 0.008, 0.012, 0.015, and 2.740, respectively. Similarly, for NativeWhetstone2, LinpackSP2, LivermoreLoops2, and FFT1, standard deviation from mean is noticed 0.207, 0.534, 0.277, and 1.923, respectively. The main reason of high standard deviation from the mean for some of the benchmarks is due to background activities such as DFS, Garbage collection, and context switching of applications. The standard deviation from the mean and variance from the standard deviation among all selected benchmarks is noticed 48.46 and 2446.26, respectively. The main reason for this high standard deviation is because the selected benchmarks are not homogenous. For instance, the 100000000k test program is 1042.857 times faster than the 10000k program.

Table 3.1 highlights the effect of increasing data size on the total energy estimation time for matrix multiplication program. Matrix multiplication program multiplies two input matrices to store the results in a third matrix. In this experiment, the size of input matrices (number of Rows and Columns) is varied from 100×100 to 850×850 to analyze the impact of varying input data sizes to the total energy estimation time. For a specified input size of matrices for multiplication, total energy estimation time is recorded for 15 runs of the program. Standard deviation from mean and confidence interval for 95% percentile is calculated. It is noticed that increasing the input data size increases the total energy estimation time of matrix multiplication program. For input data sizes of 100×100, the average energy estimation time is found 0.30 seconds. The main reason of this small energy estimation time is the low application execution time on smart-phone device. For 15 runs of the program, standard deviation from the mean is noticed 0.0125.

University of Malaya

95% percentile is chosen to present confidence interval of the experiments. For 95% desired confidence interval, the energy estimation time for 100×100 based matrix multiplication program is noticed in the range of 0.30 ± 0.01. The energy estimation time for matrix multiplication program with different input matrix sizes including 150×150, 200×200, 250×250, 300×300, 350×350, and 400×400, is observed 0.75s, 0.99s, 2.34s, 3.43s, 4.17s, and 8.07s, respectively. The confidence interval for energy estimation time for aforementioned input data sizes is observed, ±0.02, ±0.03, ±0.17, ±0.82, ±0.01, and ±0.82, respectively. The main reason for comparatively higher values of energy estimation time for matrix multiplication program is the more CPU capacity required to process the data for matrix multiplication program. Increasing the matrix sizes increase the amount of data that CPU processes for multiplying matrices. With the increase in data size, the CPU execution time increases. As a result, estimation time of application increases as it depends on the execution time of application. Total energy estimation time for matrix multiplication program for large input matrix sizes such as 450×450, 500×500, 550×550, 600×600, 650×650, and 700×700, is observed high. The energy estimation time is noticed 10.15s, 14.15s, 20.08s, 26.14s, 33.402, 43.00s, and 53.71s, for matrix multiplication program with data sizes of 450×450, 500×500, 550×550, 600×600, 650×650, and 700×700, respectively. Standard deviation from the mean for energy estimation time readings for 15 runs of matrix multiplication with 450×450 input matrix size is observed 0.00896. For 95% percentile, the energy estimation time is observed 10.15 ± 0.01 for matrix multiplication program with matrix sizes of 450×450. Similarly, for matrix input sizes of 500×500, 550×550, and 600×600, the estimation time is observed 14.15 ± 0.11, 20.08 ± 0.73, and 26.14 ± 0.35, respectively (95% percentile). Considering very large input matrix sizes such as, 700×700, 750×750, 800×800, and 850×850, the energy estimation time is 43.00s, 53.71s,

University of Malaya

Table 3.1: Energy Estimation Time Analysis for Power Tutor Estimation Tool for Matrix Multiplication Program

Matrix Size	Estimation time (s)	Standard Deviation	Confidence Interval
100×100	00.30	0.0125	±0.01
150×150	00.75	0.0206	±0.02
200×200	00.99	0.0321	±0.03
250×250	02.34	0.1744	±0.17
300×300	03.43	0.8392	±0.82
350×350	04.17	0.0136	±0.01
400×400	08.07	0.8406	±0.82
450×450	10.15	0.0089	±0.01
500×500	14.15	0.0068	±0.11
550×550	20.08	0.7461	±0.73
600×600	26.14	0.3540	±0.35
650×650	33.40	0.4508	±0.44
700×700	43.00	0.7871	±0.77
750×750	53.71	1.0098	±0.99
800×800	75.16	0.7954	±0.78
850×850	84.96	0.7452	±0.73

75.16s, and 84.96 s, respectively. Standard deviation from the mean for very large size matrices is estimated 0.78716, 1.00983, 0.79541, and 0.75521, respectively. For very large input matrix sizes, the confidence interval for the execution runs is estimated, ±0.77, ±0.99, ±0.78, and ±0.73, respectively. In comparison to large size matrices, the estimation time for very large size matrices is very high because of significant increase in CPU execution time.

From above analysis, it is observed that increasing the data size for an application significantly increase the total energy estimation time of an application. During multiple execution runs of a program on smart-phone, estimation time varies with limited margin depending on the OS activities scheduled in the background during program execution on smart-phone device. Energy estimation time for Trepn Profiler is exactly same as estimated using Power Tutor energy estimation tool.

University of Malaya

3.3 Energy Consumption Analysis for Dynamic Analysis Based Estimation Tools

Energy estimation accuracy states how much accurate the findings of a particular estimation tool are relative to an external physical measurement method. Prior to smart-phone application energy consumption estimation, to ensure high estimation accuracy, system baseline power is captured to subtract it from the estimated value to improve the estimation results. Baseline power represents the energy consumed by the smart-phone during no activity period. To capture baseline power, the current study has used external physical measurement setup.

Smart-phone application triggers hardware components to perform a set of activities. Eq. 3.8 highlights the total energy consumption of an application on a smart-phone. In Eq. 3.8, E_{CPUAPP} represents total energy budget of application consumed by the CPU component; whereas, $E_{Wi-FiAPP}$ represents Wi-Fi energy consumption during discovery and connection establishment phases. E_{byte} defines the total energy per-byte data transfer over network connection from mobile phone. In Eq. 3.8, $tcpBytReceived + tcpBytSent$ demonstrates total data sent/received on the network link (Hoque et al., 2015; A. Ahmad, Paul, Rathore, & Rho, 2015).

$$E_{App} = E_{CPUAPP} + E_{wakeLock} + E_{Wi-FiAPP} + (tcpBytReceived + tcpBytSent) \times E_{byte} \tag{3.8}$$

Eq. 3.9 demonstrates that CPU energy cost depends on the total execution time of application when executing user and system code. Alternatively, $P_{Speed-i}$ represents power consumption of CPU executing at speed i.

$$E_{CPUApp} = \sum_{i=1}^{N} \frac{T_{speed-i}}{\sum_{i=1}^{N} T_{speed-i}} \times (T_{appcode} + T_{systCode}) \times P_{speed-i} \tag{3.9}$$

University of Malaya

Smart-phone components wake lock energy "$E_{wakeLock}$" as defined in Eq. 3.10 depend on the time the component remained lock in waiting for state and power associated with that lock. Alternatively, $E_{Wi-FiApp}$ (Eq. 3.11) depend on the amount of energy application consumes during scan time ($P_{Wi-Fi_{SCAN}}$) and idle time of Wi-Fi component ($P_{Wi-Fi_{ON}}$). Lastly, $T_{Wi-FiAPP}$ represents the time for which Wi-Fi discover the network connections.

$$E_{WakeLOCK} = P_{Wake-LOCK} \times T_{Wake-LOCK} \tag{3.10}$$

$$E_{Wi-FiApp} = T_{Wi-FiAPP} \times P_{Wi-Fi_{ON}} + T_{Wi-Fi_{SCAN}} \times P_{Wi-Fi_{SCAN}} \tag{3.11}$$

During data transfer via network link, energy consumption highly depends on the number of bytes transmitted and received on the radio. The energy cost of E_{byte} depends on, (a) power consumption of mobile radio and Wi-Fi in active states, (b) transfer rate, and (c) total data transferred or received using Wi-Fi and radios as shown in Eq. 3.12. Eq. 3.13 models GPS energy consumption. In Eq. 3.13, P_{GPS} and T_{GPS} parameters represent GPS power consumption and total time for which application accesses GPS module.

$$E_{byte} = \frac{\frac{P_{Wi-Fiactive}}{Wi-FiBps} \times Wi-Fi_{Data} + \frac{P_{radioactive}}{MobBps} \times MobData}{Wi-Fi_{DATA} + mobDATA} \tag{3.12}$$

$$E_{GPS} = T_{GPS} \times P_{GPS} \tag{3.13}$$

Fig. 3.2 highlights energy consumption of different components of a smart-phone device captured through Power Tutor for a set of benchmark applications. The energy consumed by the smart-phone components highly depends on the type of

University of Malaya

operations within a smart-phone application. For instance, in all of the chosen benchmark applications, none is performing any network activity. As a result, the energy cost for Wi-Fi and 3G for all benchmarks is zero. The major cost for selected benchmark is by CPU component of smart-phone device. LCD energy consumption too is low as this study has estimated energy consumption with brightness level adjusted to its minimum state. All the chosen benchmark applications are computational bounds, therefore, 85-92% of the energy budget of benchmarks is consumed by CPU component.

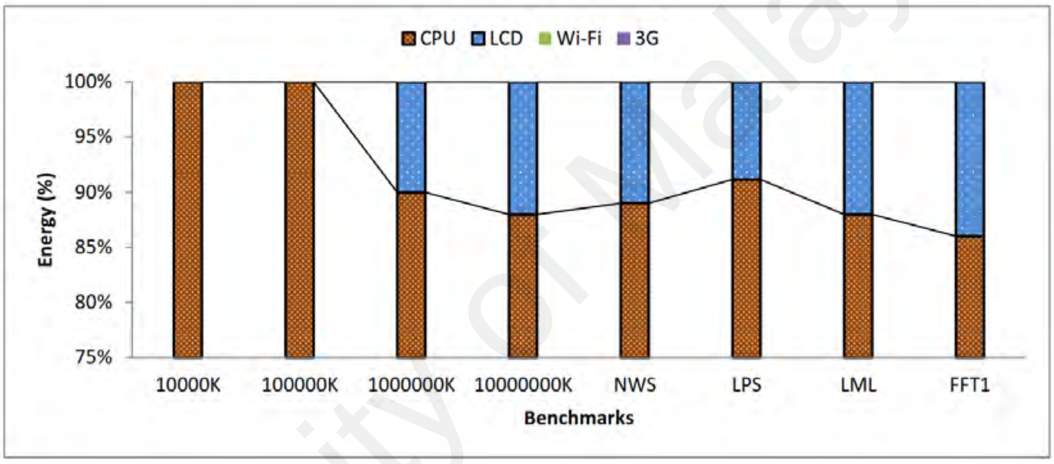

Figure 3.2: Smart-phone Components Energy Consumption Break-down for Benchmark Applications

Fig. 3.3 demonstrates energy consumption for a set of benchmark applications estimated using the Power Tutor energy estimation tool. Energy consumption of all selected benchmark applications is different and varies with execution time of benchmark application. Using Power Tutor, the estimated energy for 10000k, 100000k, 1000000k, and 10000000k, is observed 0.006j, 0.035j, 6.1j, and 70.4j, respectively. Alternatively, for NativeWhetstone2, LinpackSP2, LivermoreLoops2, and FFT1 benchmarks, energy is estimated 8.02j, 3.4j, 5.96j, and 37j, respectively. Among all benchmark applications, 10000k has consumed lowest energy whereas 10000000k has consumed the highest amount of energy when executed on the smart-phone de-

University of Malaya

vice. During multiple runs of benchmark application, the standard deviation from mean was observed minimal. For 10000k, 100000k, 1000000k, and 10000000k, standard deviation from mean is observed $9.5E^{-6}$, 0.001, 0.045, and 3.95, respectively. Similarly for remaining benchmarks in Fig. 3.3, standard deviation is noticed 0.58, 0.20, 0.12, and 2.25, respectively. The main reason of comparatively high standard deviation (for 10000000k test program) is the background noise and effects of battery discharge rate when it runs for a longer period of time. Due to high dissimilarity among chosen benchmark applications, the standard deviation from the mean for the findings of chosen benchmark applications is observed 24.87515 (high). The variance from the standard deviation is noticed 541.426.

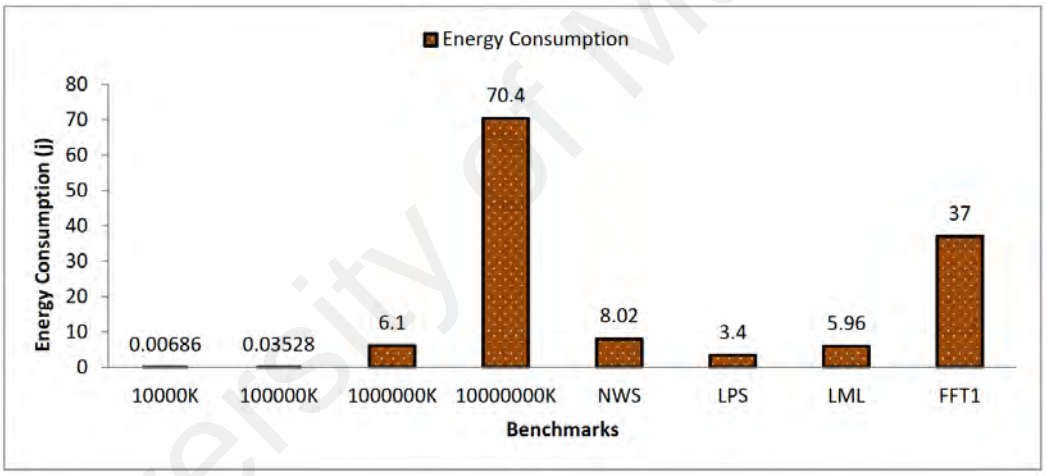

Figure 3.3: Power Tutor based Energy Estimation for Benchmark Applications

Fig. 3.4 highlights energy consumption for a set of benchmark applications estimated using Trepn Profiler energy estimation tool. Based on Trepn Profiler, the estimated energy for 10000k, 100000k, 1000000k, and 10000000k, is observed 0.008j, 0.041j, 7.9j, and 82.776j, respectively. Alternatively, for NativeWhetstone2, LinpackSP2, LivermoreLoop2, and FFT1 benchmarks, energy is estimated 10.54j, 4.8j, 7.09j, and 44.52j, respectively. Among all benchmark applications, 10000k has consumed lowest energy whereas 10000000k has consumed the highest amount of energy when executed on smart-phone. Standard deviation from the mean for 15 runs

University of Malaya

for each benchmark application is calculated. For instance, for 10000k, 100000k, 1000000k, and 10000000k, standard deviation from mean is observed $2.2E^{-6}$, 0.021, 0.27, and 4.01, respectively. Similarly, for remaining benchmarks, the standard deviation is noticed 0.36, 0.93, 0.89, and 3.19, respectively. The main reason of high standard deviation (in few cases) is the background noise and effects of battery discharge rate when it runs for a longer period of time. The standard deviation from the mean is noticed very high due to high dis-similarity among the selected benchmark applications. It is estimated 29.20. On the other hand, variance from standard deviation is noticed 746.53.

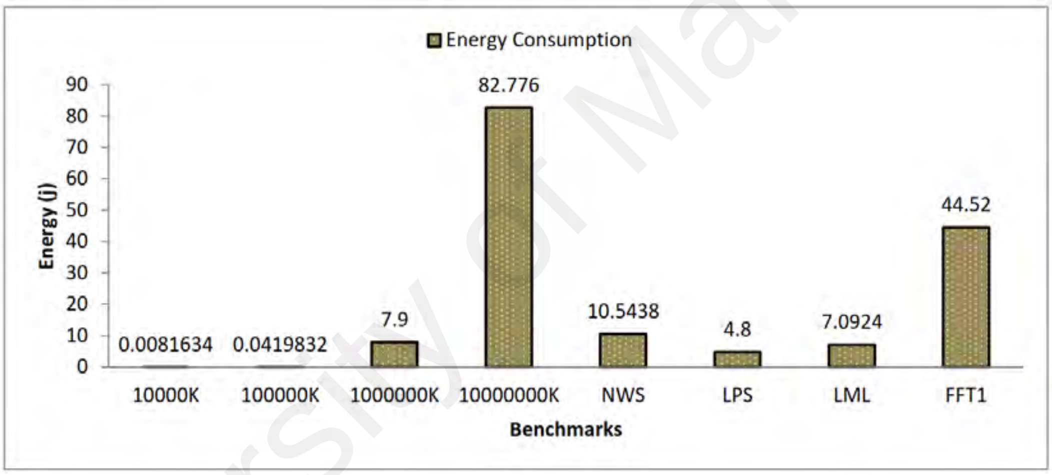

Figure 3.4: Trepn Profiler based Energy Estimation for Benchmark Applications

Fig. 3.5 compares the findings of Power Tutor and Trepn Profiler energy estimation tools. As can be seen from Fig. 3.5, Trepn Profiler estimated higher values of energy consumption for benchmark applications than Power Tutor based estimation. The main reason for this behavior is the difference in energy estimation methods opted by both of the selected energy estimation tools. Another reason of Trepn Profiler's high energy estimation values is this that it is originally designed for qualcomm processors and is low accurate for chosen smart-phone device compared to Power Tutor energy estimation tool. Fig. 3.6 highlighted the total difference in energy estimation values. The estimation difference in the estimated values for

University of Malaya

chosen benchmark applications ranges from 22 to 14% as shown in Fig. 3.6. The estimation difference is highest for LinpackSP2 benchmark and is lowest for the 100000000K test program.

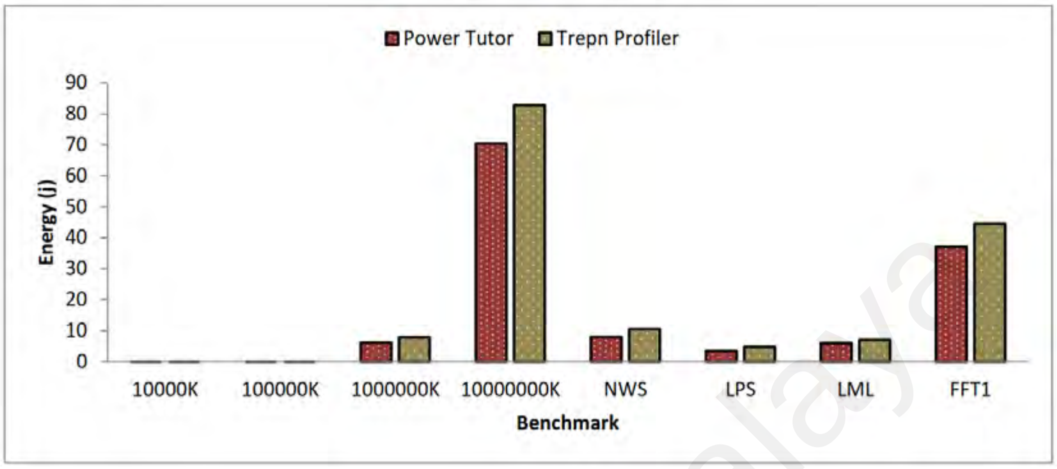

Figure 3.5: Power Tutor vs. Trepn Profiler

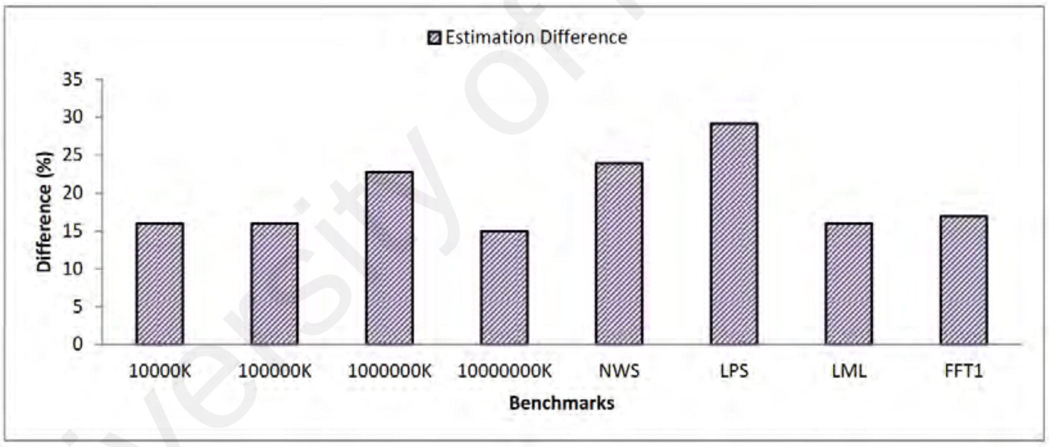

Figure 3.6: Power Tutor vs. Trepn Profiler Estimation Accuracy Difference Analysis

Table 3.2 presents energy consumption behavior of matrix multiplication program for different data sets estimated through Power Tutor energy estimation tool. For a specific input size of matrices for multiplication, total estimated energy is recorded for 15 runs of the program. Standard deviation from mean and confidence interval for 95% percentile is calculated. It is observed that increasing the input data size linearly increases the total energy consumption of matrix multiplication program. For matrix sizes of 100×100, the average energy consumption is found

University of Malaya

0.25 j. For 15 runs of the program, standard deviation from the mean is noticed 0.021 for 100×100 matrix multiplication. For desired confidence interval, the estimated energy for 100×100 matrix multiplication program is noticed in the range of 0.15 ± 0.02 j. The main reason of small value of energy consumption is the lower execution time of matrix multiplication program with 100×100 matrices. CPU took 0.30s only to execute 100×1000 matrix multiplication program. As a result, energy consumption of 100×100 based matrix multiplication program is very limited.

The energy consumption estimation for matrix multiplication program with different input matrix sizes including 150×150, 200×200, 250×250, 300×300, 350×350, and 400×400, is observed 0.47j, 0.88j, 1.17j, 1.71j, 2.24j, and 4.02j, respectively. The confidence interval for energy consumption statistics for aforementioned input data sizes is ±0.02, ±0.01, ±0.031, ±0.10, ±0.13, and ±0.12, respectively. In comparison to 100×100 size matrices, energy consumption of 150×150, 200×200, 250×250, 300×300, 350×350, and 400×400 size matrices is higher. The main reason of difference in energy consumption is the high execution time of former sizes of matrices for matrix multiplication program. For instance, the execution time of 250×250 based matrix multiplication program is 87% higher than 100×100 size based matrix multiplication program. Due to this execution time difference, the energy consumption of 250×250 based matrix multiplication program is higher than 100×100 based matrix multiplication. The more is the execution time of an application, higher is its energy consumption. Total energy consumption for matrix multiplication program for large input matrix sizes such as 450×450, 500×500, 550×550, 600×600, 650×650, and 700×700, is observed very high as shown in Table 3.2.

The energy consumption is noticed 4.97j, 7.01j, 10.04j, 12.93j, 16.53j, 21.03j, and 51.83j, for matrix multiplication program with data sizes of 450×450, 500×500,

University of Malaya

550×550, 600×600, 650×650, and 700×700, respectively. Standard deviation from the mean for energy consumption readings for 15 runs of matrix multiplication with 450×450 input matrix size is observed 0.467. For 95% percentile, the energy estimation time is observed 10.15 ± 0.01 for matrix multiplication program with matrix sizes of 450×450. The energy consumption of large sized matrices is high because of the extended execution time of matrix multiplication program with increasing matrix sizes. Smart-phone resources such as CPU, memory, buses, cache, and LCD are the major entities contributing to the total energy consumption of matrix multiplication program. With high matrix sizes, the system resources remain engaged for a longer period of time. As a result, for large data sized matrices, the energy consumption is high. Considering very large input matrix sizes such as 700×700, 750×750, 800×800, and 850×850, the energy consumption is estimated 21.03j, 51.83j, 66.52j, and 80.48j, respectively. Standard deviation from the mean for very large size matrices is estimated 1.77j, 1.108j, 2.48j, and 1.70j, respectively. For very large input matrix sizes, the confidence interval for the execution runs is estimated, ±1.77, ±1.37, ±3.09, and ±2.11, respectively.

Table 3.3 presents an analysis of energy consumption behavior of benchmark applications estimated through Trepn Profiler when input data size for matrix multiplication is increasing. For 95% percentile, the confidence interval for matrix multiplication program with different input matrix sizes including 100×100, 150×150, 200×200, 250×250, 300×300, 350×350, and 400×400, is observed ±0.006, ±0.062, ±0.016, ±0.138, ±0.189, ±0.576, and ±0.409, respectively. Similarly, for 450×450, 500×500, 550×550, 600×600, 650×650, and 700×700, confidence interval is observed ±0.681, ±0.886, ±1.756, ±1.834, ±1.355, and ±1.658, respectively. Considering very large input matrix sizes such as 700×700, 750×750, 800×800, and 850×850, confidence interval is noticed ±1.658, ±2.973, ±2.807, and ±3.586, re-

University of Malaya

Table 3.2: Energy Consumption Analysis using Power Tutor for Matrix Multiplication Program

Matrix Size	Estimated Energy (j)	Standard Deviation	Confidence Interval
100×100	00.25	0.021	±0.02
150×150	00.47	0.018	±0.02
200×200	00.88	0.009	±0.01
250×250	01.17	0.025	±0.03
300×300	01.71	0.082	±0.10
350×350	02.24	0.109	±0.13
400×400	04.02	0.097	±0.12
450×450	04.97	0.467	±0.58
500×500	07.01	0.783	±0.97
550×550	10.04	1.016	±1.26
600×600	12.93	1.164	±1.44
650×650	16.53	1.153	±1.89
700×700	21.03	1.773	±1.77
750×750	51.83	1.108	±1.37
800×800	66.52	2.489	±3.09
850×850	80.48	1.700	±2.11

spectively. The main reason of comparatively high standard deviation for large input data sets is due to the unpredictable behavior of processes running in the background. Also, for a process that runs for longer period of time, the probability of frequent context switching is high. Energy bugs within smart-phone hardware components and application results in high standard deviation (comparatively).

3.4 Energy Overhead Analysis for Dynamic Analysis Based Estimation Tools

Energy overhead states amount of battery charge an estimation tool consumes during estimating energy consumption of smart-phone application. The energy overhead consists of energy consumed by estimation tool and application being analyzed for estimation. In this section, the overhead of energy estimation tools is reported.

Energy estimation overhead of each energy estimation tool is different depending on the methodology it opts to profile and construct power models for smart-phone components. In the case of Power Tutor, total energy overhead of Power

University of Malaya

Table 3.3: Energy Consumption Analysis for Trepn Profiler for Matrix Multiplication Program

Matrix Size	Estimated Energy (j)	Standard Deviation	Confidence Interval
100×100	000.36	0.00	± 0.006
150×150	000.90	0.05	± 0.062
200×200	000.27	0.01	± 0.016
250×250	002.80	0.11	± 0.138
300×300	004.31	0.15	± 0.189
350×350	005.00	0.46	± 0.576
400×400	009.68	0.32	± 0.409
450×450	012.18	0.54	± 0.681
500×500	017.23	0.71	± 0.886
550×550	024.09	1.41	± 1.756
600×600	031.36	1.47	± 1.834
650×650	040.08	1.09	± 1.355
700×700	050.69	1.33	± 1.658
750×750	064.45	2.39	± 2.973
800×800	090.19	2.26	± 2.807
850×850	101.92	1.88	± 3.586

Tutor is divided into a set of modules as depicted in Eq. 3.14. In Eq. 3.14, the elements that contribute to the total energy overhead of Power Tutor are highlighted. In the aforementioned equation, $E_{ExecutionLOG}$ represents the total energy consumed during logging the system states when android APIs are called during activity within the smart-phone application. This is the most expensive operation as it requires logging execution profile of application throughout its execution life cycle. Alternatively, $E_{Estimation}$ depicts energy cost when Power Tutor uses predefined smart-phone power cost models and captured power states of smart-phone components to estimate total energy consumption of the application.

$$PowerTutor_{OVERHEAD} = E_{ExecutionLOG} + E_{Estimation} \tag{3.14}$$

Trepn Profiler consumes more energy than Power Tutor as it follows on-device power pro-filing and modeling paradigm. Eq. 3.15 presents a conceptual model to state the energy estimation overhead breakdown of Trepn Profiler. In Eq. 3.14,

University of Malaya

$E_{SOCEstimation}$ represents the amount of energy to estimate current and voltage drop during application execution on smart-phone device. The rate of Trepn Profiler is very high as it triggers fuel gauge every 100ms to capture voltage and current drop for energy estimation. In Eq. 3.14, $E_{ExecutionLoG}$ represents the amount of energy required to record the execution profile of the application in terms of CPU state and Wi-Fi mode. Following these two stages, $E_{ModelConstruction}$ represents amount of energy required to generate power models on smart-phone device.

$$TrepnProfiler_{OVERHEAD} = E_{SOCEstimation} + E_{ExecutionLoG} + E_{ModelConstruction} \tag{3.15}$$

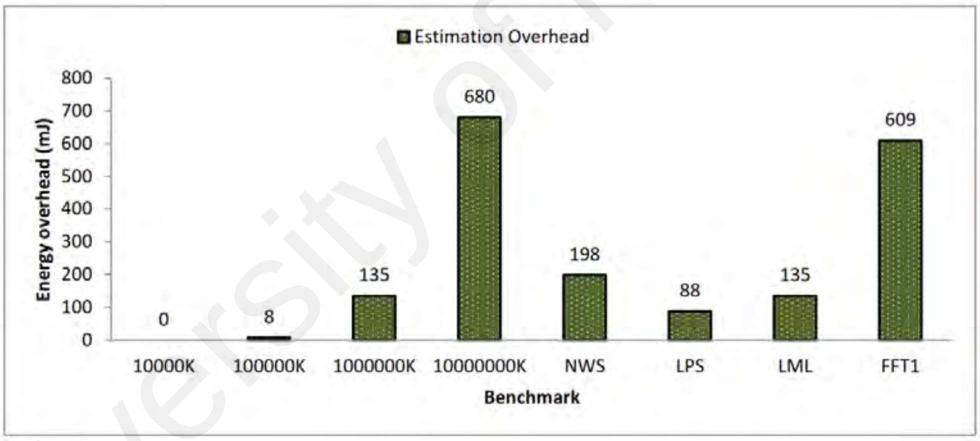

Figure 3.7: Trepn Profiler Energy Estimation Overhead

Fig. 3.7 highlights energy estimation overhead of Trepn Profiler energy estimation tool for a set of benchmark applications. Among all selected benchmark applications, Trepn Profiler has consumed the highest energy (e.g., 680mJ) during energy estimation of the 10000000k test program. The main reason for high energy estimation overhead is the high estimation time of 10000000k test program. Among all chosen benchmark applications, energy estimation time for 10000000k is noticed highest. The energy estimation overhead of an estimation tool is directly related

University of Malaya

to the estimation time of the energy estimation tool. Alternatively, for the 10000K test program, it has consumed negligible energy (≤ 1 mj). The main reason for the negligible energy overhead is the low estimation time of Trepn Profiler while estimating energy consumption of 10000K test program. For the remaining benchmark applications such as LinpackSP2, NativeWhetstone2, LivermoreLoops2, 100000K, 1000000K, and FFT1, Trepn Profiler has consumed 88mJ, 198mJ, 135mJ, 8mJ, 135mJ, and 609mJ energy as shown in Fig. 3.7. The value of energy estimation overhead for each benchmark application demonstrates the average of 15 runs on a smart-phone. For all the selected benchmark programs, standard deviation from the mean for the energy estimation overhead of Trepn Profiler is observed 223.86. The main reason of high value of standard deviation is the high divergence among chosen benchmark applications in terms of their execution time.

Fig. 3.8 presents energy overhead of Power Tutor energy estimation tool during energy estimation for a set of benchmark applications. The energy estimation overhead for all selected benchmark applications is dissimilar depending on their execution time. The energy estimation overhead behavior of Power Tutor is similar to the one presented in Fig. 3.7. Among all selected benchmark applications, power tutor has consumed the highest energy (e.g., 615mJ) during energy estimation of the 10000000k test program. For the remaining benchmark applications such as LinpackSP2, NativeWhetstone2, LivermoreLoop2, 100000K, 1000000K, and FFT1, power tutor has consumed 81mJ, 182mJ, 124mJ, 7mJ, 122mJ, and 555mJ energy as shown in Fig. 3.8. For the 10000K test program, Power Tutor has consumed negligible energy (≤ 1 mj) during energy estimation process. The main reason for this negligible energy overhead is the low estimation time of Power Tutor while estimating energy consumption of 10000K test program. This study has calculated and reported standard deviation from the mean for aforementioned benchmarks. It was

University of Malaya

noticed the standard deviation from the mean is 246.86. The main reason for high value of standard deviation is the non-similarity among the designs/functionality of chosen benchmark applications.

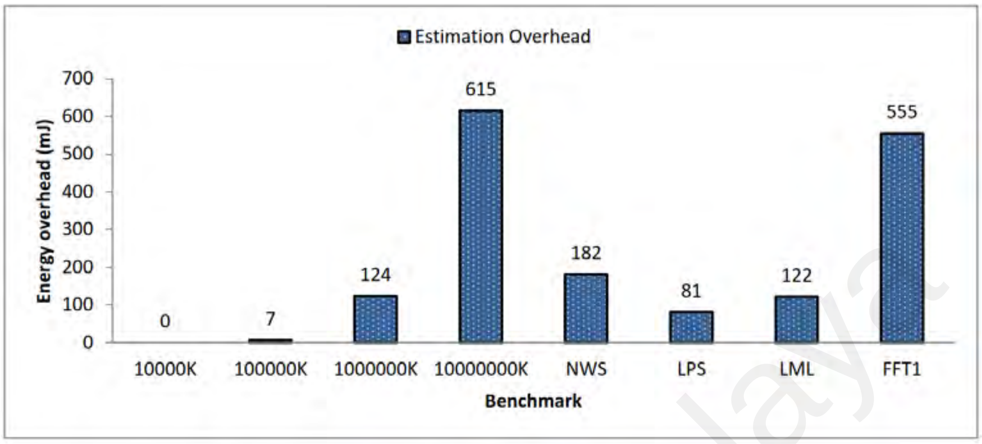

Figure 3.8: Power Tutor Energy Estimation Overhead

Fig. 3.9 compares Power Tutor and Trepn Profiler to highlight the energy estimation overhead difference between their findings for a set of benchmark applications. Fig. 3.9 revealed that Power Tutor is more energy efficient energy estimation tool than Trepn Profiler. The energy estimation overhead of Trepn Profiler is 9-14% higher than Power Tutor. The main reason of high energy estimation overhead of Trepn Profiler is its heavy weight design to construct power models online, more smart-phone components to model, and high valued GUI to represent the findings.

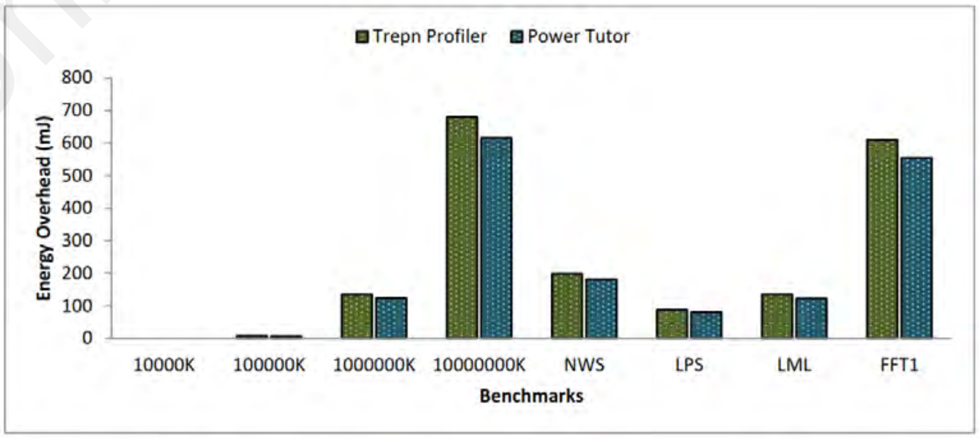

Figure 3.9: Power Tutor vs. Trepn Profiler Energy Estimation Overhead Difference Analysis

University of Malaya

Table 3.4 presents an analysis of energy overhead for Power Tutor energy estimation tool for matrix multiplication program when input data size is linearly increasing. For a specified input size of matrices for multiplication program, total energy estimation overhead is collected for 15 runs of the program. Standard deviation from mean and confidence interval for 95% percentile is calculated. It is noticed that increasing the input data size increases the energy overhead of matrix multiplication program. The energy estimation overhead for matrix multiplication program for matrices of size 100×100, 150×150, 200×200, 250×250, 300×300, 350×350, and 400×400, is observed 0.02mj, 0.6mj, 001mj, 022mj, 031mj, 037mj, and 073mj, respectively. It is noticed that increasing the size of matrices for multiplication increases energy overhead. Energy estimation overhead is directly related to the estimation time of an application. Therefore, increasing data size indirectly increases energy overhead of Power Tutor energy estimation tool. For instance, the estimation time of 100×100 is lower than 200×200; hence, energy overhead of former is lower than latter as highlighted in Table 3.4. For 95% percentile value, the confidence interval for matrix multiplication program with different input matrix sizes including 100×100, 150×150, 200×200, 250×250, 300×300, 350×350, and 400×400, is observed ± 0.00, ± 0.003, ± 0.010, ± 0.74, ± 0.92, ± 1.76, and ± 1.55, respectively. Alternatively, for matrices of size 450×450, 500×500, 550×550, 600×600, 650×650, and 700×700, energy estimation overhead is observed 091mj, 127mj, 181mj, 236mj, 301mj, and 387mj, respectively. For 450×450, 500×500, 550×550, 600×600, 650×650, and 700×700, confidence interval is calculated ± 1.69, ± 2.5, ± 2.12, ± 3.04, ± 3.24, and ± 3.28, respectively. For very large input matrix sizes such as 700×700, 750×750, 800×800, and 850×850, the energy overhead is estimated 387mj, 483mj, 676mj, and 764mj, respectively. For all matrix sizes considered for matrix multiplication, the energy overhead is noticed highest

University of Malaya

Table 3.4: Energy Estimation Overhead Analysis for Power Tutor for Matrix Multiplication Program

Matrix Size	Energy Overhead (mj)	Standard Deviation	Confidence Interval
100×100	0.02	0.001	±0.00
150×150	0.6	0.002	±0.003
200×200	001	0.010	±0.010
250×250	022	0.700	±0.74
300×300	031	0.940	±0.92
350×350	037	1.672	±1.76
400×400	073	1.473	±1.55
450×450	091	1.580	±1.69
500×500	127	2.300	±2.50
550×550	181	2.401	±2.12
600×600	236	2.900	±3.04
650×650	301	2.971	±3.20
700×700	387	3.640	±3.80
750×750	483	2.701	±3.91
800×800	676	4.293	±4.53
850×850	764	4.990	±5.24

for 850×850 because of its high energy estimation time. For 700×700, 750×750, 800×800, and 850×850 matrix sizes, confidence interval is noticed ±3.8, ±3.9, ±4.5, and ±5.2, respectively.

Table 3.5 presents an analysis of energy overhead for Trepn Profiler energy estimation tool for matrix multiplication program when input data size is linearly increasing. For a specified input size of matrices for multiplication program, total energy estimation overhead is collected for 15 runs of the program. Standard deviation from mean and confidence interval for 95% percentile is calculated. It is observed that on increasing the input data size of matrices, energy overhead of matrix multiplication program linearly increases. The energy estimation overhead for matrix multiplication program for matrices of size 100×100, 150×150, 200×200, 250×250, 300×300, 350×350, and 400×400, is observed 0.02mj, 0.73mj, 1.40mj, 022mj, 033mj, 041mj, and 079mj, respectively. It is noticed that increasing the size of matrices for multiplication program increases energy overhead. Energy estima-

University of Malaya

tion overhead is directly related to the estimation time of an application. Therefore, increasing data size indirectly increases energy overhead of Trepn Profiler energy estimation tool. For instance, the estimation time of 100×100 is lower than 200×200; hence, energy overhead of former is lower than latter as highlighted in Table 3.5. For 95% percentile, the confidence interval for matrix multiplication program with different input matrix sizes including 100×100, 150×150, 200×200, 250×250, 300×300, 350×350, and 400×400, is ± 0.00, ± 0.003, ± 0.006, ± 0.63, ± 0.92, ± 1.27, and ± 1.05, respectively. Alternatively, for matrices of size 450×450, 500×500, 550×550, 600×600, 650×650, and 700×700, energy estimation overhead is observed 100mj, 139mj, 197mj, 259mj, 328mj, and 422mj, respectively. For 450×450, 500×500, 550×550, 600×600, 650×650, and 700×700, confidence interval is, ± 1.62, ± 1.76, ± 2.48, ± 3.03, ± 1.99, and ± 3.12, respectively. For very large input matrix sizes such as 700×700, 750×750, 800×800, and 850×850, the energy overhead is estimated 422mj, 532mj, 751mj, and 857mj, respectively. For all matrix sizes considered for matrix multiplication, the energy overhead is noticed highest for 850×850 because of its high energy estimation time. For 700×700, 750×750, 800×800, and 850×850 matrix sizes, confidence interval is noticed ± 3.12, ± 3.35, ± 2.68, and ± 2.07, respectively.

University of Malaya

3.5 Resource Usage Analysis for Dynamic Analysis Based Estimation Tools

Dynamic analysis based energy estimation tools inefficiently exploit resources of smart-phone devices. Dynamic analysis energy estimation tools engage resources of smart-phone for a longer period of time for energy estimation of smart-phone applications as discussed in Section 3.2 and Section 3.3. This section investigates the amount of CPU and RAM capacity that a dynamic analysis based energy estimation tool requires during energy estimation of a smart-phone application. The reported

Table 3.5: Energy Estimation Overhead Analysis for Trepn Profiler Tool for Matrix Multiplication Program

Matrix Size	Energy Overhead (mj)	Standard Deviation	Confidence Interval
100×100	0.02	0.001	±0.00
150×150	0.73	0.004	±0.003
200×200	1.40	0.007	±0.006
250×250	022	0.691	±0.63
300×300	033	0.995	±0.92
350×350	041	1.38	±1.27
400×400	079	1.12	±1.05
450×450	100	1.74	±1.62
500×500	139	1.70	±1.76
550×550	197	2.68	±2.48
600×600	259	2.99	±3.03
650×650	328	2.16	±1.99
700×700	422	3.37	±3.12
750×750	532	3.8	±3.35
800×800	751	4.29	±2.68
850×850	857	4.99	±2.07

results in this section has shown the resource consumption for energy estimation tool only.

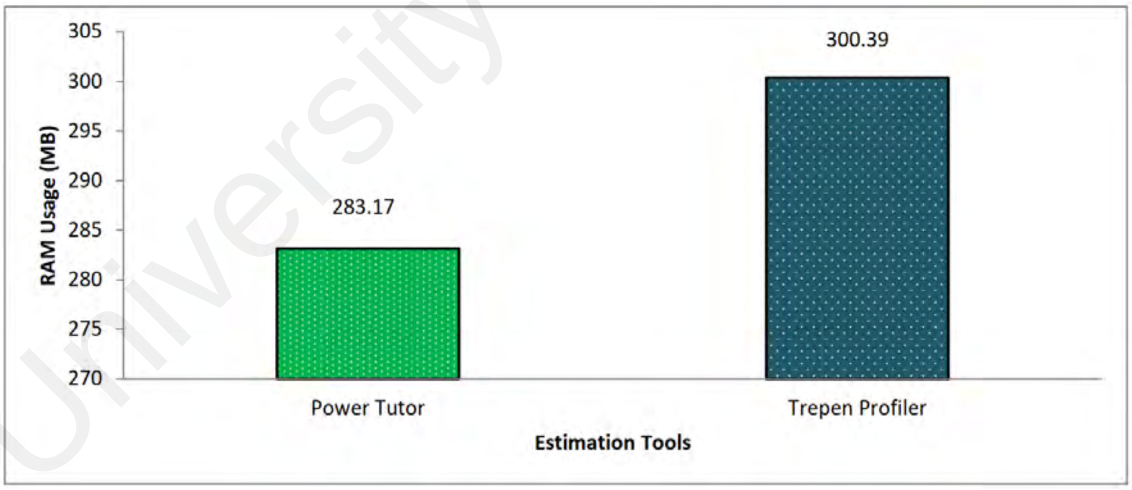

Figure 3.10: RAM Usage Comparison for Power Tutor and Trepn Profiler Energy Estimation Tools

To estimate resource usage of estimation tools, this study has used *Pmap* command of Ubuntu distribution to estimate the amount of RAM the estimation tool needs to load its pages for energy estimation of smart-phone applications. During the experiments, the estimation tool was set to run in the background of smart-phone

University of Malaya

to investigate its resource usage behavior. To investigate CPU usage of estimation tools, *htop* Linux utility is used. The resource consumption rate for Power Tutor and Trepn Profiler energy estimation tools was monitored by running htop and pmap processes from the terminal emulator installed on the smart-phone.

Fig. 3.10 highlights RAM usage of Trepn Profiler and Power Tutor energy estimation tools to estimate energy consumption of a smart-phone applications. The memory and CPU usage are investigated during energy estimation of OS running utilities such as system UI, Kernel, Media Server, system, and radio sub-system, using Power Tutor and Trepn Profiler tools. It is shown in the Fig 3.10 that Power Tutor consumes less memory than Trepn Profiler during energy estimation process. During energy estimation process, Power Tutor has consumed 283.17MB memory, whereas, Trepn Profiler has consumed 300.39MB RAM. In terms of memory, Power Tutor consumes 6% less memory than Trpen Profiler energy estimation tool.

Fig. 3.11 highlights CPU usage comparison for Power Tutor and Trepn Profiler energy estimation tools. It is observed that Power Tutor requires less CPU capacity than Trepn Profiler during energy estimation of a smart-phone application. For background system activities, during estimation process Power Tutor has consumed 2-10% of the total CPU capacity whereas Trepn Profiler has consumed up to 55% CPU capacity. The CPU usage of Trepn Profiler is observed 75% higher than Power Tutor for energy estimation of smart-phone applications. Trepn Profiler offers high functionality than Power Tutor (e.g, energy estimation, performance measurement); therefore, it requires more CPU capacity during the estimation process.

3.6 Analysis of Application Components Energy Consumption and Simulation Based Architecture Level Profiling

Smart-phone application triggers components of smart-phone device to perform specific tasks. Smart-phone application is the main force behind smart-phone battery

University of Malaya

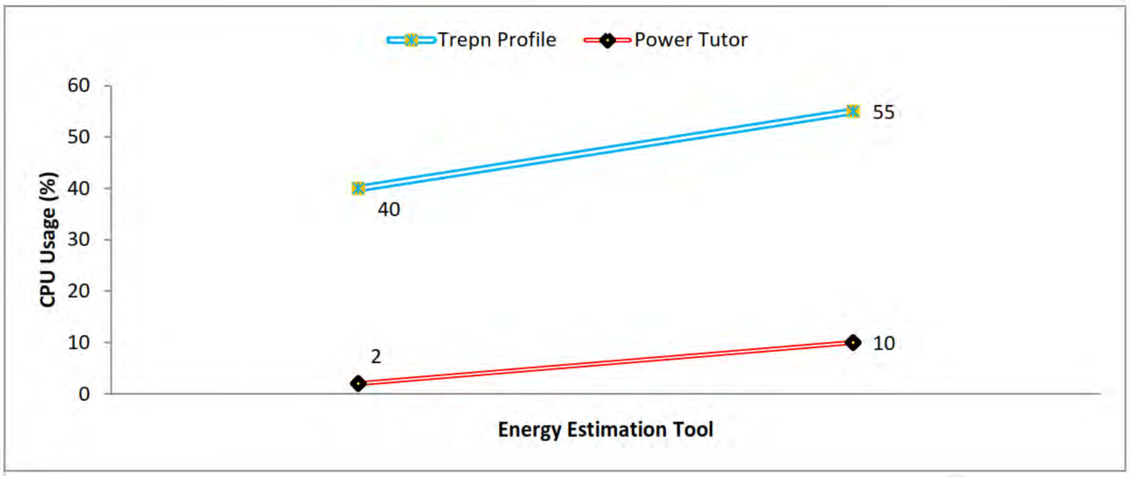

Figure 3.11: CPU Usage Comparison for Power Tutor and Trepn Profiler Estimation Tools

University of Malaya

charge usage. The amount of energy consumed by an application highly depends on the activity time of application. Within a smart-phone device, application constructs such as function calls, loops, arrays, and arithmetic processing, are the utmost energy consuming entities. In this section, the energy consumption of different constructs of a smart-phone application are explored.

3.6.1 Application Components Energy Consumption

This section highlights most energy consuming element within an application in terms of its execution time and energy consumption.

3.6.1.1 Experiments

Fig. 3.12 highlights the experimental setup to estimate execution time and energy consumption of smart-phone application's constructs (e.g., function, loops, paths, etc). It instrumented target smart-phone application to record the execution time and energy consumption across target application construct. In the next phase, it runs the instrumented application on the smart-phone device to log timestamped execution profile on the the smart-phone device. The energy consumption behavior of the smart-phone application is profiled (parallel to execution log at the mobile

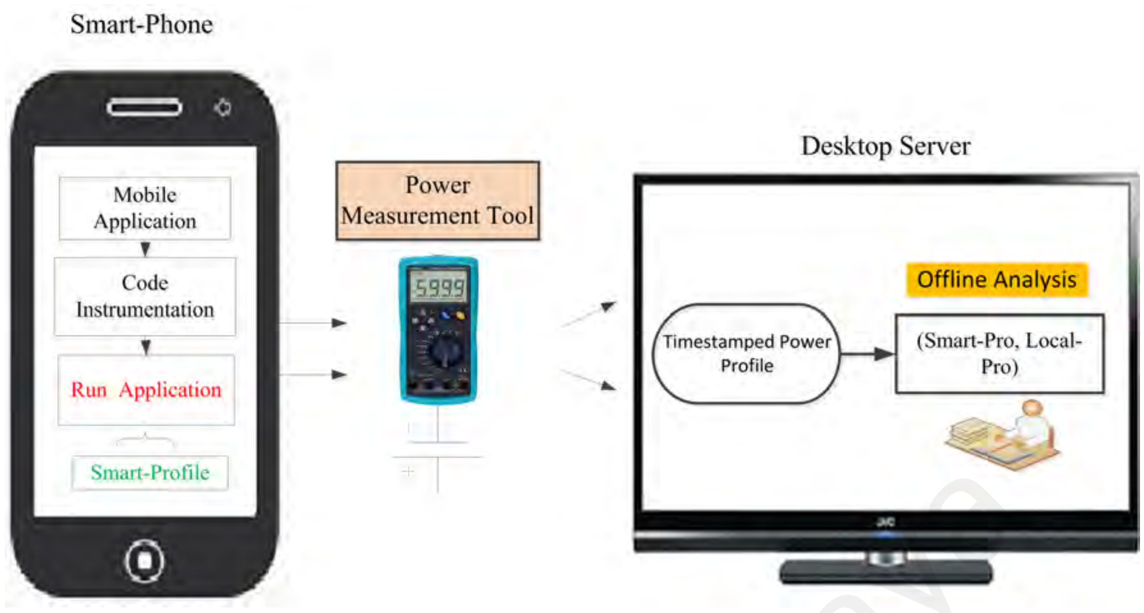

Figure 3.12: Evaluation Framework for Energy Consumption Behavior Analysis of Smart-phone Application Components

side) at the external physical server using EM6000 multi-meter. EM6000 multimeter records timestamped power profile at external physical server. After profiling stage, the recorded timestamped power profile is post processed at the server to examine the energy consumption of smart-phone application constructs. For energy finding, the offline analysis correlates timestamped execution profile generated at smart-phone to the timestamped power profile recorded on the server side. During the experiments, this study has used ClockSync (NTP server) tool to insure that the clocks of the smart-phone and server are strictly synchronized to eliminate any chances of errors. During the offline analysis phase, it is noticed that function calls and loop structures are the major entities where the majority of application energy budget goes. In this section, the energy consumption and execution time analysis are highlighted for few selected benchmark applications. During post-processing on time-stamped power profile of the smart phone application, the proportion of energy consumed by loops was noticed highest. Therefore, energy consumption of loops within the smart-phone application is highlighted.

University of Malaya

3.6.1.2 Experimental Results

Fig. 3.13 demonstrates proportion of energy consumed by loop structures to the rest of the program. A set of benchmark applications for experiments including 10000000K, NativeWhetston2, LinpackSP2, LivermoreLoops2, and FFT1 are selected. It was observed that loop significantly consumes energy and execution time within a smart-phone application. For instance, in NativeWhetstone2, body of loop executes several thousands million of times while performing arithmetic and trigonometric operations. The code within loops of 10000000K, NativeWhetstone2, LinpackSP2, LivermoreLoops2, and FFT1, has attributed 100%, 94.5%, 93%, 96%, and 90% of the total execution time, respectively. During benchmark execution on a smart-phone device, 96% of the CPU time was spent in executing code residing within loops in Livermoreloop2 benchmark to perform arithmetic, trigonometric, and array based operations on integer and float point data.

University of Malaya

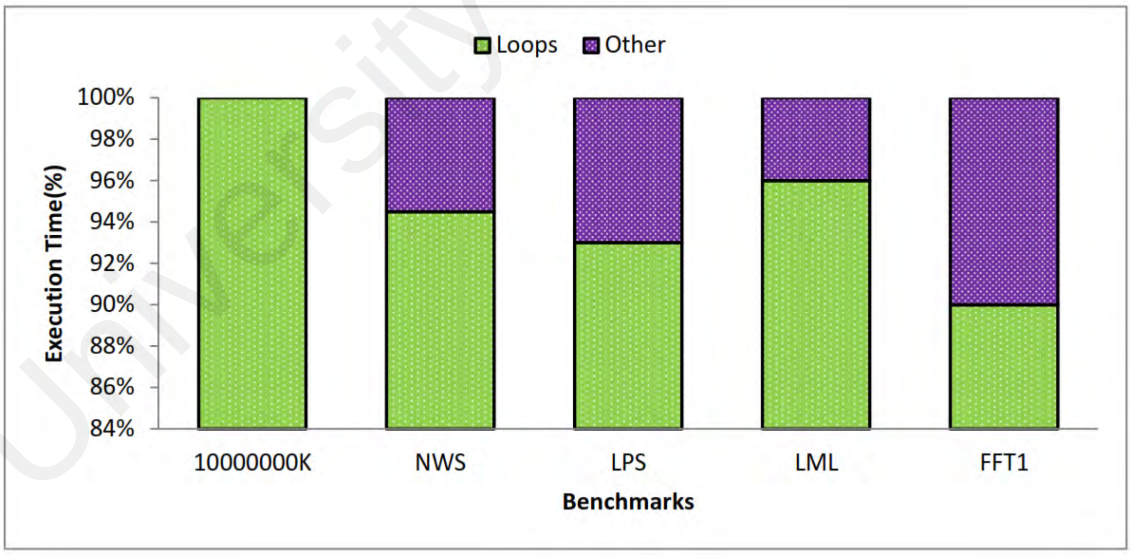

Figure 3.13: Loops Execution Time Analysis for Benchmark Applications

Fig. 3.14 demonstrates energy consumption of loops in comparison to rest of the parts of a program for a set of benchmark applications. The benchmarks selected for experiments include 10000000K, Netwhetstone2, LinpackSP2, LivermoreLoop2, and FFT1. The code within loops of 10000000K, Netwhetstone2, LinpackSP2, LivermoreLoop2, and FFT1, has attributed 100%, 92%, 89.2%, 95%, and 84% of the total energy consumption, respectively.

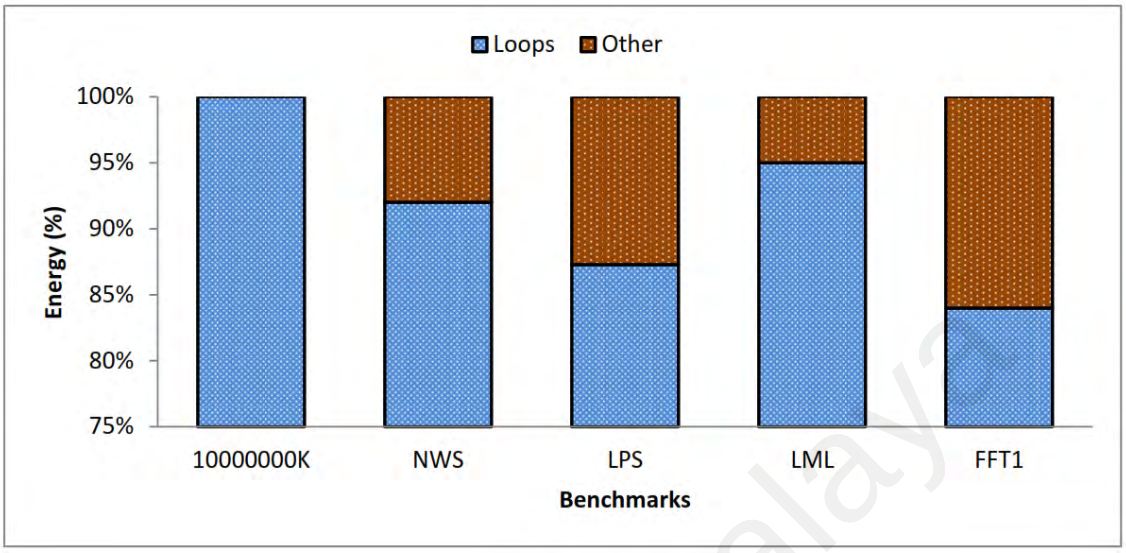

Figure 3.14: Loops Energy Consumption Analysis for Benchmark Applications

From above discussion it is noticed that for a computations intensive benchmark application loops are the most critical part of the program as they consume major portion of the program battery consumption. The execution time and energy of loops highly depends on (a) loops iteration count, (b) size, and (c) type of operations within loop body. Also, memory access pattern too effects energy consumption of smart-phone applications. Higher the cache size, less the execution time program takes (e.g., audio song player). In the next section, the memory access pattern during smart-phone applications execution using simulation based method is analyzed.

3.6.2 Simulation Based Architecture Level Profiling

This section discusses overhead associated with estimating low level architectural details of a smart-phone device using Val-grind tool. It highlights the time analysis while collecting low architectural details for a set of applications.

University of Malaya

3.6.2.1 Experiments

To perform experiments, a set of test programs were designed and tested to monitor the performance of simulation based architecture level profiling for smart-phone applications. Test programs consist of a set of statements performing standard CPU and memory arithmetic operations. Mem-ADD test program comprised of 10^{12} instances of ADD statement written in a series without looping the program to minimize the cache hit rate. C-ADDOP test program comprised several instances of ADD instructions encapsulated within three nested loops to fetch the data from the local cache. Alternatively, WHET test program comprises of mixed standard arithmetic and trigonometric operations where program statements are encapsulated under tightly nested loops. The names of the remaining test programs such as 8MOD, 12MOD, 15MOD are chosen based on their size on hard disk storage. For instance, the 8MOD test program comprised of 8K instructions performing a modulus operation on integer data. Alternatively, 12MOD and 15MOD comprised of 12K and 15K instructions encapsulated in tight loops. The instructions within 12MOD and 15MOD test programs perform modulus operation on the integer data.

This study has used Valgrind-3.11.0 tool to analyze execution behavior of test programs to testify ARM-7's architecture level operations such as cache access pattern and execution time. It has used two tools including Lackey and Cache-grind to simulate the low architectural details. Lackey is used to measure total memory references during program execution on ARM architecture. Alternatively, Cache-grind is used to analyze the cache access pattern for the chosen programs on ARM architecture. Moreover, the performance of simulation based method to the performance of experimentation on real smart-phone is compared. The size of cache for ARM-7 was chosen 16kB. For experiments, cache was configured to 4-way set associativity.

University of Malaya

Table 3.6: Val-grind Simulator based Program Execution Analysis

Test Program	Memory reference	Cache Hits	Cache Miss	Miss/Hit(%)	Program size
Mem-ADD	152,010,363	118,653,082	33357281	22/78	608MB
C-ADDOP	201,765,406,976	201,765,406,563	413	0.00/99.99	593kB
WHET	137,283,478,361	137,281,686,679	179,168,2	0.001/99.99	810kB
8MOD	3,955,381,412	3,911,142,941	44,238,471	1.11/98.8	619kB
12MOD	407,404,285,4	4,028,477,056	45565798	1.12/98.7	641kB
15MOD	423,700,456,9	4,189,615,219	47389350	1.11/98.8	651kB

3.6.2.2 Experimental Results

Table 3.6 highlights cache access pattern for seven test programs. It is observed that Mem-ADD test program has highlighted the largest cache miss rate among all the selected test programs. The main reason of this behavior is the sequential organization of statements within the program to perform a specific task.

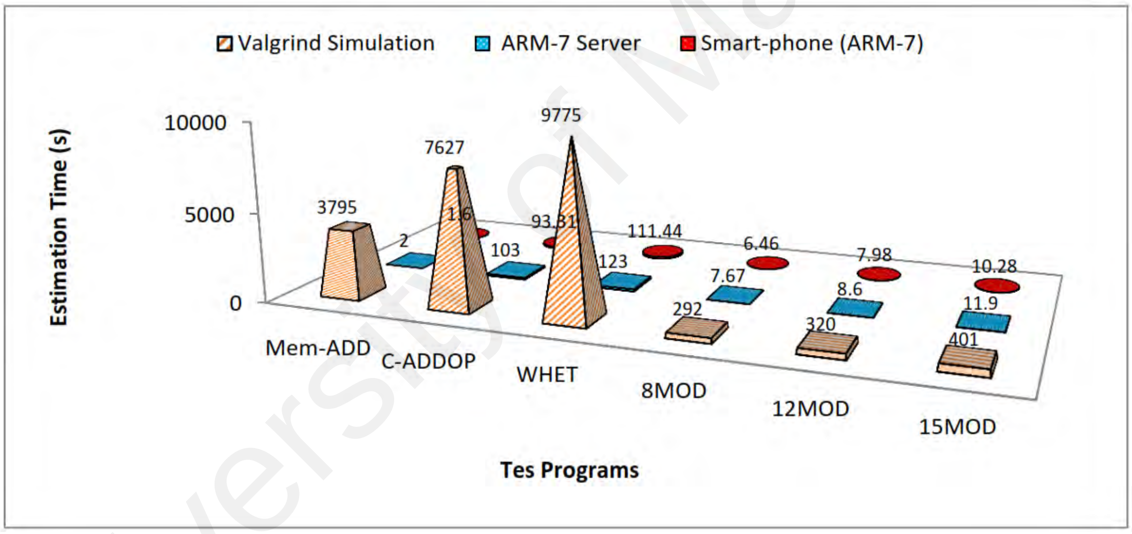

Figure 3.15: Estimation Time Analysis

Fig. 3.15 compares the performance of three test execution modes including Val-grind simulation, ARM-7 server, and real smart-phone device. It can be seen that application execution on the server (emulation) is 10-19% times slower than operations on real smart-phones. Moreover, as shown in Fig. 3.16, analyzing an application based on Val-grind increases the application execution time by 97-99.8%. This overhead makes the emulation based estimation impractical as estimation time is very large.

University of Malaya

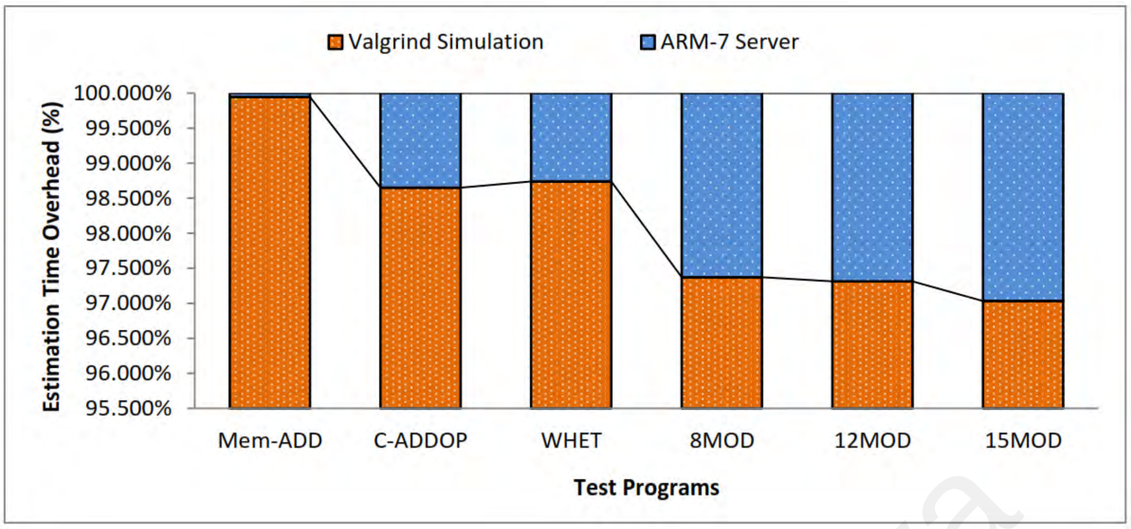

Figure 3.16: Val-grind Simulator based Application Profiling Time Overhead Analysis

3.7 Conclusion

In this chapter, the performance of dynamic analysis based energy estimation tools is analyzed. It performed experimentation to analyze estimation time, energy overhead, energy consumption, and resource consumption of existing dynamic analysis based energy estimation tools.

A higher degree of complexity within an application prolongs its execution time and significantly impacts to the battery lifetime of a smart-phone device. It was observed that total energy estimation time of an estimation tool highly depends on the execution time of smart-phone application on smart-phone device. Energy consumption of a smart-phone application highly depends on its execution time on a smart-phone device. It was observed that Trepn Profiler estimated energy is 13% higher than the Power Tutor estimation tool. Dynamic analysis based energy estimation is costly as it consumes a significant amount of energy to execute an application to estimate its energy consumption. The energy overhead of Trepn Profiler is observed 9-14% higher than Power Tutor energy estimation tool. Moreover, Trepn Profiler has consumed 6% more storage resources during energy estimation of a smart-phone application. Trepn Profiler's CPU resource usage is 75% more than

University of Malaya

Power Tutor energy estimation tool. It is also noticed that loops are utmost time and energy consuming entities within a smart-phone application as it uses up to 95% CPU and battery resources of the application. Considering all these issues, it is important to propose an estimation method that should minimize energy estimation time and energy overhead of existing energy estimation tools.

University of Malaya

CHAPTER 4: SA-LEEF: STATIC ANALYSIS BASED LIGHTWEIGHT ENERGY ESTIMATION FRAMEWORK

Smart-phones are resource constrained devices and carry limited battery power. Due to the emergence of rich mobile applications, efficient battery usage has become a must to meet requirement for recent smart-phone devices. Dynamic analysis based energy estimation tools considerably use system resources to estimate energy consumption of smart-phone applications. This chapter discusses the methods and procedures to solve the problem of traditional dynamic analysis estimation methods identified in chapter 3.

The contents of this chapter are divided into six sections. Section 4.1 briefly discusses the design of proposed static analysis based lightweight energy estimation framework for smart-phone applications. Section 4.2 discusses assumptions and constraints. Section 4.3 presents a flow diagram and system model of the proposed SA-LEEF framework. Section 4.4 discusses distinguishing features of SA-LEEF. Section 4.5 presents the data design to evaluate proposed framework. Lastly, Section 4.6 concludes this chapter and highlights the main findings.

4.1 Proposed Static Analysis Based Lightweight Energy Estimation Framework

This section proposes and presents a 2-tier lightweight energy estimation framework for smart phone applications[1]. The proposed framework called SA-LEEF solves the issues of dynamic analysis based energy estimation schemes by employing static analysis methodology. Static analysis method does not run the application on smart-phone to estimate energy consumption of the smart-phone application. SA-LEEF solves the issues of non-deterministic nature of smart-phone applications

[1]Ahmad, Raja Wasim, Abdullah Gani, Siti Hafizah Ab Hamid, Anjum Naveed, Kwangman KO, and Joel JPC Rodrigues. "A case and framework for code analysis–based smartphone application energy estimation." International Journal of Communication Systems (2016)

University of Malaya

based on weighted probability based execution path estimation and static cache analysis method.

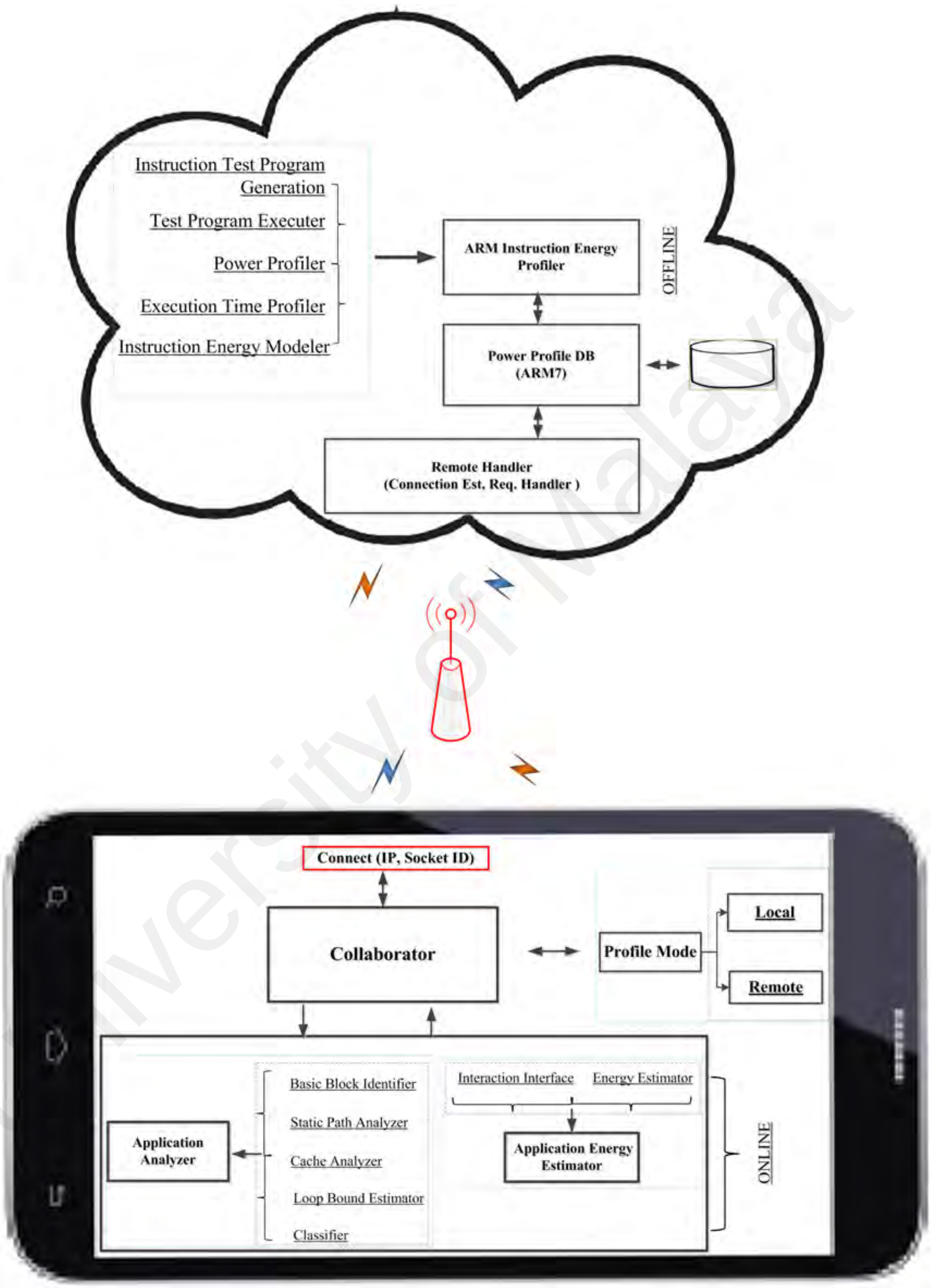

Figure 4.1: SA-LEEF: Smart-phone Application Energy Estimation Framework

Fig. 4.1 demonstrates the proposed 2-tier lightweight estimation framework to estimate application energy consumption for smart-phones. In comparison to exist-

University of Malaya

ing energy estimation methods, it employs static analysis methodology to not run the application to estimate its energy consumption. Tier-1 of the proposed framework is hosted on nearby cloud servers. Alternatively, Tier-2 of the proposed framework hosts locally on the smart-phone device. SA-LEEF is lightweight by design as, (a) it does not run the application on smart-phone, (b) it schedules resource expensive tasks on resource-rich cloud servers, and (c) it employs probabilistic analysis of code to forecast execution paths of application. Proposed framework is generic and is applicable to all mobile architectures whose ARM-IS energy profile is available on cloud servers.

Proposed estimation framework comprises a set of modules which are, Collaborator, Application Energy Estimator, ARM instruction Energy Profiler, Application Analyzer, Power Profile DB, Remote Handler, and Profile Mode, as depicted in Fig. 4.1. Each module of the framework has some unique features associated with it to estimate application energy consumption. Among all modules in SA-LEEF, a few hosts at cloud side (i.e., ARM Instruction Energy Profiler, Power Profile DB, and Remote Handler) whereas remaining modules reside on smart-phone (i.e., Application Analyzer, Collaborator, and Application Energy Estimator). On the cloud side, all modules except Remote Handler runs in an offline execution mode. On the other hand, all modules of smart-phone side execute in an online mode to estimate application energy consumption. Among all modules in SA-LEEF, ARM instruction energy Profiler is resource expensive as it profiles energy consumption of ARM-IS for a particular ARM architecture using a number of test programs.

4.1.1 ARM Instruction Energy Profiler

The main responsibility of ARM instruction energy profiler module is to estimate and log energy consumption of each instruction within target ARM-IS at Energy

University of Malaya

Profile DB module. In contrast to profiling energy consumption of high level source code operations, ARM Instruction Energy Profiler estimates energy consumption of low level ARM assembly instructions. Also, during energy estimation of ARM assembly instructions, it applies neighborhood operation (weighted filter) on test program's power profile to improve the estimation accuracy.

ARM Instruction Energy Profiler is a cloud-based offline computational module. Basic operations in ARM instruction energy profiler includes, (a) instruction test program generation, (b) test program execution for power profiling, (c) execution time logging, and (d) energy estimation for individual ARM instructions.

Algorithm 1 details the order of execution of different operations in ARM instruction energy profiler module. In the first step, the proposed module runs the test programs on the smart-phone to record the power profile (using multi-meter) and execution time for each test program. In the next step, it applies neighborhood operation on time-stamped power profile ($Neighborhood\_Operation()$) to suppress the effect of noise due to running of background OS activities such as threads context switching and garbage collection, to the total estimation accuracy. In the neighborhood operation, it analysis the power profile to identify the abnormal power peaks within power profile of instructions test programs. In the subsequent stages, it replaces abnormal power peaks within test programs power profile with the weight of power consumption readings in its neighborhood. The weight of a power reading (abnormal power peak) is estimated by finding the average of 8 power values in its neighborhood. For Pt_{10} (an abnormal power peak at time "t") in a test program, neighborhood operation employs "$Pt_{10} \leftarrow \frac{\sum_{t=6}^{14} P_t}{8}$" to estimate power consumption for it (Appendix B). The baseline system power (captured at line 2) is eliminated from the power profile of test program to isolate test program's power profile from the idle system power to ensure accurate instruction energy consumption estimation

University of Malaya

(see line 9-11). The average energy consumption for a single instruction is estimated based on energy consumed during test program execution, test program execution time, and test program size as highlighted in Algorithm 1(steps 9-11). In Algorithm 1, $TestProgramSize$ is an integer constant that represents count of statements in the test program.

Algorithm 1 ARM Instruction Energy Profiler

```
1:  Input: $N$ number of ARM instructions test programs
2:  $Base\_P \leftarrow Baseline\_Power()$ ▷Power Consumption with no-activity on phone
3:  for $I = 0$ to $N$ do
4:      $< Pow\_Prof[I], Exe\_Time_I > \leftarrow Run\_Test\_Program(I)$ ▷Capture Power Profile and Execution time
5:  end for
6:  for $I = 0$ to $N$ do
7:      $< Pow\_Prof[I] > \leftarrow Neighborhood\_Operation(Pow\_Prof[I], Weight\_Filter)$
8:  end for
9:  for $I = 0$ to $N$ do
10:     $Avg\_Pow[I] \leftarrow \frac{\sum_{n=1}^{M} Pow\_Prof[J]}{\#ofPowerReadings} - Base\_P$ ▷Eliminate Idle Power for Test Programs
11: end for
12: for $I = 0$ to $N$ do
13:     $ARM\_ISA\_ENER[I] \leftarrow \frac{Avg\_Pow[I] \times Exe\_Time_I}{TestProgramSize}$ ▷Per-instruction Energy Cost
14: end for
15: Return ARM_ISA_ENER
```

4.1.2 Application Analyzer

Application analyzer module scans assembly based source code of application to estimate the execution flows, storage locations, and loop bounds. In contrast to application instrumentation based execution flow analysis, Application analyzer module uses weighted probability based function to estimate execution paths in an application. Moreover, it improves the estimation accuracy of existing code analysis based estimation methods by adding cache analysis on assembly source code of the application. The core responsibility of Application Analyzer module is, (a) finding basic building blocks of application, (b) execution paths and loops bound estimation for

University of Malaya

each building block, (c) instruction storage location analysis, and (d) classifying instructions within application into a set of categories.

Algorithm 2 Application Analyzer

```
Input: Assembly Based Mobile application (App)
 1: $B\{B_1, B_2, B_3....B_n\} \leftarrow Identify\_Application\_Building\_Blocks(App)$
 2: for $I = 0$ to $N$ do
 3:     $\{\varnothing, \partial, \hbar\} \leftarrow LoopsBound\_Estimation(Block_I)$ ▷$\varnothing = baseline$,$\partial = growthrate$, and $\hbar = endline$
 4:     $Loops\_Count(Block_I, \varnothing, \partial, \hbar)$
 5:     $ExecutionPath\_Estimation(Block_I)$ ▷$Equation($ 4.2)
 6:     $InstructionStorageLocation\_Analysis(Block_I)$
 7: end for
 8: for $I = 0$ to $N$ do
 9:     $CD = MaximumCache\_Distance(Block_I, Block_I)$
10:     $IteratCount_{Block_I} \leftarrow Count\_Iteration()$ ▷Blocks iteration counting
11:     $InstructionCAnal_{Block_I} \leftarrow CacheAnalysis(IteratCount_{Block_I}, CD)$
12: end for
13: for $I = 0$ to $N$ do
14:     $Block_{I_{IRAM,ICACH}} \leftarrow Classfier(Block_I, InstructionCAnal_{Block_I}, IteratCount_{Block_I})$
15:     Return Block$_{\{I \rightarrow N\}_{IRAM,ICACHE}}$▷Instructions classified within each block
16: end for
```

4.1.2.1 Application Construct Analyzer

The main responsibility of this module is to analyze assembly source code of the application to predict its run-time execution behavior. Application Construct Analyzer scans application assembly source code to find and analyze basic building blocks of an application. During the analysis process, it estimates execution paths and loops bounds within building blocks to empower Instruction Storage Location Analyzer and classifier to classify the instructions based on their storage location analysis.

For each basic building blocks, App construct analyzer estimates loops bound based on slicing method. It extracts the loops from the assembly based code to generate loop slices that contain information necessary to estimate loop bounds. Slicing method estimate parameters including loop baseline, growth rate, and loop

University of Malaya

final criteria for all loops within each building block (Ermedahl, Sandberg, Gustafsson, Bygde, & Lisper, 2007). In next stage, it estimates bound if required variable's values are already known. Alternatively, if parameters are not pre-defined, it exploits data set for the variables within the application to choose the highest value for worst case bound estimation. For instance, consider a statement $for(i=0; i<=UNKNOWN; i++)$; let's assume data set for UNKNOWN variable is $\{1,6,12,15\}$. In this case, App construct analyzer will estimate that the body of loop will execute for 16 times.

Within a smart-phone application, total energy cost highly depends on the execution paths it follow during application execution on smart-phone. App construct analyzer proposes a weighted probability function to statically estimate execution paths within the application. It analysis parameters within branch headers to estimate the probability of execution of code following the branch header. Based on the probability, it assigns weight to the code followed by the branch statement. It calculates weight based on Eq. 4.1. In Eq. 4.1, $\otimes$ describes the bitwise operation among variables X_I and M. $pr[X_I \otimes M = 1]$ represents the cases when the probability is $TRUE$. Eq. 4.2 demonstrates the proposed model to find total energy cost of a path within smart-phone application.

$$Weight = \sum_{I \in SET_I} pr[X_I \otimes M = 1] \tag{4.1}$$

$$E_{Exe\_path} = E_{Nstatments_{I_{block}}} \times (\frac{weight}{M}) + E_{Nstatments_{J_{block}}} \times 1 - (\frac{weight}{M}) \tag{4.2}$$

University of Malaya

4.1.2.2 Instruction Storage Location Analyzer

The instruction storage location analyzer module is responsible for estimating storage location of instructions within target smart-phone application. At run-time, CPU fetches instructions either from cache or memory for execution. Loops and function calls are the two major entities that execute same code over and over depending on loop bounds and number of function calls, respectively. For the loops structure, Instruction storage location analyzer estimates storage location of instructions based on the reference iteration number of the loop. For the first iteration of the loop, it always estimates RAM as the target storage location as instructions are copied from RAM to cache. In case of a cache hit, the instructions and data of the application is fetched from the local cache, whereas, for cache miss case it is fetched from the RAM storage. Therefore, for the first iteration of the loop, Storage location analyzer module result in cache miss for the instructions within the loop body. During remaining loop iterations, for 20% of the instructions within loop body it estimates cache miss whereas for remaining 80% it predicts cache hit (Stallings, 2000; Guan et al., 2013; Grund, 2012). Except for first loop iteration, the probability of data access from the cache is high as loop executes the same code over and over. 20% of instructions within loop body are usually LOAD/STORE; therefore, Storage Location Analyzer considers 20% cache miss penalty. The execution and energy cost while accessing instructions from local cache and RAM is different due to the high difference in their access latency as presented in Eq.4.3. Accessing instructions or data from the local cache consumes less energy compared to RAM access as cache offers limited access latency.

$$Cache_{Latency} < RAM_{Latency} < SD-CARD_{Latency} \tag{4.3}$$

University of Malaya

Functions of an application eliminate the need to write the same code again and again. Calling a function executes same code segment, again and again, depending on the number of calls to it. The information about the distance between two calls to a function helps to statically predict storage location of instructions within the body of a function. Frequent calls (minimum distance between calls) to the same segment of code increases the probability to incur high cache hit rate during run time execution. For a function call, storage location analyzer estimates storage location for the body of function based on cache distance, the size of the cache, and cache lines. During analysis, Instruction Storage Location Analyzer calculates cache distance between function calls by counting instructions that are executed between two calls to the same chunk of code. In the case of ARM architecture, instructions are either 32 bit or 16bit long (thumb2 instructions). In the next step, it compares cache distance to the number of cache lines (already known) to predict the execution location of the instructions. It estimates a cache hit if cache distance of function calls is less than the number of cache lines. Otherwise, it predicts cache miss for the function calls. Eq. 4.4 demonstrates the cache distance between two function calls accessing the same chunk of code. During static code analysis, it marks the function calls within application's code and sums the instructions till the next function call to the same piece of code ($Distance(n_i,n_j)$). In $Distance(n_i,n_j)$, n_i and n_j are the function calls from i_{th} and j_{th} locations from within application source code. Alternatively, $max\{\alpha(p)\}$ represents the maximum distance between subsequent calls; whereas, $P(n_i,n_j)$ represents sets of paths that an application can take to reach next function call. Eq. 4.5 decides storage location of instructions based on the cache distance of the code. Instruction storage location analyzer module also considers the case when size of the body of a function is higher than cache line size.

University of Malaya

In this scenario, it recursively adopts Eq. 4.5 to estimate storage location.

$$Distance(n_i,n_j) = \begin{cases} max\{\alpha(p) \mid p \in P(n_i,n_j)\} & IF n > 0 \\ \\ 0 & otherwise \end{cases} \tag{4.4}$$

$$Decision\_Variable = \begin{cases} HIT & IF distance(n_i,n_j) < K \\ \\ MISS & otherwise \end{cases} \tag{4.5}$$

4.1.2.3 Classifier

The main responsibility of this module is to lexically analyze the instructions within each basic building block of application to classify them into several classes. Classifier considers the output of storage location analyzer module to classify instructions into two main categories including RAM and system Cache. RAM category includes set of instructions within the application that are fetched from the RAM memory. Alternatively, cache category includes instructions which are fetched from the local system cache. Inside each category, it further classifies instructions into arithmetic, logical, LOAD/STORE, MOVE, Built-in Libraries, and JUMP categories. The arithmetic class includes a set of ARM instructions that performs arithmetic operations such as ADD, SUB, MUL, RSB (C, n.d.). The logical class includes a set of instructions responsible for performing logical operations on contents of registers such as ORR, NOR, and AND instructions. LOAD/STORE class includes memory operation based instructions such as LOAD and STORE instructions. MOVE class includes all the instructions to perform data movement operations to CPU registers. Built-in libraries class contains set of libraries to perform required operation such as scanf and printf. JUMP class contains set of assembly instructions that shift control to a different segment of code such as bl, ble, and blg (Hohl & Hinds, 2016;

University of Malaya

C, n.d.). Classifier component analyze the instructions within each building block of application to categorize instructions into aforementioned classes. The output of this module is the instruction count for each instruction within each instructions class for both RAM and Cache categories as presented by $Block_{I_{IRAM,ICACH}}$ variable in algorithm. 2. The estimator module uses classified instructions to estimate energy consumption of the application.

4.1.3 Collaborator

Collaborator works as an interface between cloud hosted modules to the one hosted on the smart-phone device. It is responsible for handling connection management tasks at the local smart-phone device. In the first step, it establishes the network connection to a remote server based on IP address and port ID of receiver server. After the connection is successfully established to the remote cloud, it queries to the remote cloud for ARM-IS energy profile for the target smart-phone device. Based on the response from the cloud it dynamically triggers appropriate profiling mode for energy estimation for smart-phone applications. Profile mode is either local or remote as shown in Fig. 4.1. It triggers remote profile mode if the ARM-IS energy profile for the target smart-phone is already available on a cloud server. Alternatively, if for the target smart-phone device ARM-IS energy profile is not available, it triggers local profile mode. In local profile mode, it downloads ARM-IS test programs from the server to generate ARM-IS energy profile locally based on the Power Tutor energy estimation module. In local profile mode, it runs test programs locally to record time-stamped energy profile for ARM-IS energy profile. Once a smart-phone generates ARM-IS energy profile, it optionally stores it back on a cloud server to make it available for the future queries.

University of Malaya

4.1.4 Application Energy Estimator

Application energy estimator module statically estimates energy consumption of a smart-phone application based on the inputs from the Application Analyzer, Collaborator, and ARM instruction energy profiler modules. In contrast to existing code analysis based methods, Application Energy Estimator module considers energy consumption overhead while running concurrent programs on the smart-phone, user-system interaction, and ARM-IS energy profile downloading cost in addition to application base cost energy.

Application energy estimator module triggers interaction interface (See Fig. 4.1) to communicate with collaborator module to access the ARM-IS energy profile for the required target smart-phone architecture. Estimator module runs the test program locally to generate timestamped power profile based on android APIs if collaborator triggers local profile mode. In the next step, it considers the output of application analyzer module to estimate energy consumption of the smart-phone application. If collaborator module triggers remote profile mode, estimator considers offline generated ARM-IS energy profile to estimate energy consumption of the smart-phone application. Eq. 4.6 demonstrates energy estimation for a set of modules within the application. For each block, Application energy estimator sums the energy consumed by RAM and cache storage location based instructions. In Eq. 4.6, $Count_{B_{I_{RAM}}}$ function represents number of times instruction "I" (e.g., ADD and SUB) in block "B" has been accessed from the RAM memory. Whereas, $count_{B_{I_{CACHE}}}$ represents a count of instruction "I" (e.g., ADD and SUB) in block "B" fetched from the cache. Also, $Cost_{I_{CACHE}}$ and $Count_{B_{I_{RAM}}}$ variables represent energy consumed while executing instruction "I" on smart-phone when fetched from

University of Malaya

system cache and RAM, respectively.

$$E_{BaseCost} = \sum_{B=1}^{N}\sum_{I=1}^{M}((Count_{B_{I_{RAM}}} \times Cost_{I_{RAM}}) + (count_{B_{I_{CACHE}}} \times Cost_{I_{CACHE}})) \tag{4.6}$$

Eq. 4.7 represents total energy consumption of an application estimated by the SA-LEEF framework. In Eq. 4.7, $E_{Eviction}$ highlights the amount of energy consumed due to cache eviction during concurrent program execution on smart-phone device. Alternatively, E_{Inter} highlights the amount of energy consumed when the smart-phone is waiting for the inputs from the user in interactive applications. Finally, $E_{Download}$ demonstrates the total energy consumed while accessing ARM assembly based source code of application and ARM-IS energy profile from the remote cloud server.

$$Energy_{Total} = E_{BaseCost} + E_{Eviction} + E_{Inter} + E_{Download} \tag{4.7}$$

4.1.5 Remote Handler and ARM Profile DB

Remote handler is responsible for listening and establishing the connection to the requests from the smart-phone device. After successful connection establishment, it queries Power Profile DB to search for the ARM-IS energy profile for the target smart-phone model. It forwards the ARM-IS energy profile to the collaborator module if it is available in Power Profile DB. Otherwise, it accesses test programs to copy it to the collaborator module. ARM Profile DB module logs ARM-IS energy profile for a set of smart-phone architectures.

4.2 Assumptions and Constraints

- The current research has proposed a model that is useful for energy estimation of disk I/O based smart-phone applications (e.g., the audio song playing). The interactive smart-phone applications require user input at different intervals

University of Malaya

when the application is executed on smart-phones. The variance in delays by the users, while data is input to the application, is not considered in this system.

- The current research has been carried to estimate energy consumption of native smart-phone application (written in objective C). Native smart-phone applications are high-performance applications and can access the hardware components of smart-phone.

- Application developers set the possible values of variables in the application for ease of analysis. In proposed research, it is assumed that the possible value set for the majority of the variables in the application is already available. The proposed model considers these values to decide whether a branch is taken or not taken.

- The energy estimation framework predicts storage location of data and instructions based on the size of the cache. It is assumed that size of the cache and number of cache lines are already known to the estimation framework for the storage estimation of instructions.

- The proposed framework is valid for the applications for which accessing obj dump is possible. It estimates energy consumption based on ARM-IS energy profile. Therefore, it is a must for the target application to be in assembly language. Linux offers obj dump utility to extract the obj dump of the target application.

- To use the proposed SA-LEEF system, a smart-phone user has to access ARM-IS energy profile from the remote server. Therefore, a stable network con-

University of Malaya

nection is a mandatory requirement to use the proposed energy estimation framework.

- It is assumed that only one user application other than the SA-LEEF is running during energy estimation of smart-phone application.

4.3 System Overview

This section discusses the flow diagram and system model of the proposed SA-LEEF framework.

4.3.1 Flow Diagram of SA-LEEF Framework

Fig. 4.2 demonstrates the execution flow of the activities within SA-LEEF framework. It presents activities that run on the smart-phone side in online mode to estimate energy consumption of applications. Before smart-phone application energy estimation, SA-LEEF establishes the network connection to the remote cloud to access the ARM assembly code of the target smart-phone application. After accessing the assembly source code, it triggers application analyzer to, (a) find the basic building blocks in source code of application, (b) estimate loop bounds, (c) analyze storage location of instructions, and (d) classify the instructions within each building block of application. After application analysis, it calls collaborator module to access the ARM-IS energy profile from the cloud server. Collaborator triggers remote profile mode if ARM-IS energy profile for the target mobile is already available on cloud servers. Otherwise, it triggers local profile mode to locally generate ARM-IS energy profile based on android power APIs. After this phase, estimator module accesses ARM-IS energy profile and classified blocks of the application to estimate energy consumption of the smart-phone application. If the local profiling mode is selected, it (optionally) store back the ARM-IS energy profile on the cloud

University of Malaya

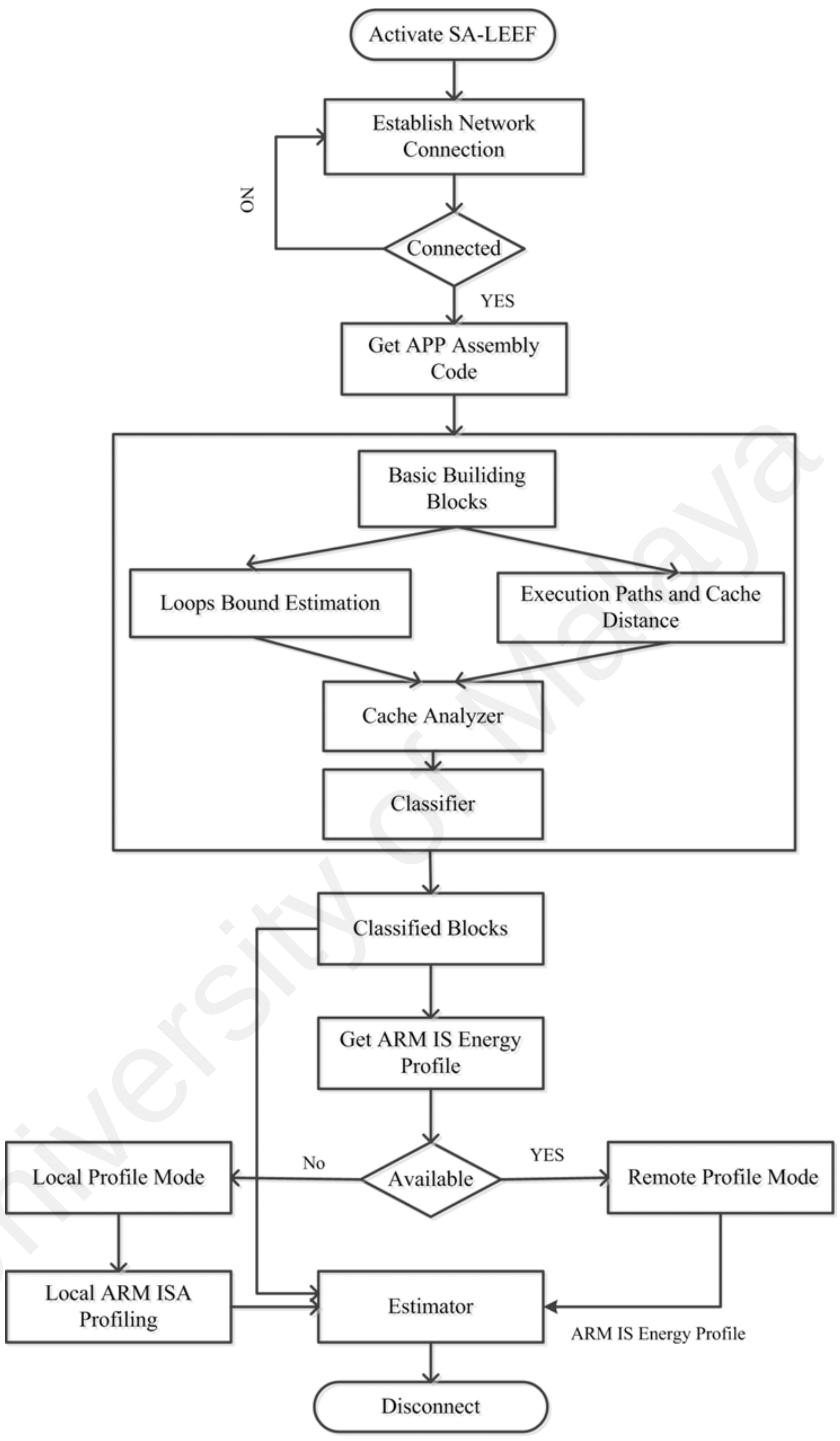

Figure 4.2: Flow Diagram of SA-LEEF Framework

for future accesses.

University of Malaya

4.3.2 SA-LEEF System Model

Eq. 4.8 presents system model of the proposed framework for energy estimation of smart-phone applications. In comparison to existing code analysis based energy estimation models, the proposed SA-LEEF system model has considered static analysis of the code to overcome the overheads of existing models. Existing code analysis based energy estimation models overlooked the cache analysis on the source code of the application while estimating base cost energy of an application. Base cost energy of a program states the amount of energy that the instructions of an application consumes during execution on smart-phone. However, SA-LEEF system model has improved the base cost energy estimation by adding cache analysis on the source code to present more real execution scenario. Also, the proposed SA-LEEF system model has considered weighted probability based execution flow estimation instead of application instrumentation method to estimate execution flows for minimizing energy estimation time (base cost energy estimation module). Moreover, as compared to existing models, SA-LEEF system model has considered the associated overheads due to frequent context switching of applications (cache eviction). During application execution, user interact with application to provide the required inputs. The proposed SA-LEEF framework has modeled the user-system interaction module to simulate run time execution environment.

In Eq. 4.8, total energy consumption of a smart-phone application is divided into (a) program base cost energy, (b) system idle period energy consumption, (c) concurrent program execution energy overhead, and (d) network ARM-ISA access cost. Program base cost energy ($E_{BaseCostEnergy}$) states energy consumed by source code instructions of the smart-phone application. Alternatively, ($N_{Idletime} \times E_{IdleState}$) demonstrates energy consumed by smart-phone device when target smart-

University of Malaya

phone application waits for the input event from the user. Normally, a smart-phone runs more than one application at a time. Co-hosting applications share the resources of the smart-phone device. In Eq. 4.8, $E_{Con-Exe-Over}$ represents the energy overhead occurred due to co-running smart-phone applications because of application context switching.

$$Total_{Energy} = E_{BaseCostEnergy} + (N_{Idletime} \times E_{IdleState}) + E_{Con-Exe-Over} + E_{ISA-Access} \tag{4.8}$$

In the proposed model, $N_{Idletime}$ represents count of the number of times application waits for the input events from the smart-phone user. For the interactive smart-phone applications such as mobile-games, the value of $N_{Idletime}$ is very high as application needs user inputs for decision making. However, Sdcard targeted I/O based applications (e.g., audio song player) does not depend on user's inputs to perform desired functionality. Therefore, for Sdcard I/O based applications, value of $N_{Idletime}$ and $E_{IdleState}$ parameters is negligible. $E_{Con-Exe-Over}$ demonstrates amount of energy consumed by SA-LEEF when executed in parallel with other applications on the smart-phone device. Concurrent application execution leads to cache evictions (shared resource) that as a result increases application execution time and surges energy consumption of the smart-phone application. $E_{ISA-Access}$ parameter demonstrates the amount of energy required to access ARM-IS energy profile from the remote cloud server.

$$E_{BaseCostEnergy} = E_{System-Libraries} + E_{App-developer-modules} \tag{4.9}$$

Eq. 4.9 estimates base cost energy of a program. $E_{BaseCostEnergy}$ of a program

University of Malaya

demonstrates the energy consumed by the smart-phone application based on fixed energy cost model of ARM-IS. $E_{BaseCostEnergy}$ of a smart-phone application as depicted in Eq. 4.9 is attributed as the energy consumed by the library functions and user defined modules within smart-phone application. Each library function has a fixed energy cost associated with it depending on the nature of operation it performs when it is called from within the application. Also, $E_{System-Libraries}$ is calculated based on the sum of energy consumed by all the built-in functions within application code as shown in Eq. 4.10. In the said equation, Mod parameter describes a particular user defined module within smart-phone application. In eq. 4.10, E_{LF} states energy consumption of a library function; whereas, $COUNT_{LF}$ determines the execution frequency of a particular library function. For $M = 0$, the number of calls to this module is fixed to one as it represents main function of the application.

$$E_{System-Libraries} = \sum_{Mod=0}^{N} \left(\sum_{LF=0}^{K} E_{LF} \times COUNT_{LF} \right) \tag{4.10}$$

$E_{App-developer-modules}$ demonstrates energy consumed by the part of a smart-phone application written by an application developer to implement the specific functionality of the application. The code within smart-phone application considers modular structuring to reduce the complexity. The total energy of application in terms of application modules is estimated as the sum of energy consumed by individual modules times number of calls to those modules as shown in Eq.4.11.

$$E_{App-developer-modules} = \sum_{n=0}^{M} E_{Modules_n} \times N \tag{4.11}$$

Eq. 4.12 demonstrates the energy estimation within each module of application. For each module, the total energy is attributed in terms of set of instructions executing in sequential flow order ($E_F \times F_{count}$), paths dependent instruction execution

University of Malaya

$(E_{Exeflow_i})$, and a set of statements executing within loops (E_{Loop_i}).

$$E_{Modules} = \sum_{F=0}^{N} E_F \times F_{count} + \sum_{i=0}^{n} E_{Loop_i} + \sum_{i=0}^{n} E_{Exeflow_i} \tag{4.12}$$

Energy consumption of a loop structure highly depends on the count of iterations and size of program enclosed within the body of the loop. Eq. 4.13 demonstrates that total energy of a loop is composed of two blocks including loop body and loop invariant initialization. Considering loop body, being executing repeatedly, SA-LEEF considers cache access pattern to minimize the total energy consumption of an application based on cache access based ARM-IS energy profile. For the first iteration of the loop, SA-LEEF estimates instruction energy based on RAM memory model. However, for the remaining iterations, majority of the time, it fetches instructions from the local cache as loops repeatedly executes same chunk of code over and over. However, it was observed that for the iterations following the first iteration, 20% of the instructions are fetched from the RAM due to load store operations as highlighted in eq. 4.14 and eq. 4.15. In the equation, $E_{Loop-Invariant-Initialization}$ demonstrates energy consumed during initialization of loop invariants. It consists of one MOVE, one LOAD, and one STORE operation.

University of Malaya

$$E_{Loop} = E_{Loop-Block} + E_{Loop-Invariant-Initialization} \tag{4.13}$$

$$E_{Loop-Block} = E_1 + E_2 + E_3 \tag{4.14}$$

$$E_{[1,2,3]} = \begin{cases} E_1 \leftarrow \sum_{i=1}^{N} i_{count} \times E_i, [\forall i, E_i \in RAM-Model] & N = 1 \\ E_2 \leftarrow \sum_{i=1}^{0.8\times N} i_{count} \times E_i, [\forall i, E_i \in Cache-Model] & N > 1 \\ E_3 \leftarrow \sum_{i=1}^{0.2\times N} i_{count} \times E_i, [\forall i, E_i \in RAM-Model] & N > 1 \end{cases} \tag{4.15}$$

Eq. 4.16 models energy consumption of conditional statements based on proposed weighted probability function as described in previous section. Eq. 4.17 and Eq. 4.18 estimates energy consumption of source code blocks following the conditional statements. $E_{Body_{TRUECase}} \times P$ describes the energy consumed by a block following branch statement (branch taken); whereas, $(P-1) \times E_{Body_{FALSECase}}$ represents the total energy consumption if branch is not taken.

$$E_{Exeflow} = E_{Branch_{check}} + E_{Body_{TRUECase}} \times P + (P-1) \times E_{Body_{FALSECase}} \tag{4.16}$$

$$E_{Body_{TRUECase}} = \sum_{i=1}^{N} i_{count} \times E_i \tag{4.17}$$

$$E_{Body_{FALSECase}} = \sum_{i=1}^{N} i_{count} \times E_i \tag{4.18}$$

SA-LEEF, in comparison to existing code analysis based estimation methods, has considered the cache eviction overhead during energy estimation of smart-phone application. The adding of cache eviction overhead has improved the realistic execution environment for SA-LEEF framework. Eq. 4.19 demonstrates the cache eviction energy overhead when concurrently more than one applications are run-

University of Malaya

ning and sharing the system cache. In the mentioned equation, $Time_{Con-exe}$ and $Time_{Program}$ represents the execution time of a program running with its tenants and in isolation, respectively. Alternatively, $Base-Power_{Cache-Evi}$ demonstrates the average power consumed when cache is evicted (Harizopoulos & Ailamaki, 2004).

$$E_{Co-Ex-Ov} = (Time_{Con-exe} - Time_{Program}) \times Base - Power_{Cache-Evi} \tag{4.19}$$

The execution time of a program is statically estimated by dividing size of the program to the processor speed as shown in Eq. 4.20. The size of the program is estimated based on instructions count and CPI of instructions in each category of instructions as shown in Eq. 4.21[2].

$$Time_{Program} = \frac{Program - Size}{Processor - Speed} \tag{4.20}$$

$$Program_{size} = Instructions - count \times CPI \tag{4.21}$$

The energy consumption while accessing the ARM-IS energy profile from the remote cloud is modeled in Eq. 4.22. In Eq. 4.22, $Activity_{time}$ parameter describes the amount of time a particular networking task consumes performing network activity when the file is accessed from the remote cloud. Eq. 4.23 has modeled activity time on Wi-Fi link in terms of throughput and the total data size of the target file. Avg_{Power} of the Wi-Fi module represents the average power consumed during network activity to download a text file from the cloud server. $Activity_{time}$ is estimated based on ratio of data size to the total bandwidth of the system. Bandwidth of the

[2]http://stackoverflow.com/questions/35208398/calculate-cpu-execution-time

University of Malaya

Wi-Fi links is easy to know as many apps at play store are publicly available to estimate network bandwidth.

$$E_{ISA-Access} = Activity_{time} \times Avg_{Power} \tag{4.22}$$

$$Activity_{time} = \frac{Data - size}{Bandwidth} \tag{4.23}$$

4.4 Distinguishing Features of SA-LEEF

This section comprehensively discusses distinguishing features of SA-LEEF in comparison to traditional dynamic analysis based energy estimation schemes.

4.4.1 Offline Usability

Traditional dynamic analysis based energy estimation schemes run target application on smart-phone to estimate its energy consumption. Traditional dynamic analysis based energy estimation schemes either exploit pre-built power models or design their own to estimate energy consumption of the smart-phone application. SA-LEEF exploits static analysis based methodology to estimate energy consumption of smart-phone applications. Being a 2-tier architecture by design, it exploits offline usability feature to schedule resource expensive tasks on resource-rich cloud servers for effective resource usage. Among all operations in static analysis based estimation, energy profiling of ARM-IS is most expensive task. ARM-IS energy profiling is one-time activity as SA-LEEF log it on the remote cloud to offer ARM-IS energy profile as-a-service. Offline usability feature hides per ARM instruction energy estimation finding complexity and offers a simple profile-as-a-service interface. Also, SA-LEEF hosted source code extraction from the executable program at cloud server to optimally utilize underlying resources of smart-phone. Offline usabil-

University of Malaya

ity feature empowered the SA-LEEF framework to augment smart-phone battery lifetime.

4.4.2 Lightweight Design

Traditional dynamic analysis based energy estimation tools inefficiently utilize resources of a smart-phone device during energy consumption estimation of a smart-phone application. Traditional dynamic analysis based energy estimation schemes consume a significant amount of CPU and RAM storage as discussed in chapter 3 while estimating energy consumption of smart-phone application. As discussed in chapter 3, majority of CPU time is consumed while estimating loops and function calls of smart-phone applications. Loops repeatedly execute the same chunk of code to perform specific tasks. Calling a peace of code over and over at different time intervals from different locations increases the execution time of application. Lightweight static analysis estimation design of SA-LEEF suppresses execution time of loops and function calls as it scans application only three times. Moreover, considering dynamic analysis based estimation, estimating the execution path based on the cache-look-ahead buffer is expensive especially when it leads to false prediction. SA-LEEF considers weighted probabilistic estimation approach to find the execution path within the application for effective resource usage. SA-LEEF significantly reduces the energy estimation time and system resources ultimately to achieve lightweight solution.

4.4.3 Non-incentive Based ARM-IS Profile Sharing

Unlike traditional utilization based energy estimation tools, SA-LEEF is more trustable as it is not designed only for a few set of smart-phone devices. It is applicable for all smart-phone devices whose ARM-IS energy profile is available. Traditional utilization based tool is only accurate for the smart-phone for which it is devel-

University of Malaya

oped. For instance, Power Tutor is platform dependent and accurately estimates energy consumption for HTC Desire and Google Nexus One smart-phone devices. Alternatively, SA-LEEF is a generic framework that can be used for all types of smart-phones (ARM-IS-Energy profile-as-a-service). It also offers test programs to the requesting host to locally generate ARM-IS energy profile based on android power APIs. Once ARM-IS energy profile is generated, it offers a non-incentive ARM-IS energy sharing method to store ARM-IS energy profile on a remote cloud server for future requests. This dual profiling mode increases generality and adaptability features of SA-LEEF.

4.4.4 Energy Estimation Support for Native Smart-phone Applications

SA-LEEF support energy estimation of native smart-phone applications. Native smart-phone applications are high-performance applications as they have direct access to the hardware components such as a camera. Estimating energy consumption of an application at high source code level effects its applicability for other high-level languages due to high differences in their syntax. SA-LEEF is easily applicable for all type of applications as it considers an assembly based application to estimate its energy consumption. Estimation based on assembly code is not affected by the optimization within compiler design; therefore, it gives more accurate results compared to high level energy profiling based solutions. For energy estimation, SA-LEEF generates assembly file of an executable program using ARM obj dump program. Obj dump based assembly code represents the lower system operations. SA-LEEF schedules application translation process on cloud servers to augment device battery lifetime.

University of Malaya

4.4.5 Ground For Optimization

Traditional energy estimation tools estimate energy consumption at the coarse granular application level. However, SA-LEEF estimates energy consumption at fine granular loops, functions, instruction, path, and library routine. Also, it estimates energy consumption at the application level. SA-LEEF assists application developers to estimate energy consumption of their mobile applications at earlier development stages. Based on energy assessment of application at earlier development stages, developers can redesign their application for effective battery usage. Due to fine granular energy estimation support, SA-LEEF is an ideal platform that assists developers to optimize their application code to improve battery charge usage.

4.4.6 Green Mobile Computing

Traditional dynamic analysis based energy estimation tools consume a significant amount of smart-phone resources as their estimation time is very high. However, energy estimation time of SA-LEEF framework is very limited as it considers the static analysis of the application for energy estimation. As estimation overhead is proportional to the total execution time of the application, therefore, SA-LEEF imposes limited energy overhead on smart-phone devices during the estimation process. Limited energy overhead augments device battery lifetime. Energy estimation based on lightweight estimation tool is the first step towards green mobile computing.

4.4.7 Non-voluminous Communication Overhead

SA-LEEF exploits network links to download ARM-IS energy profile and assembly code of the smart-phone application (executable) from the cloud servers. The size of these files is very limited. ARM-IS energy and application's assembly version are in simple plain text format. Therefore, the size of these files is in the range of kilobytes normally. As the majority of smart-phones are Wi-Fi enabled or equipped

University of Malaya

with 3G radio, therefore, downloading these files does not require ample time and energy. Also, data transfer over network communication link is a one-time activity. Therefore, it does not significantly impact the total energy consumption budget of smart-phone. MCC considers code size as one of the parameters to decide the execution location of the smart-phone application. In contrast to code size, energy consumption of an application highly depends on the type of operations within it. SA-LEEF when integrated with MCC computational offloading frameworks can improve the accuracy in decision making for execution location of the application.

4.5 Data Design

This section describes method and metrics to evaluate SA-LEEF framework.

4.5.1 SA-LEEF Evaluation Metrics

The proposed study has identified energy estimation accuracy, estimation time, and energy estimation overhead as three metrics to investigate the performance of the proposed framework.

4.5.1.1 Energy Estimation Accuracy

Energy estimation accuracy relates closeness of measured data using SA-LEEF to the ground truth value. The energy estimation accuracy is measured in percentage and is calculated based on the Eq. 4.24. In Eq. 4.24, $Energy_{GroundTruthvalue}$ states the reference value to estimate accuracy. In the current case, this study has chosen Measurement based methods as the reference values ($Energy_{GroundTruthvalue}$) to highlight accuracy of SA-LEEF. Alternatively, $Energy_{SA-LEEF-Estimation}$ states energy consumption of smart-phone application estimated using SA-LEEF frame-

University of Malaya

work.

$$Accuracy = \frac{Energy_{SA-LEEF-Estimation}}{Energy_{GroundTruthvalue}} \times 100 \tag{4.24}$$

4.5.1.2 Energy Estimation Overhead

Energy estimation overhead highlights how much energy the proposed SA-LEEF framework has consumed when estimating energy consumption of the smart-phone application. In this case, the proposed study has chosen a measurement based method to find the estimation overhead of SA-LEEF framework. Eq. 4.25 highlights the model to estimate energy overhead of the proposed SA-LEEF framework. In the mentioned equation, $SA-LEEF_{Estimat-Time}$ describes the total estimation time of SA-LEEF framework to analyze and estimate energy consumption based on application source code. Alternatively, $SA-LEEF_{Power-Consump}$ demonstrates amount of power (average) SA-LEEF consumes during analyzing and estimating smart-phone application energy consumption. The unit of energy estimation overhead is joule.

$$Estimation-Overhead = SA-LEEF_{Estimat-Time} \times SA-LEEF_{Power-Consump} \tag{4.25}$$

4.5.1.3 Energy Estimation Time

Energy estimation time states the amount of time SA-LEEF framework takes to estimate energy consumption of the smart-phone application. It includes the time to estimate base cost energy of smart-phone application, context switching overhead, time for ARM-IS energy profile downloading from the remote cloud server, and user-system activity time as shown in Eq. 4.26. In Eq. 4.26, $T_{BaseCost}$ represents the time SA-LEEF takes while estimating the base cost energy of the smart-phone

University of Malaya

application. Alternatively, $T_{Eviction}$ highlights the context switching time overhead due to cache eviction owing to smart-phone resource sharing among tenant applications. Finally, T_{Inter} and $T_{Download}$ states the user system interaction time and time to download ARM-IS energy profile from the remote cloud server, respectively.

$$Estimation - Time = T_{BaseCost} + T_{Eviction} + T_{Inter} + T_{Download} \tag{4.26}$$

4.6 Conclusions

In this chapter, a 2-tier lightweight energy estimation framework for energy estimation of native smart-phone applications is proposed. Traditional dynamic analysis based energy estimation schemes run the application on the smart-phone device to estimate its energy consumption. However, running smart-phone application on the smart-phone device for energy estimation significantly consumes smart-phone resources. The distinguishing feature of SA-LEEF is to consider static analysis to estimate energy consumption of the smart-phone application. The applicability of static analysis based energy estimation is affected by the non-deterministic execution behavior of smart-phone applications.

SA-LEEF proposes a weighted probability based execution path estimation method to resolve the issue of execution path estimation within a smart-phone application. Moreover, it employed a cache distance based method to predict the storage location of instructions within source code. The slicing based loop estimation feature of SA-LEEF empowered it to estimate bounds for loops. The incorporation of dual profiling mode in SA-LEEF increases its adaptability and generality. Also, the adoption of 2-tier design empowered SA-LEEF to lessen high profiling time and energy estimation overhead to propose a lightweight design. It is concluded that

University of Malaya

SA-LEEF provides a lightweight energy estimation solution for energy estimation of native smart-phone applications.

University of Malaya

CHAPTER 5: EVALUATION

This chapter aim to report evaluation process opted to evaluate SA-LEEF framework. It discusses tools, experimental setup, benchmark applications, evaluation parameters, and performance analysis of proposed framework. The evaluation process analysis performance of different components of SA-LEEF such as ARM Instruction Energy Profiler, Application Analyzer, and Collaborator (network communication energy). It also reports energy consumption overhead due to concurrent application execution on smart-phone device owing to cache eviction. To validate the findings of different modules of SA-LEEF, val-grind application profiling software is used. Alternatively, energy consumption during network activity for ARM-IS and application's assembly code downloading has been tested using the Power Tutor 4.0.1 energy estimation tool.

This chapter is organized into seven sections. Section 5.1 discusses the experimental setup, devices, and benchmark applications for the evaluation of SA-LEEF framework. Section 5.2 discusses the method to estimate energy consumption of a single ARM assembly based instruction within ARM-IS. It presents data collection for highlighting the effect of storage location on energy consumption of an instruction. It presents an offline neighborhood-based noise suppression method to remove outliers from the collected data. Section 5.3 discusses network energy consumption cost while downloading application's assembly code and ARM-IS energy profile from local and remote servers. Section 5.4 debates on method and data collection for concurrent program execution on a smart-phone device. Section 5.5 analysis application analyzer module of SA-LEEF framework to count and classify the instructions within a smart-phone application. Section 5.6 discusses the method and base cost power consumption of SA-LEEF framework. Section 5.7 concludes the

University of Malaya

whole chapter to verify the objectives of this chapter by presenting an overview of findings.

5.1 Evaluation of Proposed SA-LEEF Framework

SA-LEEF framework is designed for energy estimation of native smart-phone applications based on static analysis method. SA-LEEF estimates energy consumption based on the assembly code version of the smart-phone application. SA-LEEF framework receives input from ARM instruction energy profiler, application analyzer, and collaborator modules, to estimate energy consumption of the smart-phone application.

A prototype of SA-LEEF is developed and deployed on a smart-phone device and a nearby server to host ARM-IS energy profile. Multiple compute-intensive benchmark applications are used to analyze the performance of SA-LEEF. The data for benchmark applications is collected through application profiling. For validation of application profiling process, Val-grand application profiling tool is used. The energy consumption during network activity is collected through the Power Tutor energy estimation tool. For network power consumption estimation, client/server tests are performed to analyze energy consumption during application downloading process for different sizes of data. For SA-LEEF implementation, ARM assembly, C++ library, and EM6000 tool Kit are used. Server side ARM instruction energy profiler module exploits Linux based ARM-objdump utility to generate assembly code version of the native smart-phone application. Compared to non-native smart-phone applications, native smart-phone applications are of high performance. Given below is a brief description of the experimental setup, devices used during experimentation, and benchmark applications.

University of Malaya

5.1.1 Evaluation Set-up

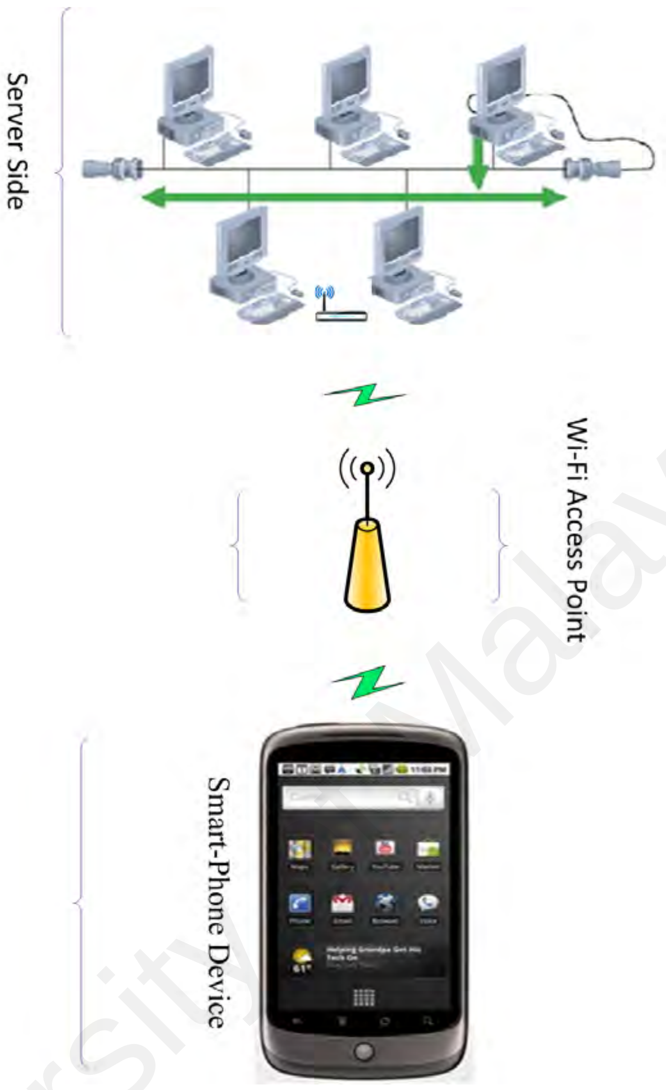

Figure 5.1: An Overview of Evaluation Setup for SA-LEEF Framework

The proposed energy estimation framework is evaluated based on experimentation on real devices. The main reasons for considering real experimentation are the high estimation time for simulation-based methods (application profiling case). Also, the existing simulation-based methods either does not support smart-phone application energy estimation or are not mature enough to provide technical capabilities to estimate energy consumption of the smart-phone application. Alternatively, real device based experimentation gives an in-depth knowledge about the performance of the system under observation for multiple parameters. Simulation-based solutions, as they simulate the real smart-phone devices, are vulnerable to results

University of Malaya

skewing and estimation that leads to low accuracy.

The physical server hosts ARM-IS energy profile and is located within the premises of a building as depicted in Fig. 5.1. On smart-phone device, a prototype of SA-LEEF runs that accesses assembly code of the native smart-phone application and ARM-IS energy profile for the target smart-phone architecture from the server. It analyzes the smart-phone application and employs ARM IS energy profile to estimate energy consumption of the application.

5.1.1.1 Experimental Devices

To prepare the experimentation, one smart-phone device, one desktop server, Cisco Linksys WRT54GL wireless communication access point, Multi-meter, and a sense resistor are used. The chosen desktop server is very powerful as it carries Intel i5-2500 processor having 4GB RAM capacity, 3.3 GHz clock speed, 32-bit Windows 7 professional, and 1 TB storage, accessible via a wireless connection. The desktop server hosts Power Profile DB, Remote handler, and ARM instruction Energy profiler module of the SA-LEEF framework. Smart-phone device hosts application analyzer, collaborator, and application energy estimator modules of SA-LEEF for energy estimation of smart-phone applications. Main features of Google Nexus One smart-phone device includes, (a) Dual Core 1.0 GHz Cortex-A9 CPU, (b) 802.11 a/b/g Wi-Fi radio, (c) 1GB RAM storage, (d) 32 GB storage, (e) android v2.3.6, (f) Adreno 200 GPU, and (g) 1400mAh Li-ion battery. For wireless communication between smart-phone and desktop server, Cisco Linksys WRT54GL is used.

5.1.1.2 Benchmark Applications

To evaluate the performance behavior of SA-LEEF framework, this study has considered several benchmark applications. Chosen benchmark applications investigate CPU performance and memory data access speed. There are four main reasons be-

University of Malaya

hind choosing the selected benchmark applications. Firstly, as SA-LEEF targets energy estimation of disk I/O based applications; therefore, during benchmark selection process it was ensured that chosen benchmark application does not require human interaction during its execution. Secondly, the framework operates on native smart-phone applications (coded in C/C++). Therefore, during the selection process, this study considered that chosen benchmark should be coded in C/C++ language. Thirdly, SA-LEEF considers energy estimation of disk I/O based applications that frequently operates on the storage memory of smart-phone. Therefore, this study has selected only those benchmark applications that heavily performs memory operations. Lastly, while choosing benchmark applications, it is ensured that the selected benchmark is an open source application.

This study has selected nine benchmark applications while considering the constraints as mentioned above. The chosen benchmark applications include MemSpeedi, Dhrystone2, RandMemi, FFT1, Nested Branches, Factorial, NativeWhetstone2, LivermoreLoops2, and LinpackSP2. Given below is a brief overview of selected benchmark applications.

MemSpeedi MemSpeedi investigates RAM and cache data access rate in megabytes per second in the range of $2 \times 8KB (16KB)$ until $2 \times 32MB (65536KB)$. It performs operation on arrays of cache and RAM based data to estimate memory data accessing speed. It operates on single and double precision float point data using $x[m] = x[m] + s * y[m]$ and $x[m] = x[m] + y[m]$ calculations. Alternatively, for single and double integer operations it employed $x[m] = x[m] + s + y[m]$ and $x[m] = x[m] + y[m]$, respectively. MemSpeedi operates on 64MB size array of integer values to access data from different offsets such as 1, 2, 4, 8, 16, and 32 (Batyuk et al., 2009).

Dhrystone2 Dhrystone2 benchmark is an integer performance measurement

University of Malaya

benchmark application. It analysis performance of CPU in terms of million of instructions per second (MIPS). Dhrystone2 has two versions including non optimized and optimized version. In this study optimized compiled version of dhrystone2 for the experiments is used. The code and data sizes of dhrystone2 benchmark is very small. It includes basic arithmetic and string based operations (Weiss, 2002).

Nested Branches Nested branches is a synthetic benchmark and is composed of a set of basic arithmetic instructions written inside the branch statements. The code within branch statements performs integer and floating point operations on the arrays. The operations on array repeatedly update the stored elements to frequently update the data inside RAM memory. Inside Nested Branches benchmark, the depth of branching is set up to four levels to execute a bunch of code.

Factorial Factorial is a synthetic benchmark application that calculates factorial of an integer number using recursion as highlighted in Eq. 5.1. It is a compute bound benchmark application to measure the performance of CPU. In the estimation method, 99999 was chosen as input number to estimate energy consumption during factorial calculation. The loop within factorial benchmark implements Eq. 5.1 to recursively call itself for factorial calculation.

$$N! = N \times (N-1)!, N > 0 \tag{5.1}$$

RandMemi RandMemi benchmark is a memory performance measurement benchmark application (Weiss, 2002). The detail on NativeWhetstone2, LivermoreLoops2, and LinpackSP2 is provided in chapter 3. During the experiments, default settings of all the chosen benchmark applications are considered [1].

[1]www.roylongbottom.org.uk/android%20benchmarks.htm

University of Malaya

5.2 Data Collection for ARM-IS Energy Consumption

This section empirically validates ARM-IS energy profiling module of SA-LEEF framework. It reports data collection for accessing and executing assembly based code of an application from RAM storage and local cache. It also reports the effect of registers reordering within the arguments of an instruction to the total energy consumption of a an assembly based instruction.

Energy consumption of a single instruction is modeled as the energy cost during different CPU pipeline stages such as instruction fetch, decode, execute, store, and memory write-back operations. Each CPU pipeline stage exhibits dissimilar execution time and energy consumption. However, execution time and energy consumption across CPU pipeline stages is very minimal; also, all smart-phone model architectures does not support finding energy consumption at these lower architecture levels. Therefore, this study has designed and tested different test programs to estimate energy consumption of each ARM-IS instruction.

5.2.1 Test Program Design and Power Measurement Setup

This section discusses the design of test programs proposed to estimate energy consumption of ARM-IS instructions. It also discusses measurement setup for profiling power consumption of test programs.

ARM-IS defines set of operations that a particular ARM architecture performs. The execution time for each ARM-IS instruction (e.g., ADD, SUB, MUL, and LOAD) is different and highly depends on the functionality it performs. The assembly based ARM instructions takes very limited time when it is executed on ARM based smart-phone devices. Therefore, it is difficult to directly estimate energy consumption for a single ARM based assembly operation. To handle this issue, a set of test programs are developed that assists in estimating energy consumption

University of Malaya

of a single ARM based assembly instruction. Each test program is designated to estimate energy consumption of a single ARM based assembly operation (Tiwari et al., 1994).

Design of a test program comprises a sequence of same instance of an ARM assembly instruction enclosed in tight loops executing millions of times as shown in Fig. 5.2. The size of each test program consists of 2×10^{12} instances of an instruction (Appendix B). The size of the test program represents the number of instructions that are executed during its execution on smart-phone device. The aforementioned size of test program is chosen such that it collects sufficient power readings to estimate energy consumption for the target ARM assembly based instructions. For instance, considering test program for ARM assembly based LOAD operation, it took 80s when executed on chosen smart-phone device.

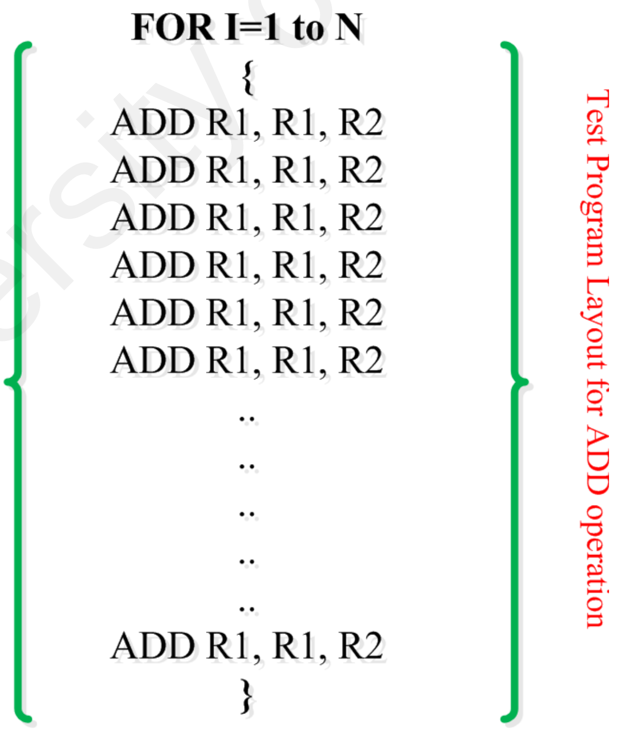
```
FOR I=1 to N
{
ADD R1, R1, R2
ADD R1, R1, R2
ADD R1, R1, R2
ADD R1, R1, R2
ADD R1, R1, R2
ADD R1, R1, R2
..
..
..
..
..
ADD R1, R1, R2
}
```

Figure 5.2: Test Program Design (Cache Based Storage Location Analysis)

During application execution, instructions are either loaded from cache or memory storage locations. Therefore, two types of test programs are designed. First

University of Malaya

category of test programs accesses data and instructions from local cache. Alternatively, second category of test programs accesses data and instructions from RAM storage. Design of a test program belonging to first category of test programs (Cache based access) executes instructions sequence in loops to increase the probability of cache hits. The size of the loop is chosen based on the upper bound estimation of cache size (16K). For the second category of test programs (RAM based access), the proposed test programs comprise of instructions executing in a sequence. In the sequential order of execution, CPU always fetches instructions from the RAM storage.

The energy consumption of an ARM assembly instruction is estimated based on the execution profile of the test program. As discussed in chapter 4, for each test program, the total execution time, size of the test program, and average power consumption rate assists estimating its energy consumption. For power measurement for the test programs, EM6000 multimeter is used. The main reason of choosing EM6000 multimeter[2] is its high rate to capture enough power samples. EM6000 captures three power samples per second. Also, EM6000 is robust and can capture voltage/current drop up to 6000 maximum value. It measures true root mean square (RMS) values of AC voltage and current. It is accurate and offers 5% to 95% duty cycles. The other differentiating features of EM6000 includes, 600m/6/60/600V DC voltage, 6/60/600 AC voltage, 9.9999.99 MHz frequency, 4000 Micro farad capacitance, and approximately 3.0V diode check. EM6000 captures voltage and current drop across a resistor attached to the power rail of a smart-phone device. We have chosen 1 ohm resistor to attach it to the battery terminals of the smart-phone device for voltage/current measurements.

Fig. 5.3 demonstrates the experimental setup to estimate energy consumption

[2]www.all-sun.com/en/d.aspx

University of Malaya

of ARM IS based assembly instructions. In the designed circuit, the battery is externally connected via a high precision resistor to the smart-phone battery terminals to record voltage and current drop as shown in Fig. 5.3. However, inserting a resistor increases the resistance of the circuit. But, as the chosen sensor offers low resistivity; therefore, resistance of circuit increases with low rate (less than 1%). To capture voltage and current drop during test program execution on the smart-phone device, EM6000 is interfaced to the sense resistor as shown in Fig. 5.3.

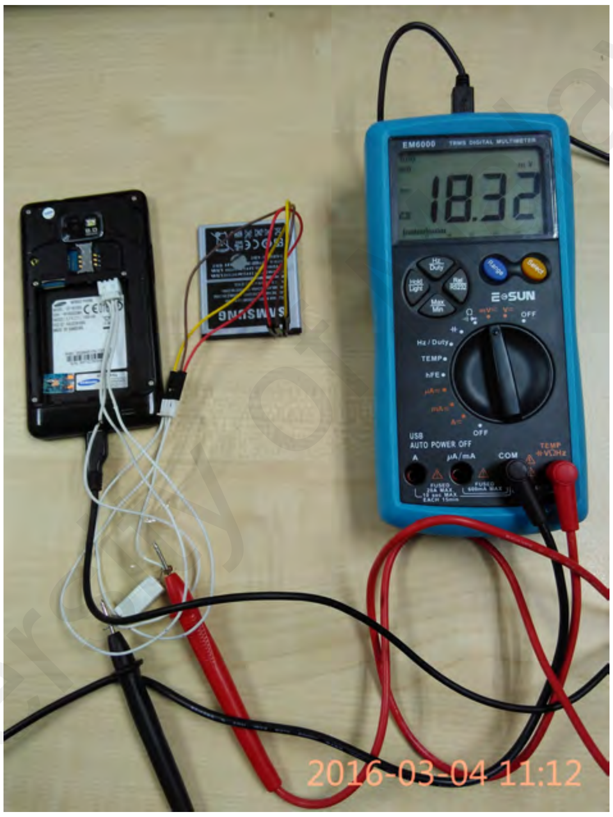

Figure 5.3: Experimental Setup for ARM-IS Energy Profiling

The energy consumed by each test program is calculated based on Eq. 5.2 and Eq. 5.3. Here, "T" represents the total execution time of test program, whereas, "P" states average power consumed in terms of current (I) and the voltage drop (V_{CC}) during test program execution on smart-phone. Energy consumption for a single instruction in a test program is estimated using Eq. 5.4. In Eq. 5.4, $Size_{TestProgram}$ represents the total size of the test program. In current case, $Size_{TestProgram}$ is 2×10^{12}. Furthermore, $E_{TestProgram}$ demonstrates total energy consumed during

University of Malaya

test program execution on smart-phone device. The power profile of an instruction contains various noise values due to background processes running on smart-phone. To suppress the noise, neighborhood operation is selected to guaranty high estimation accuracy.

$$E_{TestProgram} = P \times T \tag{5.2}$$

where,

$$P = I \times V_{CC} \tag{5.3}$$

$$E_{Instruction} = \frac{E_{TestProgram}}{Size_{TestProgram}} \tag{5.4}$$

The power consumption of OLED screen highly depends on its brightness level (Carroll & Heiser, 2010). In order to validate the experimental setup as shown in Fig. 5.4, the experiments are performed to analysis power consumption for chosen smart-phone device. Prior to experiments, all unnecessary applications and sensors were turned off to minimize the noise. In Fig. 5.4, power consumption analysis is performed for three modes of smart-phone including Idle, Idle Full-Back-light, and Idle Dim-Light. In Idle mode, smart-phone does not perform any activity except running background processes to manage its resources. In idle Full Back-light mode, the back-light of OLED screen was set to its full brightness level. However, for idle Dim-Back-light mode, the brightness level of OLED was set to dim level. The experimental setup has correctly estimated the power consumption for smart-phone for its three modes. In the experiment it was noticed that brightness level of OLED effects the total power consumption. It was noticed that the power consumption of smart-phone while setting the brightness to its highest brightness level is greater than its dim

University of Malaya

brightness mode.

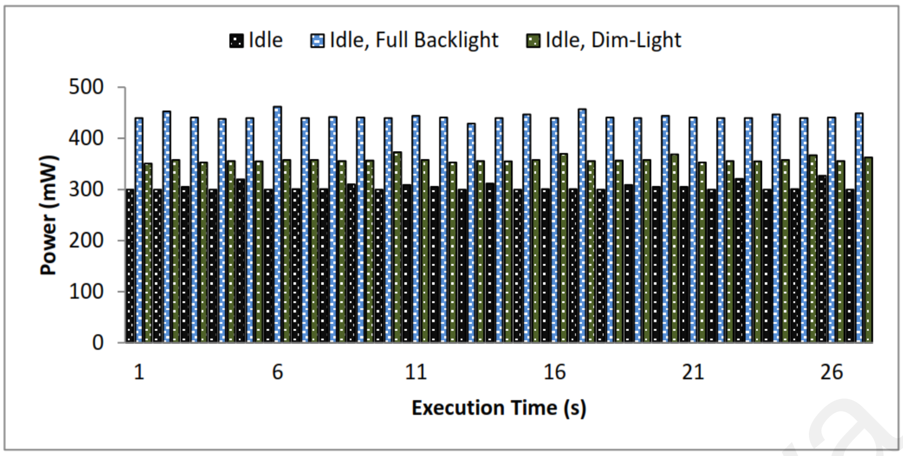

Figure 5.4: Google Nexus One's Power Consumption Analysis for its Idle Execution Mode

5.2.2 Analysis of Outliers in ARM-IS Power Profile

Power profile of a test program exhibits few abnormal power peaks called noise data. The main reason of this noise is the resource sharing by tenant processes running in the background of smart-phone device or tail energy of smart-phone components. In this section, the chosen method to suppress this noise is discussed.

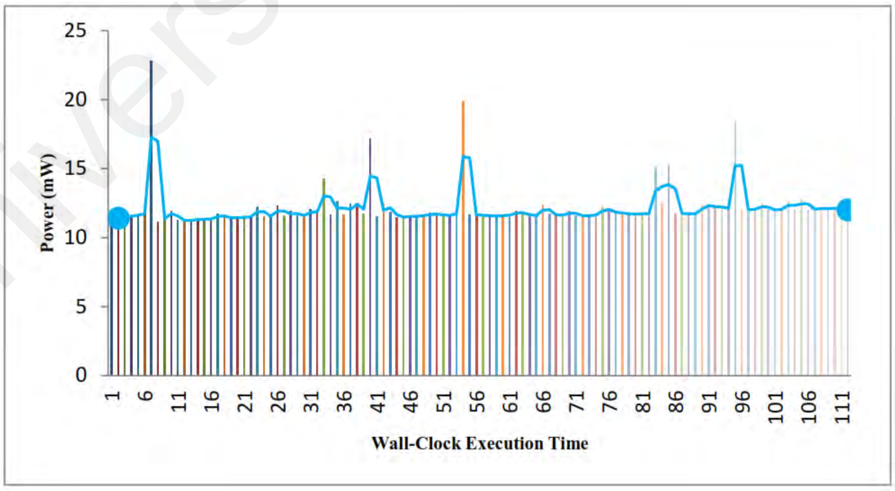

Figure 5.5: Test Program Power Consumption Behavior (AND Operation)

Fig. 5.5 and Fig. 5.6 demonstrates power consumption behavior of a test program on smart-phone device. In the mentioned figures, X-axis represents the execution time, whereas, Y-axis highlights power consumption in milliwatt (mW). As

University of Malaya

depicted from Fig. 5.5 and Fig. 5.6, power consumption varies a little over total execution time frame except for a few high power peaks. For instance, in Fig. 5.5 at execution time 6, 41, 55, 83, 86, and 95s, the power consumption is 23mW, 17.3mW, 19.8mW, 15.2mW, 15.1mW, and 18.2mW, respectively. To handle this issue this study has used neighborhood operation to suppress the effect of noise due to background running processes in offline mode (Bandyopadhyay, Chakraborty, Bag, & Das, 2016).

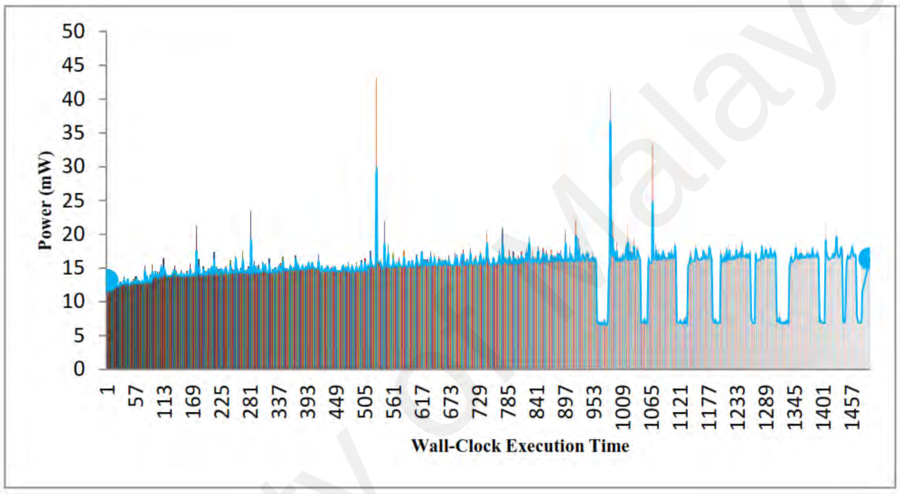

Figure 5.6: Test Program Energy Consumption Behavior (ORR Operation)

University of Malaya

Neighborhood operation operates on timestamped power profile of target test program to suppress the noise. Firstly, it calculates divergence of every power peak value from the average power consumption of test program. Secondly, it marks a power consumption reading as noise if the distance from the average is very high. To suppress the noise, it considers 8 power readings in surrounding of the power reading with noise to replace it with their average value. For $time = 6$ in Fig. 5.5(an abnormal power peak at time "t"), neighborhood operation employs "$t_6 \leftarrow \frac{\sum_{t=2}^{10} P_t}{8}$" to estimate true power consumption for it.

5.2.3 ARM-IS Energy Consumption

This section reports average energy cost for executing a single assembly instruction of ARM-IS for ARM7 architecture. It collects data for energy consumption of executing a single instruction fetched from local cache and RAM storage.

Table 5.1 highlights average energy cost for assembly based ARM instructions accessed from RAM storage. The energy cost of each instruction is different and highly depends on the CPI for each ARM instruction. To find the average energy consumption of an instruction, the experiment has been repeated ten times to suppress the effect of background activities on estimation accuracy. The standard deviation from the mean is reported in Table 5.1. The value of standard deviation from the mean is noticed very minimal. For instance, for ADD, MOV, SUB, MUL, and STR instructions, the average energy consumption is noticed 1.27×10^{-06}, 1.12×10^{-06}, 1.30×10^{-06}, 1.95×10^{-06}, and 8.5×10^{-06}, respectively. Alternatively, standard deviation for ten runs from the mean is observed 2.60×10^{-08}, 2.98×10^{-08}, 2.58×10^{-08}, 6.95×10^{-08}, and 7.42×10^{-08}, for aforementioned ARM assembly instructions. The main reason for the small variation in here mentioned standard deviation is the application of neighborhood operation on test program power profile for noise suppression.

Table 5.1: ARM-IS Energy Profile for Google Nexus One (RAM Storage Location)

Instruction	Energy (J)	ST.Dev	Instruction	Energy (J)	St.Dev
ADD	1.27×10^{-06}	2.60×10^{-08}	CMP	1.18×10^{-06}	3.00×10^{-08}
MOV	1.12×10^{-06}	2.98×10^{-08}	ORR	1.12×10^{-06}	2.71×10^{-08}
SUB	1.30×10^{-06}	2.58×10^{-08}	EOR	1.24×10^{-06}	2.58×10^{-08}
MUL	1.95×10^{-06}	6.95×10^{-08}	AND	1.13×10^{-06}	2.78×10^{-08}
B	1.27×10^{-05}	9.56×10^{-07}	CMN	1.18×10^{-06}	3.62×10^{-08}
RSB	1.39×10^{-06}	2.80×10^{-08}	TEQ	6.3×10^{-06}	3.63×10^{-08}
LDR	8.92×10^{-06}	8.19×10^{-08}	TST	6.2×10^{-06}	3.87×10^{-08}
LSL	1.49×10^{-06}	2.51×10^{-08}	SMULL	6.4×10^{-06}	5.91×10^{-08}
MLA	5.5×10^{-06}	4.98×10^{-08}	SRC	1.2×10^{-06}	3.05×10^{-08}
STR	8.5×10^{-06}	7.42×10^{-08}	MOVN	1.77×10^{-06}	3.01×10^{-08}

University of Malaya

Access to the cache memory is faster compared to RAM storage due to high latency of RAM storage. Eq. 5.5 highlights total access time for a program while considering system cache and memory as target storage locations. In the here-mentioned equation, HIT_{Rate} describes the fraction of data accessed from the cache. Alternatively, miss rate represents a fraction of data accessed from the main memory. Also, $Cache_{AccessTime}$ and $RAM_{AccessTime}$ represents the total time taken by the system to access data/instruction from cache and main memory, respectively (Kong, 2015; Neglia et al., 2016). For instance, in the majority of the systems, main memory access time is 100 ns. Cache is proven to be 10 times faster than the main memory. Consider a program that yields 0.92 hit ratio for read requests and also suppose that 85% of the memory requests generated by the CPU are for read operation. To yield effective access time, the values are substituted in Eq. 5.5 as shown in Eq. 5.6.

$$Effective_{AccessTime} = HIT_{Rate} \times Cache_{AccessTime} + MIS_{Rate} \times RAM_{AccessTime} \tag{5.5}$$

$$Effective_{AccessTime} = 0.92 \times 10 + (1 - 0.92) \times 100 \approx 17ns \tag{5.6}$$

Table 5.2 highlights average energy cost for assembly based ARM instructions accessed from cache storage. The standard deviation from the mean is reported in Table 5.2 to highlight the variance in results for multiple runs of application. For the cache based access, it is observed that average energy cost of an instruction is very small ($E^{-}10$). This study has repeated the experiments ten times to minimize the chances of the effect of background activities on estimation accuracy. The observed standard deviation from the mean is noticed very low. For instance, for

University of Malaya

ADD, MOV, SUB, MUL, and STR instructions, the average energy consumption is noticed 4.26×10^{-10}, 3.76×10^{-10}, 4.369×10^{-10}, 1.66×10^{-09}, and 7.15×10^{-09}, respectively. Alternatively, standard deviation from the mean is observed 1.4×10^{-10}, 1.9×10^{-10}, 1.5×10^{-10}, 4.6×10^{-10}, and 8.6×10^{-10}, for aforementioned ARM assembly instructions. The main reason for the small variation in here mentioned standard deviation is the application of neighborhood operation on test program power profile for noise suppression. From Table 5.1 and Table. 5.2 it can be analyzed that energy consumption of an instruction, when fetched from the local cache, is lower than RAM storage access. This is due to the fact that accessing data from cache imposes lower access latency compared to RAM based access.

Table 5.2: ARM-IS Energy Profile for Google Nexus One (Cache Access)

Instruction	Energy (J)	St.Dev	Instruction	Energy (J)	St.Dev
ADD	4.26×10^{-10}	1.4×10^{-11}	CMP	3.96×10^{-10}	2.3×10^{-11}
MOV	3.76×10^{-10}	1.9×10^{-11}	ORR	3.76×10^{-10}	2.2×10^{-11}
SUB	4.369×10^{-10}	1.5×10^{-11}	EOR	4.16×10^{-10}	2.5×10^{-11}
MUL	1.66×10^{-09}	0.1×10^{-10}	AND	3.66×10^{-10}	3.1×10^{-11}
RSL	4.27×10^{-10}	3.0×10^{-11}	CMN	3.96×10^{-10}	3.4×10^{-11}
RSB	4.60×10^{-10}	2.2×10^{-11}	TEQ	3.86×10^{-10}	3.7×10^{-11}
LDR	6.89×10^{-09}	1.9×10^{-10}	TST	3.66×10^{-10}	2.3×10^{-11}
LSL	4.69×10^{-10}	3.6×10^{-11}	SMULL	2.44×10^{-09}	0.6×10^{-10}
MLA	2.24×10^{-09}	0.5×10^{-10}	TST	2.42×10^{-09}	0.1×10^{-10}
STR	7.15×10^{-09}	1.2×10^{-10}	MOVN	5.94×10^{-10}	3.7×10^{-11}

Table 5.3 demonstrates average energy consumption of instructions of ARM-IS operating on float point values. This study has repeated the experiments ten times to report an average and standard deviation from the mean for the energy consumption of ARM-IS instructions. Also, it reports energy consumption of several library functions related to math, input/output, and string manipulation operations. From the Table. 5.3 it is seen that energy consumption while performing floating point arithmetic operations is more expensive than integer based operations. This is because of high latency while considering floating point numerical operations. For

University of Malaya

instance, addition operation on integer operands take 1 cycle whereas floating point addition consumes 3 cycles for ARM-7 architecture. In Table 5.3, the instructions with suffix "S"demonstrates single precision operations whereas remaining instructions demonstrate double precision. In compassion to single precision operations, double precision based instruction are more energy consuming as it involves comparatively complex operational logic.

Table 5.3: Floating-point Operations and Library Functions Energy Consumption Profile

Instruction	Energy (J)	St. Dev	Instruction	Energy (J)	St. Dev
FADDS	7.033×10^{-6}	1.2×10^{-7}	FMULTD	1.80×10^{-6}	2.9×10^{-7}
FDIVS	4.715×10^{-5}	2.0×10^{-6}	FSUBD	7.13×10^{-6}	1.6×10^{-7}
FMULS	7.03×10^{-5}	2.5×10^{-6}	FCPYS	6.93×10^{-6}	8.2×10^{-7}
FSUBS	7.03×10^{-6}	1.4×10^{-7}	FNEGS	7.03×10^{-6}	4.5×10^{-7}
FADDD	7.13×10^{-6}	3.7×10^{-7}	FLDS	1.188×10^{-6}	1.4×10^{-7}
FLDD	1.45×10^{-5}	1.5×10^{-6}	FSTD	1.545×10^{-5}	2.5×10^{-6}
FDIVD	8.39×10^{-5}	2.5×10^{-6}	FSTS	1.44×10^{-5}	2.7×10^{-6}
SQRT	1.00×10^{-05}	3.7×10^{-6}	STRCMP	1.17×10^{-5}	4.7×10^{-6}
STRCAT	1.17×10^{-5}	4.5×10^{-6}	PRINTF	2.75×10^{-5}	6.9×10^{-6}
ABS	2.02×10^{-6}	4.2×10^{-7}	ISDIG	1.99×10^{-6}	3.4×10^{-7}
SCAN	1.99×10^{-5}	6.9×10^{-6}	FSEEK	3.27×10^{-5}	7.3×10^{-6}

University of Malaya

5.3 Data Collection for Network Energy Communication Cost

This section validates collaborator module of SA-LEEF to estimates energy consumption during network activity. It discusses the method to estimate energy consumption while accessing ARM-IS energy profile and assembly code version of the smart-phone application from the remote/local server. It has collected data for accessing data files using Wi-Fi and 3G network links. It has collected data for accessing data from both local server and remote cloud server. The collaborator module of SA-LEEF is responsible for downloading text files from the remote/local server.

The data for energy consumption during network activity is collected using the Power Tutor energy estimation tool. To evaluate network energy cost, test programs are designed to replicate different network environments. Total energy consumption is estimated based on the average power consumption during activity on communication link (Wi-Fi, 3G) and total activity time.

5.3.1 Analysis of Wi-Fi Energy Consumption

This section discusses energy consumed by Wi-Fi network connection when files of different sizes are downloaded from a local and remote server. Energy cost of Wi-Fi link highly depends on the amount of data transferred over the network link and distance to the file hosting server. In experiments, data is collected when data files of different sizes such as 1KB, 10KB, and 100KB, are downloaded from the server. The main reason for choosing these three file sizes for the analysis is this that data size of ARM-IS energy profile and assembly code of application usually lied in this range. Cisco Linksys WRT54GL wireless communication access point is used to access the hosting server. The round trip time (RTT) of the link is adjusted to 25ms and 50ms to simulate a local and remote cloud server link. Among two RTTs, 25ms simulates accessing data files from the local server usually located within the premises of a building. Alternatively, 50ms RTT represents the total RTT for accessing the data center located in Singapore (measured using speedtest.net).

Table 5.4 and Table 5.5 demonstrates energy consumption data collected for Wi-Fi network interface while downloading data files from local and remote server. In aforementioned tables, text file size, energy consumed by Wi-Fi link, standard deviation from the mean, and confidence interval for 95% percentile are presented. The reported energy consumption is average of seven runs of the test program on smart-phone. The average energy consumed by Wi-Fi link for 25ms RTT while downloading files of size 1KB, 10KB, and 100KB is noticed 66mJ, 330mJ, and

University of Malaya

576mJ, respectively. Alternatively, average energy consumed by Wi-Fi link for 50ms RTT while downloading files of size 1KB, 10KB, and 100KB is noticed 108.4mJ, 542mJ, and 979mJ, respectively. It is noticed that for high RTT value average file downloading energy consumption is high. The main reason of this behavior is the high energy estimation time for larger RTT values. While accessing data from remote server (high RTT value), the total file downloading time is invested in the packet queuing time, source to destination routing time, propagation delay, and packet loss ratio. The standard deviation of collected data from mean for 25ms RTT is noticed 1.1401, 2.5099, and 4.1472 for 1kB, 10KB, and 100Kb file downloads, respectively. Also, the confidence interval for 95% percentile is estimated 1.4157, 3.1165, and 5.1495 for accessing data files from the local server as shown in Table. 5.4. Alternatively, the confidence interval for 95% percentile is estimated 2.1879, 4.4283, and 5.2105 for accessing data files from the remote server as shown in Table. 5.5. The main reason of high value of standard deviation is the uncontrollable network traffic ongoing on the network that yields network congestion and high packet loss. Fig. 5.7 compares energy consumption behavior when data is downloaded using Wi-Fi network connection.

University of Malaya

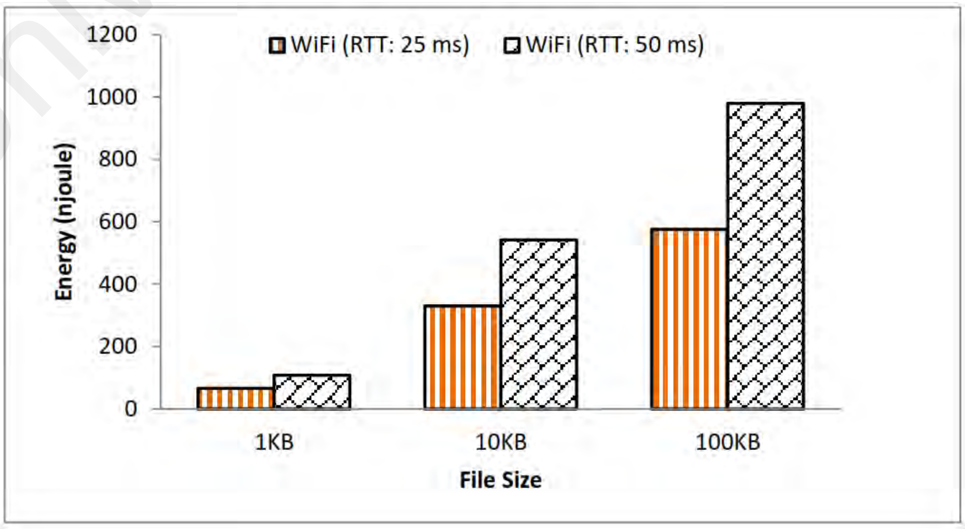

Figure 5.7: Wi-Fi Energy Consumption Analysis

Table 5.4: Network Energy Consumption for File Downloading (Local Server)

Data Size	Avg. Energy Consumption (mJ)	Standard Deviation	Confidence Interval
1KB	66	1.1401	1.4157
10KB	330	2.5099	3.1165
100KB	576	4.1472	5.1495

Table 5.5: Network Energy Consumption for File Downloading (Remote Server)

Data Size	Avg. Energy Consumption (mJ)	Standard Deviation	Confidence Interval
1KB	108.4	1.7621	2.1879
10KB	542	3.5628	4.4238
100KB	979	4.1964	5.2105

5.3.2 Analysis of 3G Network Energy Consumption

This section analysis 3G network to collect energy consumption data while downloading data files of different sizes from the remote cloud. As is the case of 3G networks, Wi-Fi is not ubiquitously available everywhere owing to unavailability of Wi-Fi hot spots. To ensure high reliability, this study has investigated and reported energy consumption of 3G network link as discussed below.

During experiments, this study has ignored the energy consumed by the smartphone device while establishing the network connection to the remote cloud. Also, during 3G network energy estimation, Wi-Fi network connection was switched off to avoid the effect of noise on estimation accuracy. This experiment has simulated 3G radio while adjusting RTT to be 200ms (remote server). Fig. 5.8 represents the energy consumption of 3G module when RTT of 200ms is simulated. As can be seen from Fig. 5.8, large-sized data file downloading process yields longer activity time on the network link. As a result, energy consumption for downloading large sized data files is high.

In comparison to Wi-Fi network link, the energy consumption during files downloading from the remote server using 3G link is higher because of larger bandwidth of Wi-Fi network connection. The energy consumption of 3G network connection depends on, (a) network congestion, (b) data rate, (c) signal to noise ratio (SNR) on

University of Malaya

channel, and (d) routing latency. For energy estimation, this study have repeated the experiments seven times to report the average energy consumption while downloading files using 3G network link. Using 3G network link, the downloading process has consumed 261.6mJ, 1308mJ, and 2401mJ energy, for 1KB, 10KB, and 100KB file sizes downlands, respectively. The energy consumption for 3G network while downloading 100KB file is more than the 10KB. The main reason of this behavior is because of the fact that the downloading time of 100KB is greater than 10KB for 3G network link. The standard deviation of the runs from mean for 1KB, 10KB, and 100KB size files is noticed 2.9453, 3.8725, and 6.9034, respectively. The main reason of high standard deviation is the unpredictable and uncontrollable network traffic ongoing on network links. The confidence interval for 95% percentile is observed 1.4491, 4.8083, and 8.5717, for aforementioned file sizes.

University of Malaya

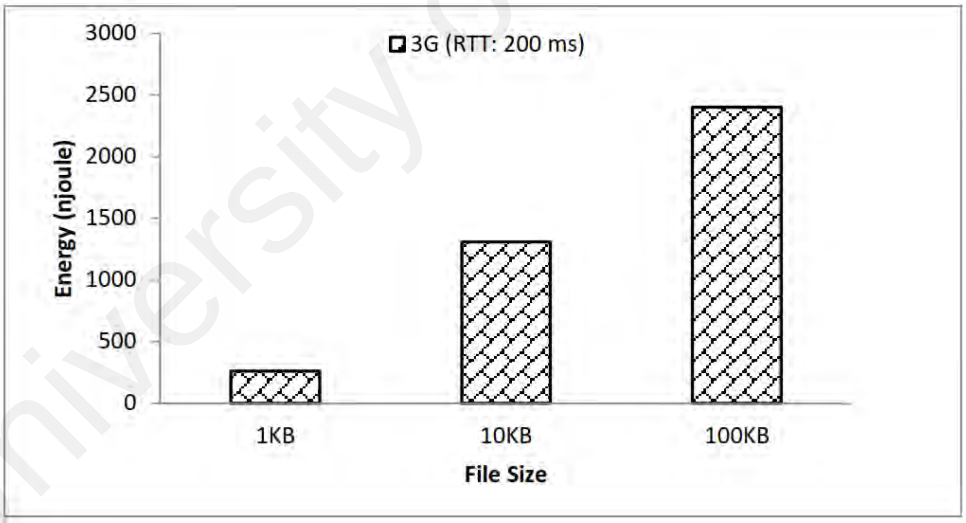

Figure 5.8: 3G Energy Consumption Analysis

5.3.3 Analysis of Energy Consumption for Benchmark Applications

Table 5.6 demonstrates total energy consumption while accessing assembly code of benchmark application and ARM-IS energy profile for a particular smart-phone architecture from the server. The energy consumption of each benchmark is attributed based on the Power Tutor energy estimation tool. The size of ARM-IS energy profile is fixed and is 446byte for the selected smart-phone device. For Wi-Fi network connection, downloading ARM-IS energy profile costs 66mJ and 108.4mJ for both local and remote server, respectively. Similarly, for 3G network connection, downloading ARM-IS energy profile costs 261.6mJ energy. In Table 5.6, the average energy consumption highlights energy cost for downloading assembly source code of benchmark application and ARM-IS energy profile. It was assumed that the assembly code of all the benchmark applications is already available at cloud server. Therefore, the energy cost while waiting for conversion of C based benchmark application to assembly code is not added in this cost.

Among all the selected benchmark applications, the network energy cost of LivermoreLoops2 is highest. The network energy cost while considering Wi-Fi network connection (Local Server) for NativeWhetstone2, LinpackSP2, FFT1, Factorial, and Dhrystone2 is estimated 340mJ, 351mJ, 348mJ, 66mJ, and 81mJ, respectively. Similarly, the network energy cost while considering Wi-Fi network connection (Remote Server) for NativeWhetstone2, LinpackSP2, FFT1, Factorial, and Dhrystone2 is estimated 565 mJ, 575mJ, 572mJ, 108.4mJ, and 118.4mJ, respectively. However, for 3G network connection, the energy cost of accessing NativeWhetstone2, LinpackSP2, FFT1, Factorial, and Dhrystone2 is estimated 1329mJ, 1342mJ, 1339mJ, 261.6mJ, and 267mJ, respectively. The energy cost of each of benchmark applications is based on the size of the benchmark applications. For instance, the size of LivermoreLoops2 is noticed largest among all the chosen benchmark applications. As a result, downloading energy cost of LiveremoreLoops2 is highest among all chosen benchmark applications.

University of Malaya

Table 5.6: Energy Consumption Cost for Network Communication for Benchmark Applications

Benchmarks	Energy (Wi-Fi-25ms)	Energy(Wi-Fi-50ms)	Energy(3G-mJ)
NativeWhetStone2	340mJ	565mJ	1329mJ
LinpackSP2	351mJ	575mJ	1342mJ
FFT1	348mJ	572mJ	1339mJ
Factorial	66mJ	108.4mJ	261.6mJ
Dhrystone2	81mJ	118mJ	268mJ
LiverMoreLoop2	550mJ	970mJ	2395mJ
MemSpeedi	338j	561mJ	1318mJ
RandMemi	339mJ	563mJ	1321mJ
NestedBranches	79mJ	114mJ	265mJ

5.4 Data Collection for Concurrent Program Execution Energy Overhead

This section empirically validates the overhead associated to SA-LEEF during concurrent program execution. Smart-phones are resource constrained devices due to their size and weight limitations. Resources of a smart-phone device such as CPU, cache, RAM, and other peripheral components are shared among the co-resident applications. Due to resource limitations, concurrent running applications on a smart-phone leads to poor performance in terms of application extended execution time and energy consumption. One of the reasons of application performance degradation is the cache eviction during context switching of tasks. In this section, the energy overhead due to cache eviction during concurrent program execution is briefly discussed.

To highlight the impact of concurrent program execution on the total execution time of an application, it has executed several instances of the same program on the smart-phone device. For the experiments, it has considered SA-LEEF as test program for execution time analysis. It considered "time" Linux shell command to collect the execution time of test programs on smart-phone. It has considered user and system time generated by time Linux command to estimate total execution time of SA-LEEF program. It was ensured that all unnecessary background applications

University of Malaya

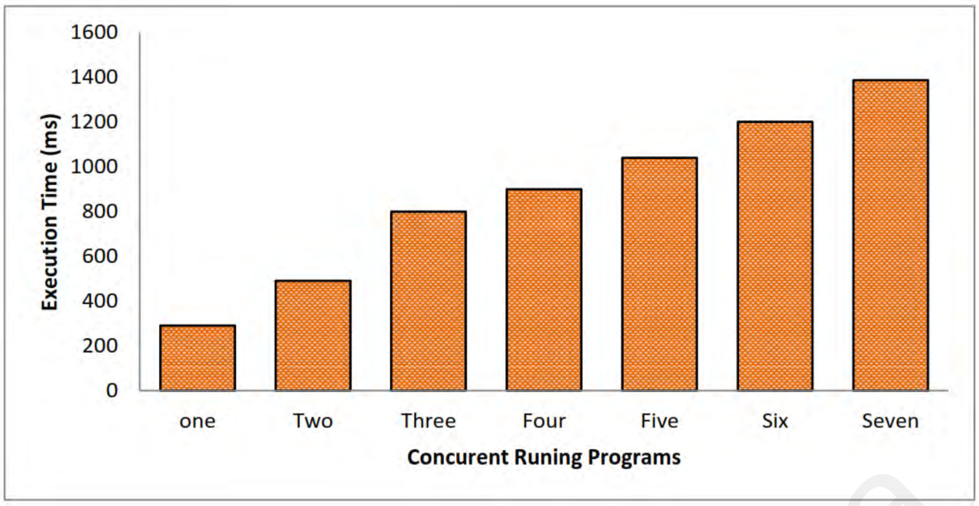

Figure 5.9: Effect of Concurrent Programs Execution to Total Execution Time of a Program

are turned off during experiments.

Fig. 5.9 highlights increase in total execution time of an application when multiple programs are concurrently running on a smart-phone device. As depicted in Fig. 5.9, the total execution time is highest when seven programs are executed concurrently on smart-phone. The increase in execution time is because of CPU sharing and cache eviction due to context switching of programs by the OS. It was noticed that during programs execution CPU usage increases up to 100% for seven concurrent programs execution case. Cache eviction is another factor impacting the total execution time when concurrently executing a set of programs on the same hardware. The execution time increases by a factor of 1.068 when two programs are executing in parallel on smart-phone. It increases by 1.063, 1.121, 1.155, 1.159, and 1.174 factors when three, four, five, six, and seven programs are executing in parallel. Due to multiple cores of the processor under observation, the increase in execution time when considering five, six, and seven programs is marginal as mentioned above.

Cache eviction leads to the extended execution time of a program if the contents of the concurrently executing programs frequently update system cache. During ap-

University of Malaya

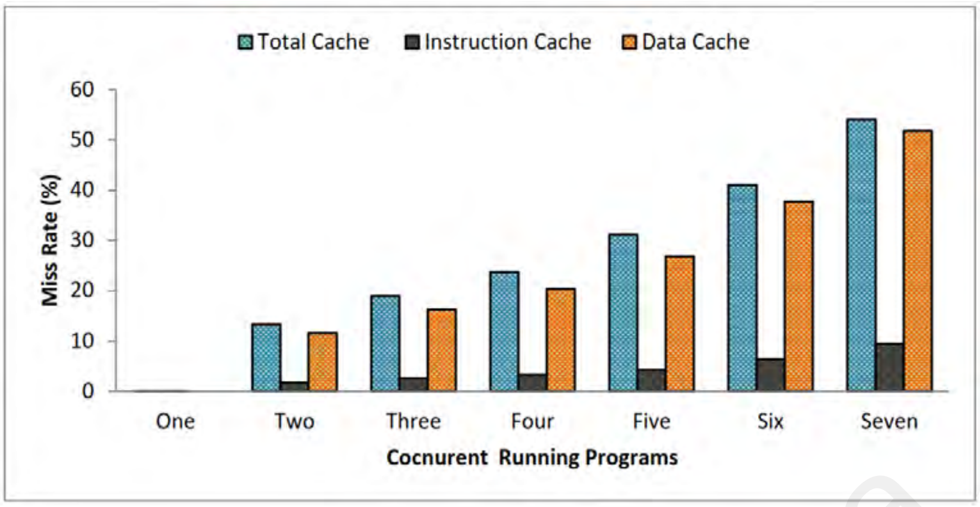

Figure 5.10: Effect of Concurrent Programs Execution on Cache Miss Rate

plication execution, CPU either notifies cache hit or a miss for instructions depending on the execution behavior of a program. To analyze the cache miss/hit behavior, this experiment ran several instances of a single program to see cache hit/miss behavior. It chose ARM 7 based server to analyze the performance of cache using the Val-grind tool. Fig. 5.10 demonstrates cache hit/miss behavior when SA-LEEF is running with its clone tenants. It is noticed that when only SA-LEEF was running on smart-phone device, instruction cache miss rate is noticed 0.08% only. For remaining cases, due to high context switching, the cache is evicted again and again causing data and instructions to be fetched from the RAM in the subsequent round robin slots. For the concurrent execution of seven clones of SA-LEEF, cache miss rate is noticed surging to 52%. In Fig. 5.10, total cache parameter demonstrates total cache misses in terms of data and instruction cache for all the levels of cache in the memory hierarchy.

Energy consumption during data/instruction eviction is estimated using external physical measurements based method. For the experiments, a test program is designed that frequently updates memory references. Due to continuously operating on the updated data the cache eviction rate was noticed very high. It has

University of Malaya

validated the behavior of test program on ARM server using the Val-grind tool. The proposed study has used the cache-grind option of Val-grind for cache analysis validation. During execution of test programs, the voltage drop across the resistor is profiled at the physical server. During the offline analysis phase, the average power consumption for the test program is calculated based on Eq. 5.2. To find the total energy consumed during cache eviction process, the average power consumption value is multiplied with the cache eviction time as highlighted in Eq. 5.7. The average power consumption during cache eviction test program execution is estimated 153.2mW. The overhead of concurrent program execution on a smart-phone device depends on the rise in execution time and base power consumption due to cache eviction as shown in Eq. 5.7.

$$Base-Power_{cache-eviction} = Avg.Power_{Consumption} \times Cache_{Evict-time} \tag{5.7}$$

5.5 Data Collection for Application Analyzer

This section empirically validates the Application analyzer module of SA-LEEF framework. It discusses the method to collect data to classify ARM assembly instructions for the target smart-phone benchmark applications. It evaluates the application analyzer module based on a set of benchmarks. In total, the application analyzer module profiles 69 ARM-IS instructions for the input smart-phone application. However, in this section, a subset of total profiled instructions which were common in the majority of benchmark applications are reported.

Application analyzer module is coded in C language. The input to application analyzer module is smart-phone application translated into ARM assembly code and pre-defined data set for variables inside the application. It trained application in offline mode against all possible execution scenarios to acquire the possible values

University of Malaya

Table 5.7: Total Assembly Instructions Count for Benchmark Applications

Benchmark	Assembly Instruction Count
LinpackSP2	1.07491×10^{10}
FFT1	7.98920×10^{10}
Factorial	5.66507×10^{09}
Dhrystone2	1.88835×10^{09}
LivermoreLoops2	1.88837×10^{10}
MemSpeedi	2.17987×10^{10}
RandMemi	2.32413×10^{10}
Nested Branches	1.13301×10^{10}
NativeWhetStone2	2.32413×10^{10}

for each data point whose value is not known. Application analyzer scans the application and classifies the instructions into memory based and cache based categories. Table 5.7 highlights count of assembly instructions for each benchmark application. The reported total number of instructions are the sum of 69 instructions profiled by the application analyzer module. It highlights aggregate sum of instructions accessed from the RAM storage and cache memory. As can be seen from Table 5.7, the total number of instructions for each benchmark is different depending on the number of operations a benchmark application performs. For instance, among all chosen benchmark applications, total instruction count for FFT1 benchmark is highest as it performs computationally complex operations while transforming data from one domain to another. The number of assembly based instructions for each benchmark depends on the type and number of operations for each benchmark application.

The instructions of an application are fetched from RAM or cache during its execution. The storage location analyzer module of application profiler (SA-LEEF framework) estimates storage location of instructions within a benchmark application. Table 5.8 demonstrates a set of assembly instructions for ARM-IS that the proposed storage location analyzer module has predicted to be fetched from the cache at run time. In Table 5.8, only those instructions which were common in the majority of benchmark applications are presented. The number and type of

University of Malaya

Table 5.8: Instructions Count for Benchmark Applications (cache access)

Inst	NWS	LSP	FFT1	FACT	DHY	LIV	MEM	RND	NB
#FADD	652811	0	0	0	0	0	0	0	22451
#FSUB	36472	22905	14740	0	0	0	0	0	7931
#MOV	1026168	2190532	15606169	456209	732689	29930763	529565	9837147	9087086
#LDR	998034	990364	80878156	399992	298028	1490564	152939	1041287	1060364
#STR	360705	498018	20916764	199996	2490091	7646292	90182	3689628	510182
#BL	450617	45600	109440	99998	22800	687923	3882	3948	58018
#FMUL	28803	345629	31106013	0	0	0	1209	0	3944
#ADD	208939	234009	1684865	99988	37657	171665	420653	18953	328675
#SUB	2500	4201415	15345042	99997	2051	3244625	4207	38009	44942
#MUL	410742	253746	8214984	99998	114097	2167843	64259	0	34942
#CMP	450617	399230	1047305	99998	18701	486438	86247	349832	196972
#FDIV	0	0	20285	0	0	0	0	0	9923
#DIV	55794	0	1334	3	0	0	0	0	9424
#ORR	0	50941	0	0	4	0	0	10341	0
#SMULL	0	70982	0	0	0	2283	389129	0	0
#RSB	50397	14519	0	0	0	34819	78	367	77644
#LSL	20289	158582	3452713	0	0	1585602	78192	90723	0

ARM-IS instructions in each benchmark depend on the functionality of that particular benchmark. For instance, LinpackSP2 has performed addition operation on single and double type data; that is why ADD instruction count is very high for LinpackSP2 benchmark. The application analyzer is ran five times but the results revealed that the estimation behaviour remained unchanged as there is no randomization involved in profiling process. Table 5.9 demonstrates a set of ARM-IS instructions which are common in the majority of benchmark applications. Table 5.9 discusses RAM storage based instructions. The number and type of instructions in ARM-IS for each benchmark depend on the functionality of that benchmark. For instance, LivermoreLoops2 has performed addition operation on single and double type data; that is why this value is noticed high for this benchmark application.

Table 5.8 and Table 5.9 demonstrates count of instructions within each benchmark application tested based on static analysis method. Based on the cache and RAM storage analysis of the smart-phone application, instructions are classified into two categories. In the first category as shown in Table 5.8, the highlighted instructions are those which are accessed from local system cache during application execution on smart-phone. On the other hand, in the second category as shown in

University of Malaya

Table 5.9: Instructions Count for Benchmark Applications (RAM Access)

Inst	NWS	LSP	FFT1	FACT	DHY	LIV	MEM	RND	NB
#FADD	00065	00000	00000	0	0	0	0	0	00019
#FSUB	00044	00005	00007	0	0	00000	00000	0	00011
#MOV	00670	00445	01257	00035	00081	05087	00492	00631	00486
#LDR	00278	00811	00587	00022	00036	01910	00344	00269	00199
#STR	00181	00220	00328	00011	00017	00612	131	113	102
#BL	00069	00184	00319	0005	00016	00599	116	88	99
#FMUL	00038	00051	00018	00000	00000	0	4	0	12
#ADD	00089	00361	00267	00008	00059	02138	00196	100	135
#SUB	00025	00090	00228	00005	00036	000294	0008	00014	00076
#MUL	00033	00256	00190	00004	00043	00370	00018	00000	00043
#CMP	00034	00077	00084	00004	00019	00241	00070	00025	00270
#FDIV	00000	00000	00005	00000	00000	00000	00000	0	00013
#DIV	00011	00000	00007	00005	00000	00000	00000	00000	00009
#ORR	00000	0022	00000	00000	00004	00000	00000	00016	00000
#SMULL	00000	0012	00000	00000	00000	00003	02515	00000	00000
#RSB	00015	0033	00000	00000	00000	00035	00001	00002	00071
#LSL	00052	00301	00189	00000	00000	00657	00090	00136	00000

Table 5.9, instructions which are fetched from the RAM storage during execution on smart-phone are shown. For Factorial benchmark application, SA-LEEF estimated that when considering SUB_{R_i,R_i,R_j} instruction, in total 99998 instances of this instruction are fetched from the local system cache during application execution on smart-phone. However, only 3 instances of SUB_{R_i,R_i,R_j} instruction are fetched from the RAM storage for execution on smart-phone.

Application analyzer is the most resource consuming module of SA-LEEF after ARM instruction energy profiler module. Table 5.10 and Table 5.11 highlights overhead of application analyzer module of SA-LEEF framework. The estimation time overhead of application analyzer highly depends on the total assembly size of the application. Total estimation time of the application analyzer module consists of, (a) execution path estimation, (b) loop estimation, (c) storage location analysis, and (d) classification of instructions within the application. It is noticed that application analyzer has consumed 1.94s, 2.15s, 1.95s, and 0.18s for NativeWhetstone2, LinpackSP2, FFT1, and Factorial benchmark applications. Similarly, it has consumed

University of Malaya

Table 5.10: Estimation Time Analysis for Application Analyzer Module

Benchmark	Avg.Time (s)	St. Dev	Conf. Interval	Population Range
NWS	1.94	0.00894	±0.01	1.94±0.01
LPS	2.15	0.0114	±0.01	2.15±0.01
FFT1	1.95	0.0129	±0.01	1.95±0.01
FACT	0.18	0.00548	±0.00	1.95±0.0
DHY	1.06	0.00548	±0.01	1.95±0.01
LIV	2.57	0.0182	±0.02	2.57±0.02
MEM	2.04	0.0083	±0.01	2.04±0.01
RND	1.88	0.0072	±0.01	1.88±0.01
NB	0.45	0.0044	±0.00	0.45±0.00

1.06s, 2.57s, 2.04s, 1.88s, and 0.45s, while classifying instructions within Dhrystone2, LivermoreLoops2, MemSpeedi, RandMemi, and Nested Branches benchmarks, respectively. For all chosen benchmark applications, the estimation time overhead is analyzed for ten runs. The descriptive statistics is applied to find the confidence interval for 95% percentile interval. It is noticed that the error in the estimation readings for different runs was noticed very small. For instance, application analyzer has reported estimation time in the range of 1.94 ± 0.01 for ten runs for NativeWhetstone2 benchmark application. Similarly, for LivermoreLoops2 and Nested Branches benchmark application, the estimation time is observed in the range of 2.57 ± 0.02 and 0.45 ± 0.00, respectively.

Table 5.11 highlights energy consumption overhead of application analyzer module of SA-LEEF framework. The energy estimation overhead is estimated based on the product of application analyzer estimation time and its average base power consumption. The average base power consumption is estimated using external physical measurement setup to log voltage and current drop during application analyzer execution on smart-phone. It ran application analyzer in nested loops while classifying instructions for FFT1 to record sufficient voltage and current drop readings to estimate average power consumption of application analyzer module. The proposed study ran application analyzer ten times while analyzing FFT1 to find the

University of Malaya

Table 5.11: Energy Overhead Analysis for Application Analyzer Module

Benchmark	Avg. Energy (mJ)	St. Dev	Conf. Interval	Population Range
NWS	56	0.2500	±0.3104	56±0.3104
LPS	62	0.5828	±0.7235	62±0.7235
FFT1	56	0.4000	±0.4000	56±0.4000
FACT	5	0.0000	±0.0000	5±0.0000
DHY	30	0.0000	±0.0000	30±0.0000
LIV	74	0.6752	±0.8383	74±0.8383
MEM	59	0.5000	±0.6208	59±0.6208
RND	54	0.1732	±0.2150	54±0.2150
NB	13	0.0000	±0.0000	13±0.0000

University of Malaya

average of ten runs to find its base power consumption. The energy overhead describes the amount of energy in mili-joule the application analyzer consumes while classifying instructions of benchmark application based on their execution paths, loop bounds, and storage location analysis. It was noticed that application analyzer module has consumed dissimilar energy for chosen benchmark applications. For instance, application analyzer has consumed 56mJ, 62mJ, 57mJ, and 5mJ energy while classifying instructions of NativeWhetstone2, LinpackSP2, FFT1, and Factorial benchmark applications. Similarly, it has consumed 30mJ, 74mJ, 59mJ, 54mJ, and 13mJ energy on average, while classifying instructions within Dhrystone2, LivermoreLoops2, MemSpeedi, RandMemi, and Nested Branches benchmarks, respectively. The estimation population range based on descriptive statistics for ten runs is estimated 56 ± 0.3104, 62 ± 0.7235, 56 ± 0.4000, 5 ± 0.0000, and 54 ± 0.2150 for NativeWhetstone2, LinpackSP2, FFT1, Factorial, and RandMemi benchmark applications, receptively. The results and validation of energy estimation module is discussed in chapter 6.

5.6 Base Power Consumption Computing for SA-LEEF Framework

This section discusses data collection method for base power consumption estimation for SA-LEEF energy estimation tool. Base power consumption represents the average power consumed by SA-LEEF during energy estimation of smart-phone applications.

This study has computed base power consumption of SA-LEEF using external physical measurement method as highlighted in Fig. 5.3. Base power consumption is estimated based on the product of voltage and current drop during SA-LEEF execution on smart-phone device. Data for voltage and current drop is collected using EM6000 multi-meter tool. It was observed that the average power consumption during SA-LEEF execution on smart-phone for energy estimation of different benchmark applications remained same. The complexity of smart-phone applications varies depending on the structure and operations in it. SA-LEEF proposes same operations for energy estimation of smart-phone applications irrespective of their complexities. As a result, the average base power consumption of SA-LEEF is almost same for all type of benchmark applications.

Six benchmark applications including NativeWhetstone2, LinpacksSP2, Factorial,FFT1, LivermoreLoop2, and MemSpeedi, are selected to collect data for base power consumption estimation of SA-LEEF framework. The estimation time of SA-LEEF framework while estimating energy consumption of here-mentioned benchmarks was observed very limited. As a result, only a few timestamped voltage readings were captured at the physical server. To handle this issue, the size of chosen benchmark applications was increased by replicating its code 20 times to increase energy estimation time of SA-LEEF. As a result, sufficient voltage and current drop readings were collected at the physical server for offline analysis to estimate base power consumption of SA-LEEF framework. Each experiment is repeated 10 times to report average base power consumption of SA-LEEF framework for benchmark applications.

It is observed that average power consumption during energy consumption esti-

University of Malaya

mation for different benchmark applications is same. The average power consumption of 10 runs for NativeWhetstone2, LinpacksSP2, Livermoreloops2, FFT1, Factorial, and MemSpeedi, is noticed 28.9mW, 28.8mW, 28.6mW, 28.8mW, 28.9mW, and 28.7, respectively. The mean of base power consumption for aforementioned benchmark applications is estimated 28.78mW. Alternatively, standard deviation from the mean is noticed 0.1169. For 99% percentile, the population range for the base power consumption of SA-LEEF is observed 28.78 ± 0.265.

5.7 Conclusion

In this chapter, SA-LEEF framework is evaluated. It has discussed data collection tools, methods, devices, and experimental setup to evaluate SA-LEFF. Based on the collected data, application analyzer, application energy estimator, and collaborator modules of SA-LEEF framework are evaluated.

It has collected energy consumption data for ARM-IS for both cache and memory-based operations. It was noticed that energy consumption of memory-based operations is more expensive than cache operations. It has analyzed application analyzer module of SA-LEEF to count the total number of instructions in ARM assembly based benchmark application and to classify them into cache and memory based categories. It was observed that the number of instructions fetched from the cache are much higher than those accessed from the main memory. Also, the overhead of application analyzer in terms of energy consumption and estimation time is noticed very minimal. It has also collected data for the energy cost of accessing benchmark applications from both local and remote servers using Wi-Fi and 3G radios. It was observed that 3G is more energy consuming than a Wi-Fi module. Also, it is observed that SA-LEEF has consumed highest energy while accessing ARM-IS energy profile and assembly code of LivermoreLoops2 benchmark.

University of Malaya

Data is collected to estimate average base power consumption of SA-LEEF framework. The average base power consumption of SA-LEEF is reported 28.78mW. To present the real application execution environments on a smart-phone, the overhead of concurrent program execution in terms of cache eviction is estimated. For cache eviction when two applications are executing concurrently, the average base power consumption is reported 153.2mW.

University of Malaya

CHAPTER 6: RESULTS AND DISCUSSION

This chapter validates system model of SA-LEEF framework against its empirical evaluation results. It compares the performance of SA-LEEF framework against Power Tutor and Measurement-based energy estimation methods. It considers energy estimation time, energy overhead, estimation accuracy, and resource consumption as performance measurement parameters for the comparisons. It considers a set of benchmark applications to analyze the behavior of SA-LEEF in comparison to other estimation methods for aforementioned parameters.

This chapter is organized into six main sections. Section 6.1 validates SA-LEEF system model by comparing it against empirical evaluation. Section 6.2 compares SA-LEEF to existing energy estimation methods for its local operation mode case. Section 6.3 compares SA-LEEF to existing energy estimation methods for its remote operational mode case. Section 6.4 compares SA-LEEF to existing energy estimation tools based on CPU and RAM resource consumption behavior. Section 6.5 presents qualitative comparison of SA-LEEF to existing code analysis based energy estimation methods. Section 6.6 concludes the whole discussion and presents the main findings of this chapter.

University of Malaya

6.1 SA-LEEF Framework Validation

This section validates system model for SA-LEEF framework discussed in Section 4.2.2 by comparing its results with results of empirical evaluation. It has validated system model for SA-LEEF framework using code review validation method based on energy consumption estimation of benchmark applications as discussed below.

6.1.1 Energy Consumption Estimation

SA-LEEF estimates energy consumption of a smart-phone application using ARM-IS energy profile. Based on SA-LEEF estimation model, total energy of an application consist of, (a) ARM ISA access cost from the server, (b) base cost energy of application, (c) user system interaction cost, and (d) energy overhead associated with concurrent program execution on resource constrained smart-phone devices. In the following section, the energy consumption of NativeWhetstone2 benchmark application using SA-LEEF system model is derived and presented.

SA-LEEF estimated that NativeWhetstone2 has consumed $7.9j$ energy when it is executed on Google Nexus One smart-phone device as highlighted in Eq. 6.1. The major portion of NativeWhetstone2's energy is consumed by the source code instructions (base cost) of the benchmark application. It is noticed that $7.6j$ of the total energy budget is consumed by the base cost energy element of SA-LEEF system model. Alternatively, concurrent program execution and ARM-ISA-Access processes consumed $48mJ$ and $257.08mJ$ energy, respectively. NativeWhetstone2 is a compute-intensive benchmark, and it does not require any input from the user during its execution. Therefore, user system interaction energy cost is $0J$ in this case.

$$Total_{Energy} = 7.6j + 0j + 48mJ + 257.08mJ \approx 7.905j \tag{6.1}$$

The total base cost energy is estimated by statically analyzing ARM assembly code of NativeWhetstone2 benchmark application. The built-in system library routines which are called from within the code and user defined routines contributes to the total base cost energy of application. For NativeWhetstone2 benchmark application, in total, $1.334J$ energy is consumed by the built-in system library routines; whereas,

University of Malaya

user defined code has consumed $7.01j$ energy as presented in Eq. 6.2.

$$E_{BaseCostEnergy} = 1.334J + 7.01j \approx 7.6j \tag{6.2}$$

A call to a built-in system library routine executes a set of instructions to perform the required task. As a result, it consumes a significant amount of energy. Eq. 6.3 highlights the energy consumption for a set of system library routines called from within NativeWhetstone2's source code (assembly based). In the NativeWhetstone2 benchmark application, overall 39 calls were made to printf routine from within benchmark application to generate outputs. Alternatively, for a few other library routines such as sqrt, cos, and sin, number of calls are profiled 55800, 19200, and 19200, respectively.

$$E_{Systm-Libraries} = (39 \times 2.7 + 55800 \times 1.0 + 19200 \times 2.0 + 19200 \times 1.9) \times 10^{-5} \approx 1.3J \tag{6.3}$$

NativeWhetstone2 benchmark consists of six user defined modules including Main, Whetstones, Pa, Po, P3, and Pout. Each module of SA-LEEF executes for k times. Among all user defined modules, main module executes only once. The remaining modules execute for 80, 539400, 246400, 02, and 8400, number of times to perform the required task as shown in Eq. 6.4. Each module of the application has consumed dissimilar energy depending on the number and type of operations it involves. For each module of the NativeWhetstone2 benchmark application, total energy consumption is estimated using, (a) energy cost of instructions residing outside of the branching constructs, (b) statements executing within loops, and (c) statements whose total energy consumption depends on the branch predictors

University of Malaya

(weighted probability). In Eq. 6.5, the energy consumption of one of NativeWhetstone2 module (5^{th} in Eq. 6.4) is presented. It represents energy cost associated with loops structures, branching statements, and the sequential code. In the said equation, $200mj$ is the energy cost associated with the code running in sequential flow outside the branching and loop constructs. Moreover, for loop constructs, the energy is divided into two categories (e.g., 1999.99mj and 0.002981) depending on the storage location of instructions within the loops at run-time.

$$\begin{aligned} E_{App-developer-modules} = (20mj \times 1 + 3mj \times 80 + (3.7078 \times 10^{-6})j \times 539400 \\ +(4.8701 \times 10^{-6})j \times 246400 + 0.000190j \times 8400 + 0.61j \times 2) \approx 6.27j \end{aligned} \tag{6.4}$$

$$E_{Module-A} = 200mj + (1999.99mj + 0.002981) + 0 \approx 2.2 \tag{6.5}$$

Eq. 6.6 calculates energy overhead associated with concurrent execution of SA-LEEF with its tenants for energy estimation of smart-phone application. The associated energy overhead is computed based on increase rate in execution time while running SA-LEEF with its tenants and average power consumption during cache eviction. To find the power consumption during cache eviction, the current study has designed a synthetic benchmark that evicted cache again and again to fetch data and instructions from RAM storage. The power consumption during cache eviction was recorded using external measurement based method. It was found that average power consumption during cache eviction process was $153.2mW$. This study has used Val-grind tool to verify the effect of concurrent program execution on the cache eviction rate. It was noticed that cache was the dominant factor that increased the execution time while running programs concurrently. While executing the same

University of Malaya

process concurrently, the execution time of a program is surged by a factor of $0.31s$. The concurrent execution energy overhead for the proposed system is found $48mj$.

$$E_{Concurrent-Execution-Overhead} = (3.41s - 3.1s) \times 153.2mW \approx 48mj \tag{6.6}$$

Total execution time of an application is estimated based on the size of the program and speed of processor as shown in Eq. 6.7 and Eq. 6.8. The size of a program is estimated based on CPI for target ARM architecture and total instructions within benchmark application. Average CPI for ARM-7 ISA is 1.6 (Blem, Menon, Vijayaraghavan, & Sankaralingam, 2015). Therefore, in this research, 1.6 CPI for the execution time estimation is chosen. NativeWhetstone2 is statically analyzed to find total number of instructions in it.

$$Time_{Program} = \frac{3701400000Ins}{1194MIPS} \approx 3.1s \tag{6.7}$$

$$Program_{size} = 2313375000 \times 1.6 \approx 3701400000ins \tag{6.8}$$

$$Time_{ConcurrentExecution} = 0.40 \times 3.1s \approx 3.41s \tag{6.9}$$

Energy consumption of Wi-Fi module while downloading a file from remote server depends on the average power consumption and total activity time on Wi-Fi module as shown in Eq. 6.11. Activity time as shown in Eq. 6.10 is estimated based on the total size of the file that is downloaded and bandwidth of the network link. This study has opted speed test application to find the bandwidth of downlink for the considered Wi-Fi connection while selecting a server located in Singapore. On the

University of Malaya

other hand, network power consumption is estimated using Power Tutor energy estimation tool.

$$E_{ISA-Access} = 0.53s \times 485mW \approx 257.08mj \tag{6.10}$$

$$Activity_{time} = \frac{110 \times 1024bytes}{1.78Mbps} \approx 0.53s \tag{6.11}$$

6.1.2 SA-LEEF Estimation Accuracy

This section compares results of system model for SA-LEEF to the results of empirical evaluation. Three benchmark applications are used to validate system model of the proposed energy estimation framework. The same procedure is followed for LivermoreLoops2 and Nested Branches benchmarks as discussed in Section 6.1.1 to evaluate system model of SA-LEEF framework.

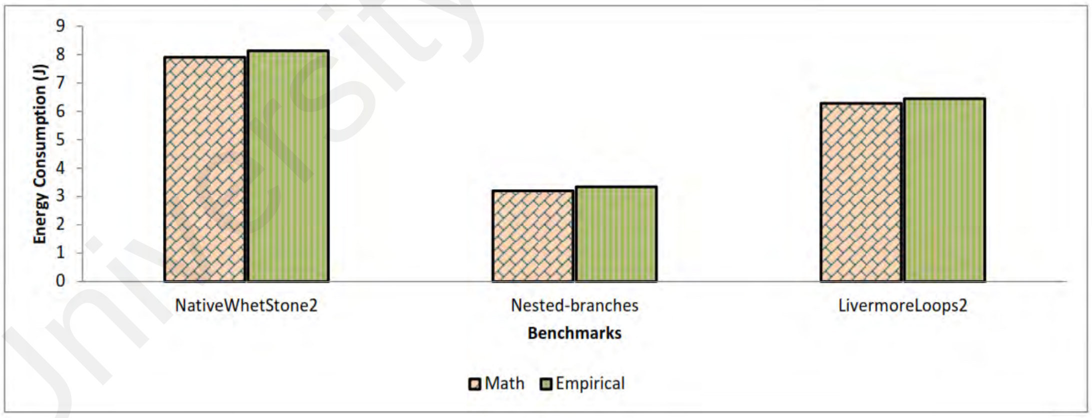

Figure 6.1: SA-LEEF System Model vs. Empirical Evaluation

Fig. 6.1 draws a comparison between energy consumption estimation for a set of benchmark applications using system model (mathematical model) and empirical evaluation of SA-LEEF framework. Two benchmark applications are selected, and one synthetic benchmark for comparison. Synthetic benchmark application composed of a set of nested branching statements. The results demonstrate that the

University of Malaya

system model is accurate up to 97.37% when compared with empirical evaluation results. The marginal error for system model is due to, variance in energy consumed by concurrent programs executing on smart-phone to the one considered in model, effect of power states of Wi-Fi component to the energy consumption, locality of ARM-IS energy profile hosting server, peak signal to noise ratio of Wi-Fi network connectivity, and the possibility to miss few instructions during application analysis. The experiments are repeated several times to examine the behavior. It was noticed that SA-LEEF model exhibits approximately similar behavior for multiple runs of SA-LEEF while examining benchmark applications for energy estimation. To find whether the correlation between results of system model and empirical evaluation is significant or not, the proposed study has applied Pearson variance method (Filina & Zubkov, 2008). The correlation between the data set is observed 0.99.

6.2 Performance Comparison of SA-LEEF Framework (Local Mode) to Existing Estimation Methods

This section compares the performance of SA-LEEF framework to existing energy estimation methods based on energy estimation time, energy estimation accuracy, and energy estimation overhead parameters. A set of benchmark applications including LinpackSP2, LivermoreLoops2, Dhrystone2, RandMemi, NativeWhetstone2, Nested branches, FFT1, and Factorial are selected. For ease of use, the current study has proposed and used abbreviation words for aforementioned benchmark applications in the rest of this chapter. For instance, it proposes MSP, LPS, LML, DHS, RNM, NWS, NSB, FFT, and FAC, abbreviation words to represent MemSpeedi, LinpackSP2, LivermoreLoop2, Dhrystone2, RandMemi, NativeWhetstone2, Nested Branches, FFT1, and Factorial benchmark, respectively. In this section, it is assumed that the ARM-IS energy profile and ARM-based assembly code is already available on the smart-phone. This study called this operational mode as local

University of Malaya

execution mode of SA-LEEF framework.

Power Tutor and Measurement-based method are selected for the comparisons. Power Tutor is a highly cited open access energy estimation tool. It is easily and freely available on Google play store. The measurement-based method is accurate as it uses high precision, rate, and accurate power measurement equipment for energy estimation of an application. In this study, measurement based method is selected for the comparisons to calculate the accuracy of SA-LEEF energy estimation tool. SA-LEEF considers estimation environment much similar to the one a dynamic analysis method uses. For instance, it models context switching overhead and user system interaction behavior during energy estimation of an application. Therefore, one main reason for choosing Power Tutor for comparison with SA-LEEF is the high resemblance in their execution environments.

6.2.1 Standard Benchmarks Code size Case

In this section, SA-LEEF and existing energy estimation methods are compared based on the standard code size of the chosen benchmark applications.

6.2.1.1 Energy Estimation Time

Fig. 6.2 demonstrates the total energy estimation time for a set of benchmark applications using SA-LEEF estimation framework. The X-axis in Fig. 6.2 represents selected android based benchmark applications; whereas, Y-axis demonstrates average energy estimation time in seconds (system time) for each benchmark application. Energy estimation time for each benchmark application highly depends on the size and complexity of assembly based operations within benchmark applications. As presented in Fig. 6.2, LivermoreLoops2 benchmark has exhibited highest energy estimation time among all selected benchmark applications. SA-LEEF took 2.96 seconds during energy estimation of the LivermoreLoops2 benchmark. On the

University of Malaya

other hand, the Factorial program took lowest estimation time when it is estimated using SA-LEEF energy estimation framework. For the Factorial program, whole energy estimation process of SA-LEEF was completed in only 0.21s. Energy estimation time for LinpackSP2 benchmark application was noticed 2.48s. Estimation time for Nested branches benchmark in comparison to other benchmark applications was noticed very limited (0.52s). Moreover, for NativeWhetstone2 and Dhrystone2 benchmarks, SA-LEEF reported 2.23s and 1.22s of estimation time, respectively. From the above reported data, it is noticed that SA-LEEF's energy estimation time is dissimilar for a set of chosen benchmark applications. For instance, the estimation time of SA-LEEF for the Factorial benchmark is 91% lower than the LinpackSP2 benchmark. The assembly code size of all benchmark applications except Factorial, Nested branches, and Dhrystone2, is noticed closer to each other due to the marginal difference in their total code size. The estimation time reported in Fig. 6.2 is the average of 15 runs of SA-LEEF for each benchmark application. The variation in estimation time for different runs for a benchmark application was noticed very nominal (Standard deviation of 0.003 for LinpackSP2). The main reason for this small variation in estimation time is the set of OS activities running in the background. The effect of background activities is high when an application runs for a large period of time. As SA-LEEF energy estimation time is very small; therefore, the effect of background activities is very small. The data is analyzed to find the correlation between estimation time for the chosen benchmark applications and found it satisfactory. The mean estimation time is noticed 1.754 seconds for all the chosen benchmark applications. The standard deviation from the mean is noticed 0.872. The 0.872 value of standard deviation is because of the fact that chosen benchmarks are non-similar in terms of their operations. A few of the benchmark applications offered very small estimation time compared to the rest of the bench-

University of Malaya

mark applications. The variance is also estimated from the standard deviation and was noticed 0.76 only.

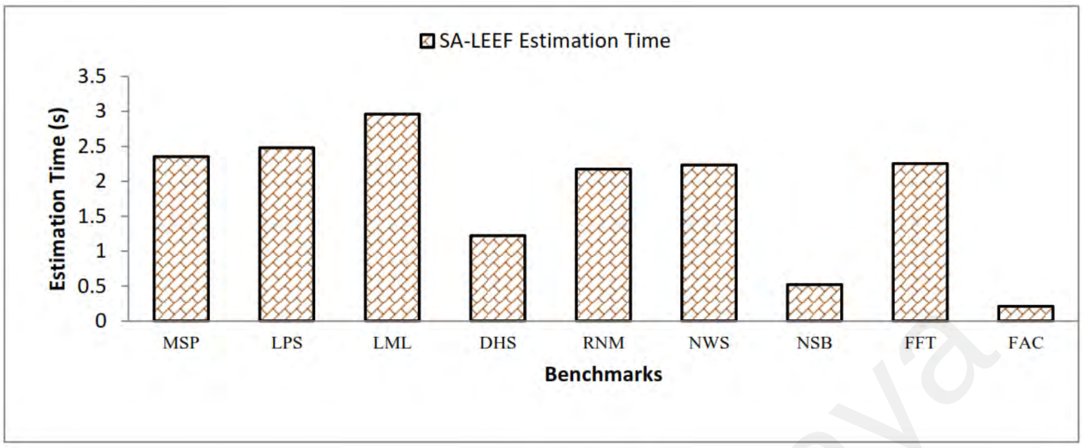

Figure 6.2: Energy Estimation Time Analysis for SA-LEEF Framework

Fig. 6.3 and Fig. 6.4 highlights energy estimation time for a set of benchmark applications using dynamic analysis based energy estimation methods. Power Tutor and measurement-based energy estimation methods are two dynamic analysis based energy estimation methods. In the aforementioned figures, X-axis presents the benchmark applications whereas Y-axis highlights the average total energy estimation time for each benchmark applications. For Power Tutor energy estimation method, among all the chosen benchmark applications, the average energy estimation time for FFT1 benchmark was noticed highest. Power Tutor estimated energy consumption of FFT1 benchmark in 55 seconds. Alternatively, for Dhrystone2, Power Tutor took 1.3 seconds only to estimate its energy consumption. For dynamic analysis based energy estimation, estimation time truly depends on the total execution time of the application. Fig. 6.3 depicts that Power Tutor took 15, 7.4, 12.2, and 16 seconds during energy estimation of MemSpeedi, LinpackSP2, LivermoreLoops2, and RandMemi benchmarks, respectively. For Nested branches, Factorial, and NativeWhetstone2 benchmarks, Power Tutor took 7.5, 3.9, and 14.5 seconds, to estimate their energy consumption. The standard deviation from the

University of Malaya

mean for energy estimation time of chosen benchmark applications is noticed 16. The high value of standard deviation from the mean is due to the noticeable difference among the estimation times of chosen benchmark applications. The variance from standard deviation for estimation by Power Tutor for chosen benchmark applications is noticed 256.2. Fig. 6.4 highlights total energy estimation time for android based benchmark applications using measurement-based estimation method. Using measurement-based energy estimation method, energy estimation time is noticed 15.4s, 7.6s, 12.5s, 16.4s, 3.9s, 55.11s, 7.8s, and 16.9s for MemSpeedi, LinpackSPs, LivermoreLoops2, RandMemi, Factorial, FFT1, Nested branches, and NativeWhetstone2 benchmarks, respectively. For both Power Tutor and measurement-based energy estimation methods, the energy estimation time presented in Fig. 6.3 and Fig. 6.4 is the average of 15 runs. The variance among estimation time readings is marginal as prior to experiments it has switched off all the unnecessary applications. Considering measurement-based energy estimation method, the standard deviation for selected benchmark applications is noticed 15.95. Also, the variance from standard deviation is 254.24 for measurement based method. The high value of standard deviation is due to high difference between estimation time of FFT1, Factorial, and Dhrystone2 benchmarks.

University of Malaya

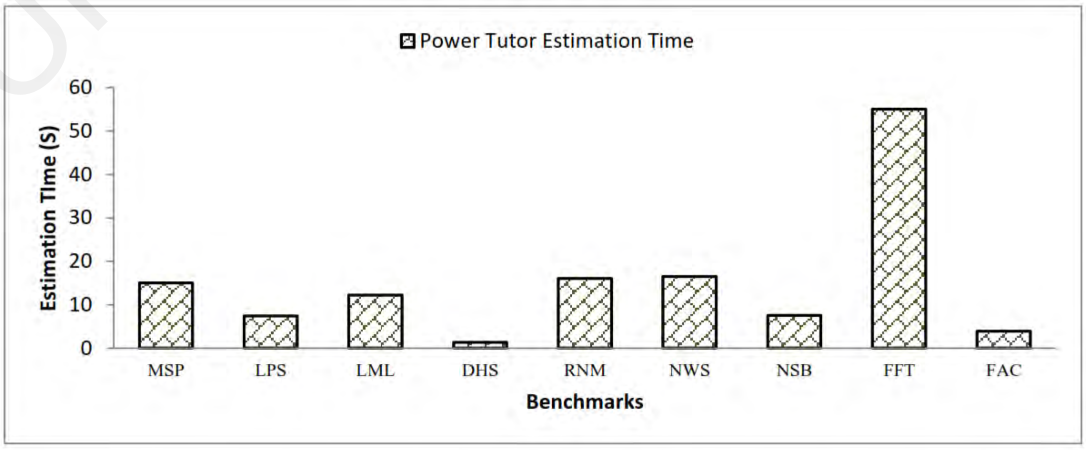

Figure 6.3: Energy Estimation Time Analysis for Power Tutor

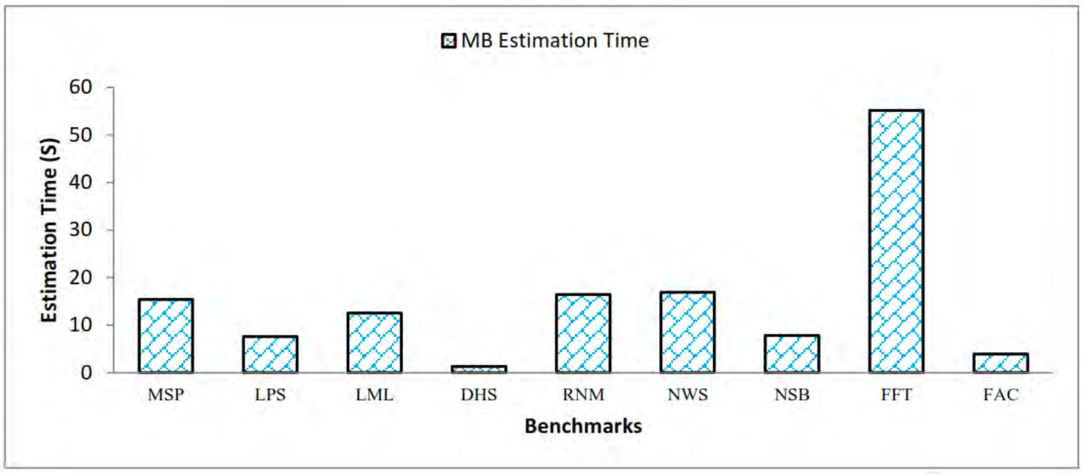

Figure 6.4: Energy Estimation Time Analysis for Measurement-based Estimation Method

Fig. 6.5 compares three energy estimation methods including SA-LEEF, Power Tutor, and measurement method, based on energy estimation time parameter for a set of benchmark applications. The X-axis highlights energy estimation tools and chosen benchmark applications. Y-axis states the average energy estimation time for the chosen benchmark applications. The energy estimation time of SA-LEEF is much lower than Power Tutor and measurement-based energy estimation methods due to its static analysis methodology to estimate energy consumption of benchmark applications. It is noticed that SA-LEEF energy estimation process is 16 times faster than Power Tutor energy estimation method when considering MemSpeedi benchmark application. For LinpackSP2 benchmark, SA-LEEF is 34 times faster than Power Tutor energy estimation method. Moreover, for LivermoreLoops2, RandMemi, NativeWhetstone2, Nested branches, FFT1, and Factorial benchmarks, SA-LEEF estimation time is lower than Power Tutor by a factor of 75%, 86%, 86%, 93%, 95%, and 94%, respectively. In the case of the measurement-based method, approximately same behavior is noticed when comparing it with SA-LEEF energy estimation framework. For instance, SA-LEEF estimation process is faster than measurement based method by a factor of 84.74%, 67.3%, 76.32%, 6.1%, 95.90%,

University of Malaya

and 94.6%, for MemSpeedi, LinpackSP2, LivermoreLoops2, Dhrystone2, Nested branches, and Factorial benchmark applications, respectively. The main reason for such huge difference in estimation time is due to the fact that loops took up to 98% (chapter 3) of the total execution time of an application. In case of SA-LEEF, the body of the loops is traversed only three times. Therefore, energy estimation time is noticeably minimized. From Fig. 6.5 it is evident that the estimation time of Power Tutor, measurement-based method, and SA-LEEF is approximately similar when the Dhrystone2 benchmark is analyzed. For Power Tutor and measurement based methods, the main reason of this minimal estimation time is the minimum iteration counts within loops of Dhyrstone2 benchmark. In the case of the Dhyrstone2 benchmark application, SA-LEEF is only 6 times faster than Power Tutor energy estimation method. While comparing measurement based method with the Power Tutor energy estimation tool, energy estimation time of Power Tutor is lower than measurement based method by a factor of 2-3.5% for all benchmark applications except Dhrystone2. The main reason for this marginal increase in estimation time of measurement-based method is the neighborhood operation to suppress the effect of noise due to background activities.

University of Malaya

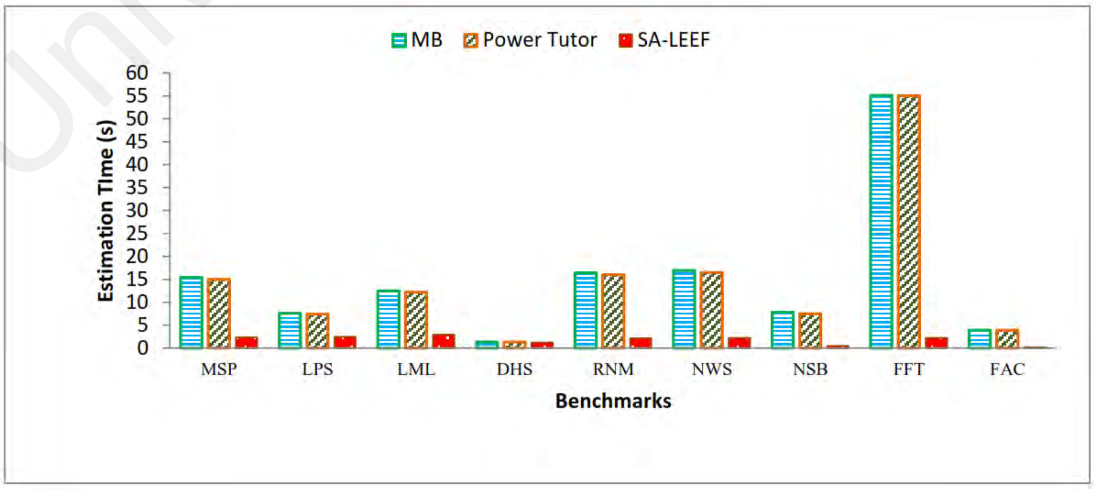

Figure 6.5: Comparison of Energy Estimation Methods for Energy Estimation Time

6.2.1.2 Energy Estimation Accuracy

Energy estimation accuracy of energy estimation methods highly depend on the accuracy of the power models used during energy estimation process. Fig. 6.6 demonstrates energy consumption for a set of android based benchmark applications when estimated using SA-LEEF framework. X-axis highlights a set of benchmark applications, whereas, Y-axis depicts average energy consumption for benchmark applications. SA-LEEF reported that MemSpeedi, LinpackSP2, LvermoreLoops2, Dhrystone2, RandMemi, NativeWhetstone2, Nested branches, FFT1, and Factorial benchmarks, has consumed 7.62 j, 3.48 j, 5.966 j, 0.6 j, 7.87 j, 8.02 j, 3.34 j, 34.99 j, and 1.85j energy, respectively. Among all the selected benchmark applications, energy consumption for Dhrystone2 benchmark was noticed the lowest. On the other hand, the value of energy estimation for FFT1 is noticed highest for all the chosen benchmark applications. From the energy consumption data for selected benchmark applications, the mean of the energy consumption is noticed 8.19. The standard deviation from the mean is noticed 10.4 as shown in Fig. 6.6. The main reason for this high value of standard deviation is the high variance among the nature of benchmark applications selected for the analysis. However, LinpackSP2 consists of several thousand numbers of instructions encapsulated in loops. As a result, estimation time and power consumption of LinpackSP2 is higher than Dhrystone2 benchmark application.

Fig. 6.7 highlights energy consumed by a set of android based benchmark applications estimated using Power Tutor energy estimation method. Power Tutor reported that MemSpeedi, LinpackSP2, LivermoreLoops2, Dhrystone2, RandMemi, NativeWhetstone2, Nested branches, FFT1, and Factorial benchmarks has consumed 8 j, 3.7 j, 6.3 j, 0.7 j, 8.2 j, 8.4 j, 3.75 j, 37.1 j, and 1.95j energy, re-

University of Malaya

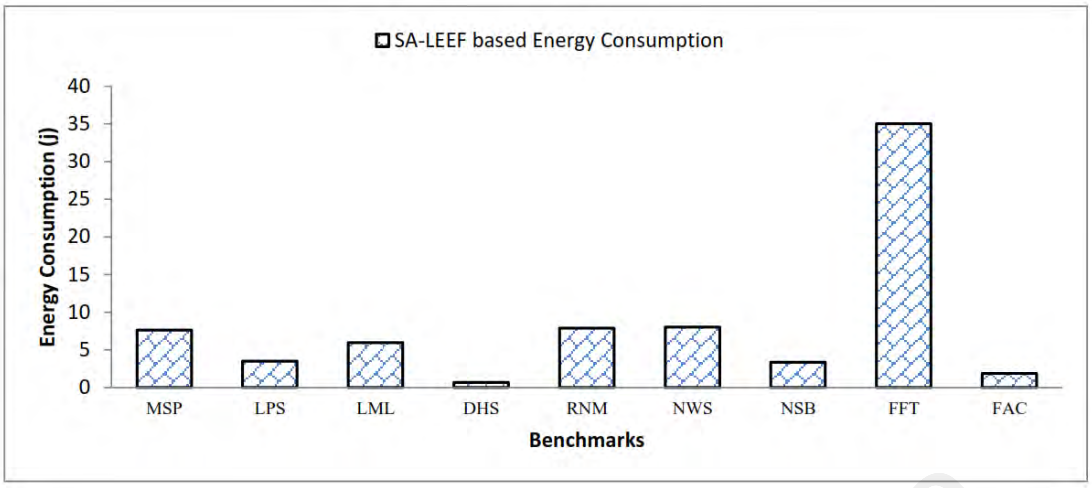

Figure 6.6: SA-LEEF based Application Energy Consumption Estimation

spectively. The standard deviation from the mean for chosen benchmark applications is noticed 11.02. However, the variance from the standard deviation is very high and it is reported 121.52. Fig. 6.8 demonstrates energy consumed by a set of android based benchmark applications estimated using measurement-based method. Measurement-based method reported that MemSpeedi, LinpackSP2, LivermoreLoops2, Dhrystone2, RandMemi, NativeWhetstone2, Nested branches, FFT1, and Factorial benchmarks has consumed 8.64 j, 3.95 j, 6.77 j, 0.75 j, 8.5 j, 9.01 j, 4.01 j, 39.77 j, and 2.08j energy, respectively. The standard deviation from the mean is reported 11.81. The main reason of high value of standard deviation from the mean is the high divergence (139.67) among the execution time statistics of selected benchmark applications.

Fig. 6.9 compares SA-LEEF, Power Tutor, and measurement-based energy estimation methods on the basis of collected data for energy consumption for a set of benchmark applications. It is shown in Fig. 6.9 that measurement-based method has highlighted highest energy consumption readings for chosen benchmark applications. Alternatively, SA-LEEF has reported lowest energy consumption readings for selected benchmark applications. Fig. 6.10 highlights the estimation accuracy of Power Tutor and SA-LEEF framework while considering findings of measure-

University of Malaya

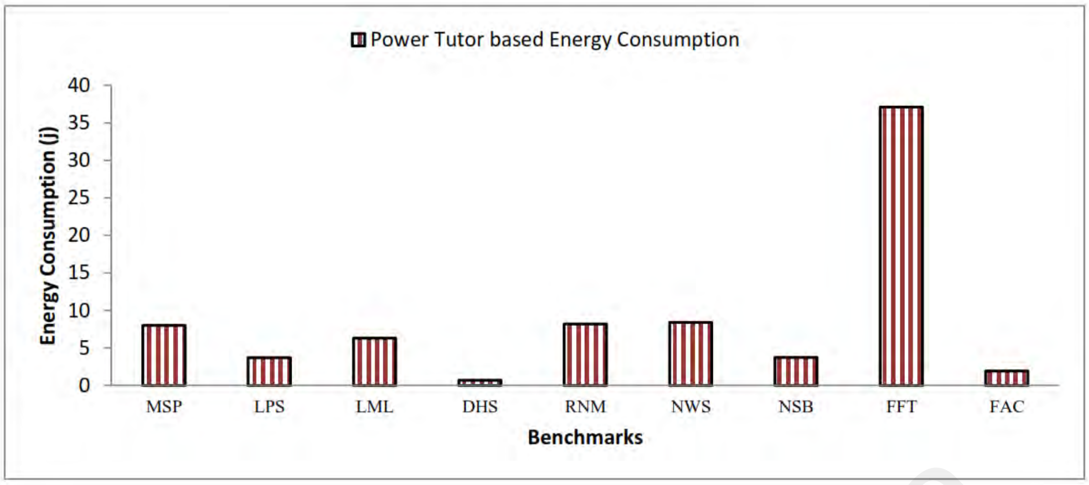

Figure 6.7: Power Tutor based Application Energy Consumption Estimation

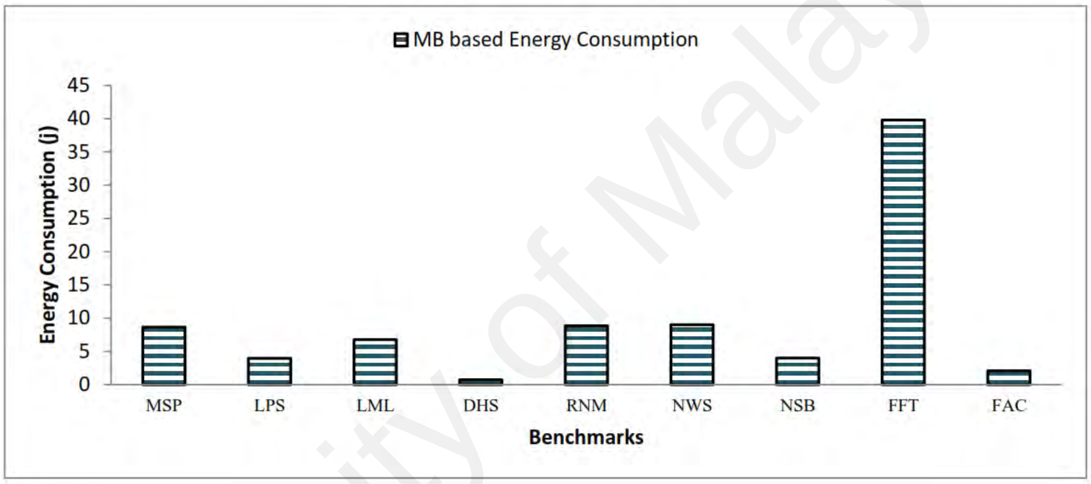

Figure 6.8: Measurement-based Application Energy Consumption Estimation

ment based method as ground truth value. The estimation accuracy of Power Tutor benchmark ranges from 92 to 93%. For instance, for MemSpeedi, LinpackSP2, LivermoreLoops2, and FFT1, the energy estimation is noticed 92.5 %, 93.4 %, 93.02 %, and 93.28 % accurate (for Power Tutor estimation method). The estimation accuracy of SA-LEEF framework is lower than Power Tutor. For instance, for MemSpeedi, LinpackSP2, LivermoreLoops2, and FFT1, the estimation is noticed 88.3 %, 88 %, 87.3 %, and 88 % accurate. For Nested branches benchmark application, the estimation accuracy of SA-LEEF is 82% accurate due to large number of Nested branches. The main reason for this behavior is the high chances of the errors while estimating the location of instructions, execution path estimation, the power states

University of Malaya

of smart-phone components, and loops termination. Power Tutor estimates energy consumption by physically running application on smart-phone; therefore, Power Tutor results are comparatively better. Estimation accuracy of Power Tutor is 5-6.3% higher than SA-LEEF energy estimation framework. The error in results of Power Tutor is due to errors in power models for the smart-phone components. The standard deviation from the mean for the accuracy statistics of the Power Tutor for chosen benchmark applications is observed 0.32. On the other hand, for SA-LEEF framework, the value of standard deviation from the mean for accuracy statistics is noticed 1.76.

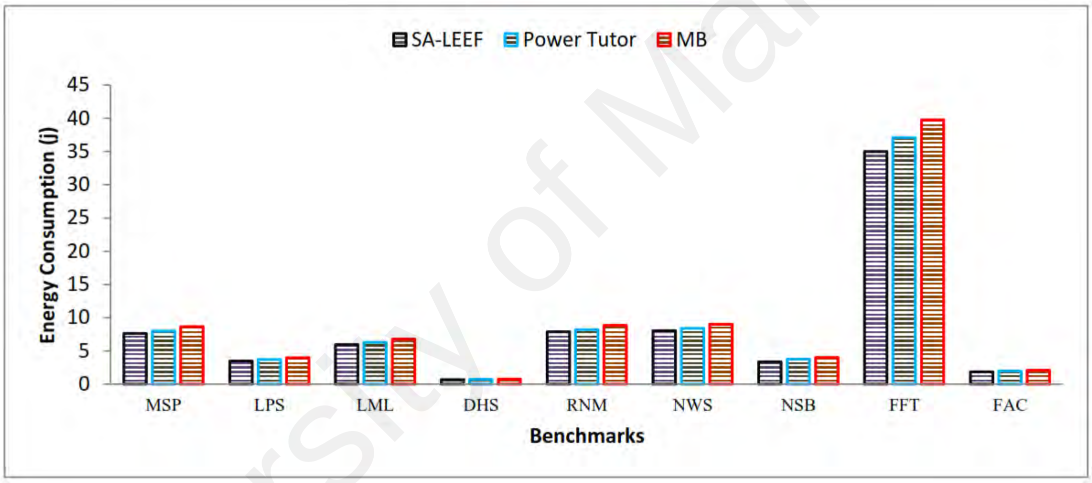

Figure 6.9: Comparison of Energy Estimation Methods based on Energy Consumption Estimation

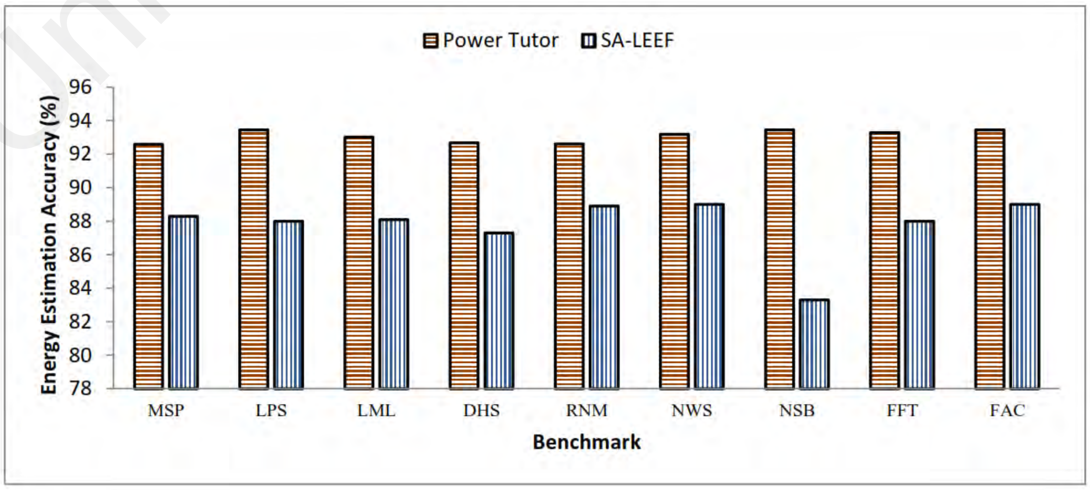

Figure 6.10: Comparison of Energy Estimation Methods based on Energy Estimation Accuracy

University of Malaya

6.2.1.3 Energy Estimation Overhead

The energy estimation overhead of an estimation tool states the total energy it requires to estimate energy consumption of the smart-phone application. Fig. 6.11 demonstrates energy estimation overhead for SA-LEEF estimation framework in milli joules for a set of benchmark applications. X-axis represents the selected benchmark applications whereas Y-axis highlights the average energy overhead of SA-LEEF benchmark. Energy estimation overhead composed of energy consumed during scanning target smart-phone application to estimate loops, execution paths, and instruction storage analysis. Among all the selected benchmark applications, during energy estimation of the Factorial program, SA-LEEF has consumed minimum energy. For LivermoreLoops2 benchmark application, SA-LEEF framework has exhibited highest energy overhead for selected benchmark applications. Statistically, the energy estimation overhead of SA-LEEF estimation tool during energy estimation of LivermoreLoops2 benchmark application is noticed 92.9 times higher than the Factorial program. During energy estimation of MemSpeedi, LinpackSP2, LivermoreLoops2, and Dhrystone2, SA-LEEF has consumed 67.91 mJ, 71.67 mJ, 85.54mJ, and 13mJ energy, respectively. Alternatively, for RandMemi, NativeWhetstone2, Nested branches, FFT1, and Factorial, SA-LEEF has consumed 62.71mJ, 64.44mJ, 15.02mJ, 65.02mJ, and 6.09mJ energy, respectively. The standard deviation from the mean for energy overhead of SA-LEEF for chosen benchmark applications is noticed 50.15. The high value of standard deviation is due to the variance in code size of chosen benchmark applications. The variance from the standard deviation is noticed 896.57.

Fig. 6.12 demonstrates energy estimation overhead for Power Tutor energy estimation tool during energy consumption estimation for a set of benchmark appli-

University of Malaya

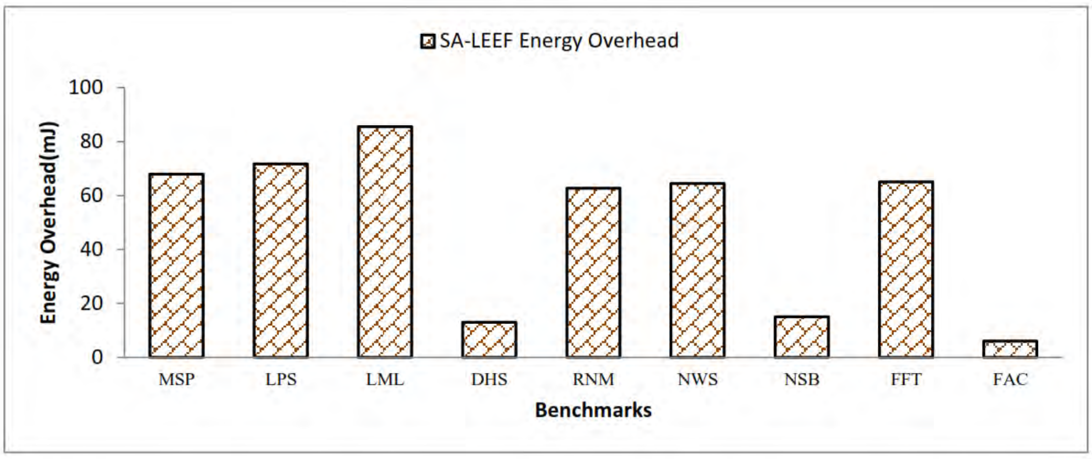

Figure 6.11: Energy Overhead of SA-LEEF Energy Estimation Framework

cations. The energy estimation overhead of Power Tutor during energy estimation of MemSpeedi, LinpackSP2, LivermoreLoops2, and Dhrystone2, is noticed 168mJ, 81mJ, 122mJ, and 16mJ, respectively. Alternatively, for RandMemi, NativeWhetstone2, Nested Branches, FFT1, and Factorial, Power Tutor has consumed 200mJ, 182mJ, 85mJ, 555mJ, and 42mJ energy, respectively. Among all chosen benchmark applications, during energy estimation of Dhrystone2, Power Tutor has consumed minimal energy. The energy estimation overhead of Power Tutor highly depends on the execution time of the smart-phone application. For instance, Power Tutor has consumed 99.97% more energy while estimating energy consumption for FFT1 benchmark than Dhrystone2. The standard deviation from mean for energy estimation overhead data for chosen benchmark applications is noticed 160.22. The variance from the standard deviation is noticed 1257.66. The main reason of high value of standard deviation is the non-homogenous nature of chosen benchmark applications.

Fig. 6.13 relates SA-LEEF and Power Tutor energy estimation tools based on their energy estimation overhead for a set of benchmark applications. Considering Power Tutor tool, the energy estimation overhead is noticed minimum for the Dhyrystone2 benchmark application. However, for SA-LEEF framework, during en-

University of Malaya

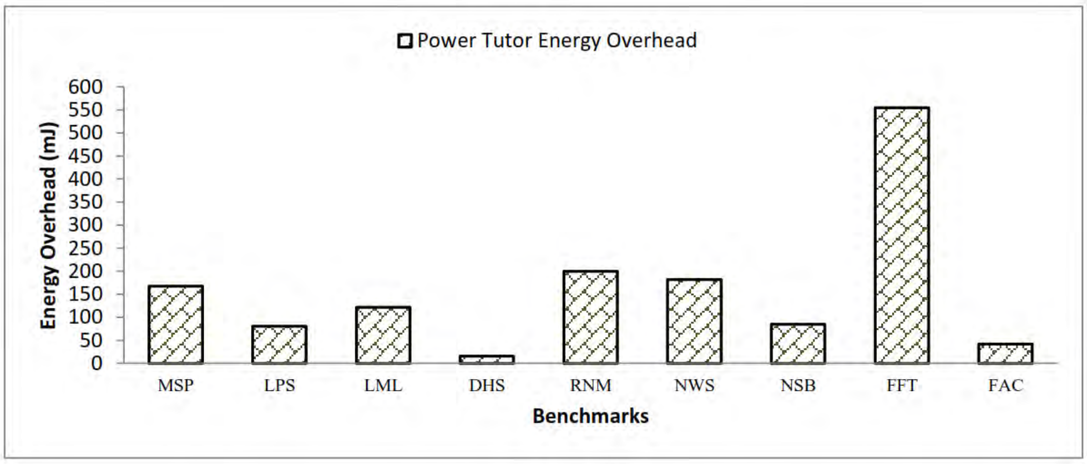

Figure 6.12: Energy Overhead of Power Tutor Energy Estimation Tool

ergy estimation of Factorial program SA-LEEF has consumed minimal energy. The main reason for this surprising behavior is because of unlike estimation approaches opted by Power Tutor and SA-LEEF estimation tools. For instance, in the case of SA-LEEF, the size of code (in assembly), loops and their size, frequency of execution paths, calls to functions, the accuracy of instruction/data storage location prediction, are the main reasons for total energy estimation overhead. However, for Power Tutor, in addition to the size of code, loops iterations are the utmost responsible entities for energy estimation overhead (Chapter 3). SA-LEEF's application profiler module scan through loops only three times and significantly suppresses the total execution time of application (overhead is dependent on execution time). Overall, energy estimation overhead of SA-LEEF framework during energy estimation of benchmark applications is much lower than Power Tutor. The ratio of energy estimation overhead of SA-LEEF to Power Tutor is different for different benchmark applications depending on their size. For MemSpeedi, LinpackSP2, LivermoreLoops2, and Dhrystone2, SA-LEEF has consumed 59.5%, 11.5%, 29.88%, and 18.75% less energy than Power Tutor during energy estimation of benchmark applications. Alternatively, for RandMemi, NativeWhetstone2, Nested branches, FFT1, and Factorial, SA-LEEF consumed 68.6%, 68.58%, 82.32%, 88.28%, and

University of Malaya

85.55% lower energy than Power Tutor energy estimation tool. Therefore, in terms of energy overhead, SA-LEEF outperforms dynamic energy estimation methods.

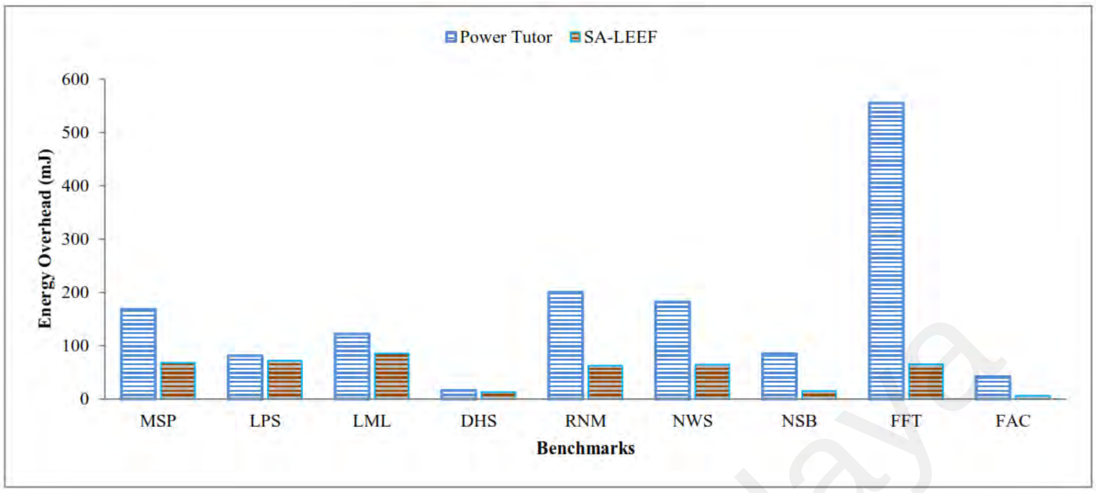

Figure 6.13: Comparison of Energy Estimation Methods based on Energy Consumption Overhead

6.2.2 Application Data Size Case

This section compares selected energy estimation tools to SA-LEEF framework when data sizes of the chosen benchmark application is increased. C variant of LinpackSP benchmark is used to analyze the impact of increasing data size on the performance of SA-LEEF framework in comparison to existing energy estimation methods. LinpackSP is a CPU bound benchmark application and it calculates the product of two matrices during execution on smart-phone. In this section, to highlight the impact of an increase in data size on the performance of SA-LEEF in comparison to existing methods, different variants of LinpackSP are created based on the size of integer matrices. For instance, in Linpack200, both input matrices are of size 200×200. In this section, it is assumed that ARM-IS energy profile and assembly code of the application is already available on smart-phone device.

University of Malaya

6.2.2.1 Energy Estimation Time

Fig. 6.14 highlights the effect of increasing benchmark data size on total energy estimation time of SA-LEEF energy estimation framework. In Fig. 6.14, X-axis highlights different variants of LinpackSP benchmark whereas Y-axis states average energy estimation time for each variant of LinpackSP benchmark. It is noticed that the average estimation time of SA-LEEF framework with different data sizes remain almost unchanged. In Fig. 6.14, the minimum reported estimation time is 2.53s when matrices of 200 are multiplied. The main reason for the ignorable rise in estimation time for SA-LEEF estimation framework is this that SA-LEEF estimation time is not affected by the increase in execution time (real-time) of the application. Rather, it changes with the increase in the line of code of the application. When a native smart-phone application is compiled with GCC compiler, it creates labels to represent the loops. Irrespective of the number of iterations of the loop, the assembly code size remains unchanged. The estimation time for Linpack200, Linpack400, Linpack600, Linpack800, Linpack1000, Linpack1200, and LinpackSP 1400, is noticed 2.533s, 2.533s, 2.534s, 2.535s, 2.535s, 2.535, and 2.536s, respectively. The small variation in estimation time is due to the activities running in the background of smart-phone during energy estimation by SA-LEEF framework. Also, standard deviation from the mean for estimation time of all benchmark applications is noticed 0.00113 only. The variance from the standard deviation is found 0.0000012 only. The main reason of this small value of standard deviation is the negligible effect of increasing data sizes within an application on the total estimation time.

Fig. 6.15 demonstrates energy estimation time analysis for Power Tutor energy estimation tool for increasing sizes of matrices within LinpackSP benchmark application. The X-axis of Fig. 6.15 states different variants of LinpackSP bench-

University of Malaya

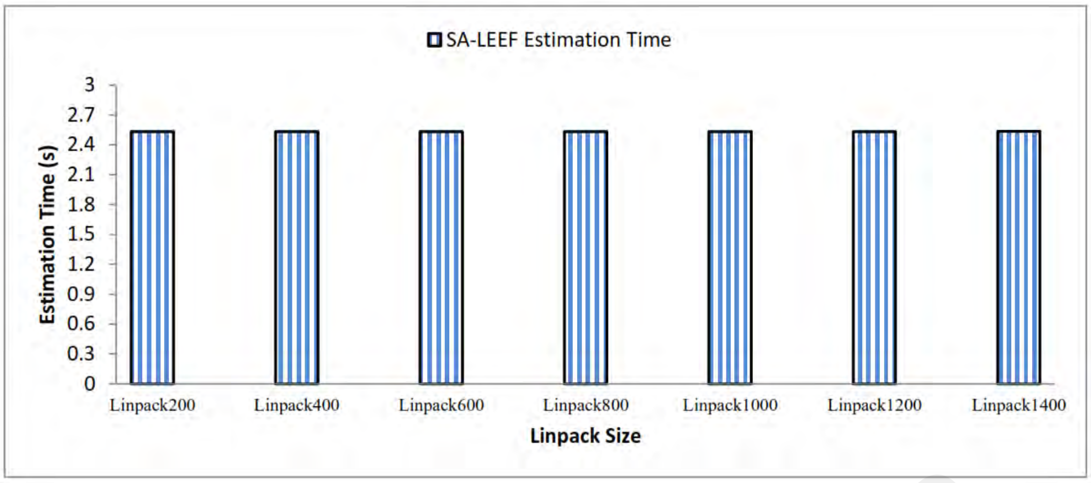

Figure 6.14: Effect of Data Size on the Estimation Time of SA-LEEF Framework

mark whereas Y-axis demonstrates average energy estimation time for each variant of LinpackSP benchmark application. It is noticed that increasing data size of matrices within LinpackSP benchmark significantly increases the total energy estimation time of Power Tutor energy estimation tool. As can be seen from Fig. 6.15, varying the data sizes of matrices within LinpackSP linearly increases the energy estimation time. The total estimation time for Linpack200, Linpack400, Linpack600, Linpack800, Linpack1000, Linpack1200, and Linpack 1400, is noticed 17.87s, 18.12s, 21.15s, 51.3s, 105.6s, 182.41s, and 298.6s, respectively. Due to high variance among the size of chosen data sets for LinpackSP benchmark, the standard deviation from the mean is noticed 78.5s. Also, the variance from standard deviation is very high and is noticed 6168.31s. The main reason of this high value of standard deviation and variance is the physical execution of the smart-phone application on the smart-phone device.

Fig. 6.16 concludes that varying data sizes within a benchmark application does not impacts the total energy estimation time of SA-LEEF framework. In comparison to Power Tutor, the performance of SA-LEEF for large sized data within LinpackSP benchmark application is much better. For instance, considering Linpack200 benchmark variant, SA-LEEF has reduced energy estimation time by a factor of 85.78%

University of Malaya

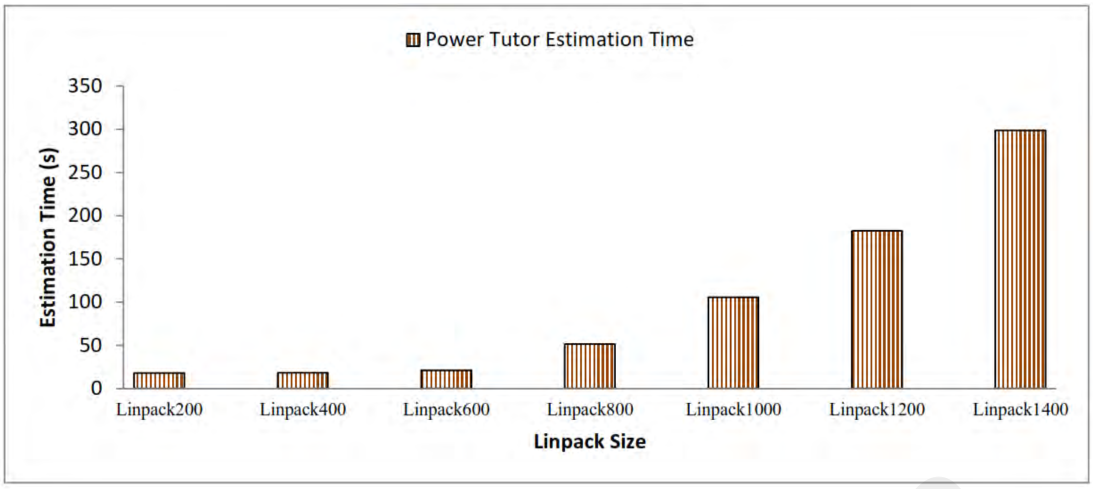

Figure 6.15: Effect of Data size on Estimation Time of Power Tutor Estimation Tool

compared to Power Tutor energy estimation tool. Whereas, for Linpack1400, it has reduced estimation time of Power Tutor by a factor of 99.14%. For Linpack400, Linpack600, Linpack800, Linpack1000, and Linpack1200, average energy estimation time of Power Tutor is 86%, 88%, 95%, 97%, and 98%, higher than SA-LEEF energy estimation framework. Therefore, SA-LEEF is a better choice when the size of data for a particular benchmark application is very large. For small data sized applications, the performance of SA-LEEF is either equal or lower than existing dynamic analysis based energy estimation methods. Considering Linpack1400 variant of benchmark application, the estimation time difference is noticed highest. The main reason for such huge difference in estimation time for Linpack1400 is due to the fact that the body of the loops is traversed only three times for SA-LEEF. For SA-LEEF, the energy estimation time is not affected by changing data size within the loops. However, in the case of Power Tutor, the estimation time is directly proportional to the data size (loop bounds) of the loops.

6.2.2.2 Energy Estimation Overhead

This section compares SA-LEEF energy estimation framework to existing energy estimation methods based on energy estimation overhead parameter. It highlights

University of Malaya

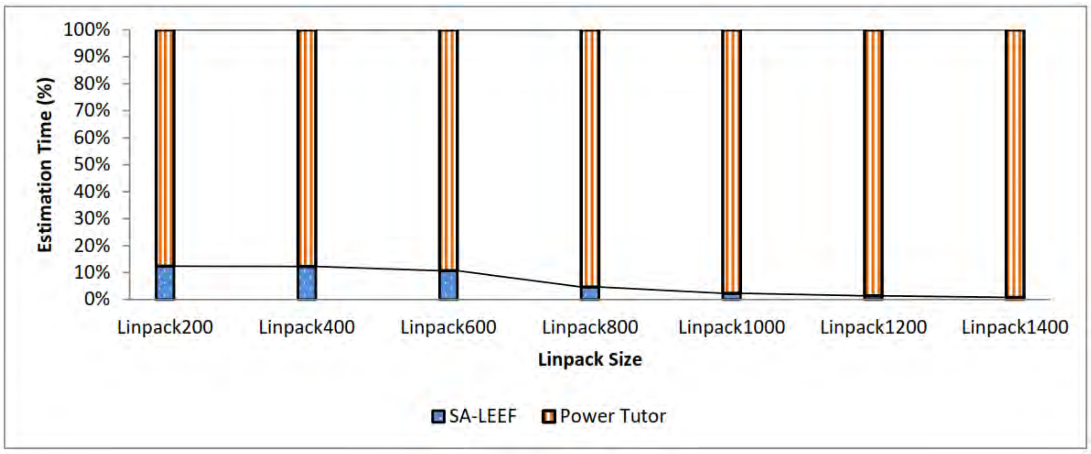

Figure 6.16: Power Tutor vs. SA-LEEF Energy Estimation Time

University of Malaya

the impact of increasing data sizes within LinpackSP benchmark on the average energy overhead of Power Tutor.

Fig. 6.17 demonstrates energy estimation overhead of SA-LEEF framework when data size within LinpackSP benchmark application is increased. In said figure, X-axis describes different variants of LinpackSP benchmark whereas Y-axis states average energy estimation overhead across each variant of LinpackSP benchmark. It is noticed that SA-LEEF energy estimation overhead is almost same for all the variants of LinpackSP benchmark. For chosen data sizes of LinpackSP benchmark, the minimum energy overhead is noticed 73.34mJ, whereas, maximum energy overhead is observed 73.59mj. For Linpack400, Linpack600, Linpack800, Linpack1000, and Linpack1200, energy estimation overhead of SA-LEEF is estimated 73.34mJ, 73.40mJ, 73.59mJ, 73.59mJ, and 73.59mJ, respectively. In case of SA-LEEF framework, for all chosen variants of LinpackSP benchmark, standard deviation from the mean is noticed 0.125 only.

Fig. 6.18 demonstrates energy estimation overhead of Power Tutor energy estimation tool for different variants of LinpackSP benchmark. The energy estimation overhead of Power Tutor is very high in comparison to SA-LEEF framework because of unlike energy estimation methods opted by both of these estimation methods.

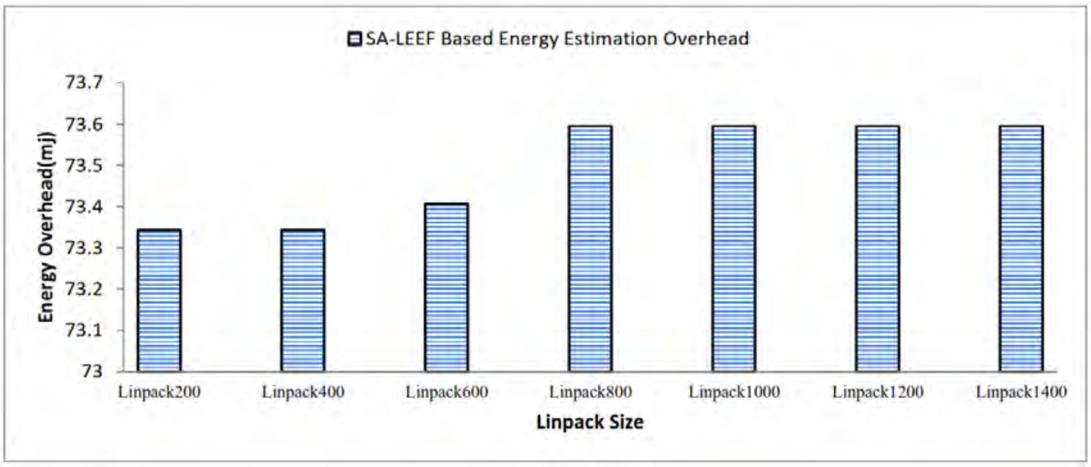

Figure 6.17: Effect of Data Size on Energy Estimation Overhead of SA-LEEF

The minimum energy estimation overhead for all variants of LinpackSP is noticed 182.81mJ when LinpackSP200 $\times$ 200 is run on smart-phone device (for Power Tutor). The highest energy overhead for Power Tutor is noticed 3580.21mj when matrices of size 1400 $\times$ 1400 and 1400 $\times$ 1400 are multiplied. The standard deviation from the mean is noticed very high due to the high difference in energy estimation overhead values. It is noticed 1291.87. The variance from standard deviation is observed 1668939.

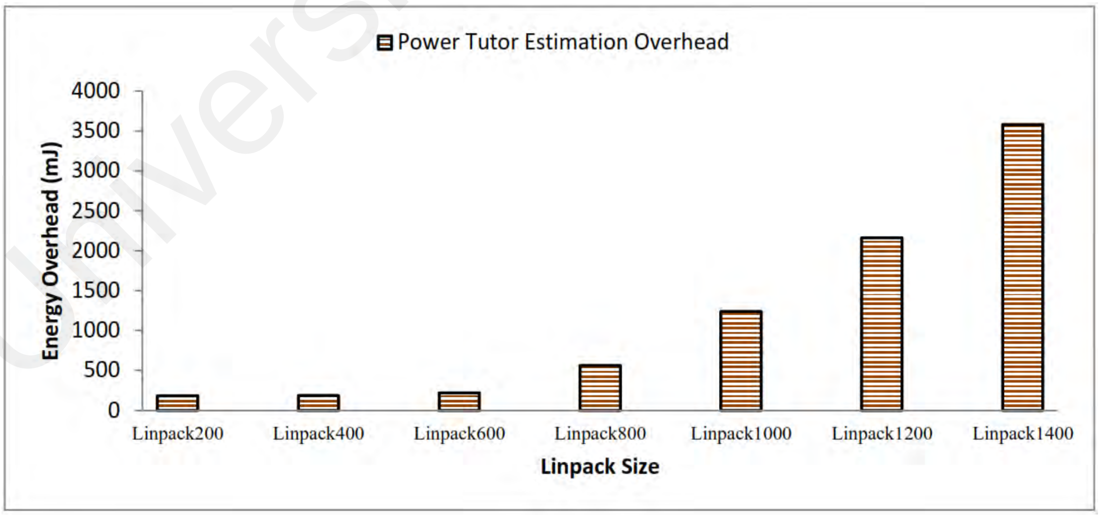

Figure 6.18: Effect of Data size on Estimation Overhead of Power Tutor Energy Estimation Tool

Fig. 6.19 compares SA-LEEF to Power Tutor energy estimation tool when size of data in LinpackSP benchmark is increasing. It is shown in Fig. 6.19 that the en-

University of Malaya

ergy estimation overhead of SA-LEEF is much lower than Power Tutor tool when the data size within benchmark application is very large. The energy estimation overhead of SA-LEEF is noticed 59%, 60%, 66%, 86%, 94%, 96%, and 97% lower than Power Tutor for LinpackSP200, LinpackSP400, LinpackSP600, LinpackSP800, LinpackSP1000, LinpackSP1200, and LinpackSP1400 benchmarks, respectively. Considering Linpack1400 variant of benchmark application, the energy estimation overhead difference among chosen energy estimation methods is noticed highest. The main reason for such huge difference in energy estimation overhead for Linpack1400 is due to the fact that the body of the loops is traversed only three times for SA-LEEF. For SA-LEEF, the energy estimation overhead is not affected by changing data size within the loops. However, in the case of Power Tutor, the energy estimation overhead is directly proportional to the data size of the loops (high estimation time).

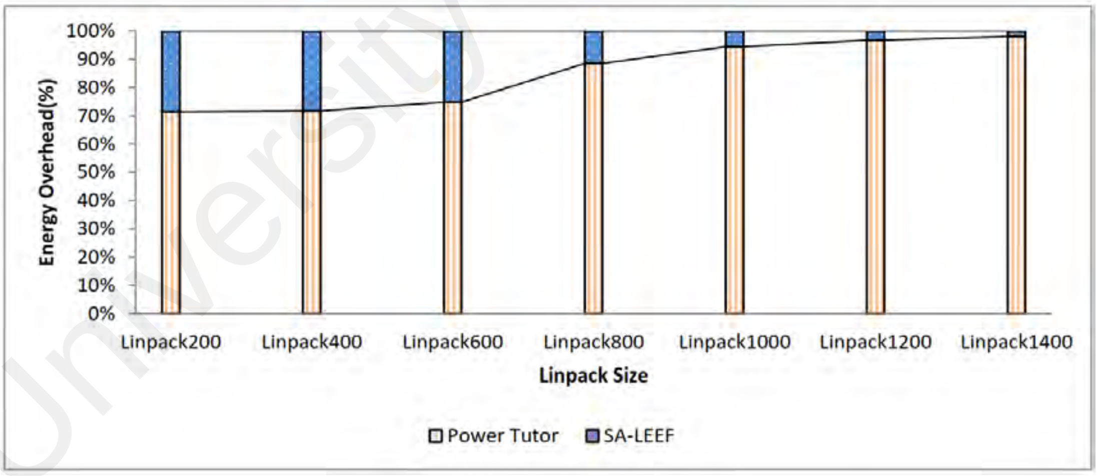

Figure 6.19: SA-LEEF vs. Power Tutor Estimation Overhead

6.2.2.3 Energy Estimation Accuracy

Fig. 6.20 demonstrates energy consumption for different variants of LinpackSP benchmark application estimated using SA-LEEF energy estimation tool. Energy consumption of benchmark application depends on the total number of instructions

University of Malaya

that CPU executes at run time. The energy consumption of different variants of LinpackSP such as Linpack200, Linpack400, Linpack600, Linpack800, Linpack1000, Linpack1200, and Linpack 1400, is estimated 8.51j, 8.57j, 9.58j ,23.66j, 49.98j, 86.66j, and 142.84j using SA-LEEF framework. The standard deviation from the mean for different benchmark variants is estimated 51.18. Alternatively, the variance from standard deviation is noticed 2620.35.

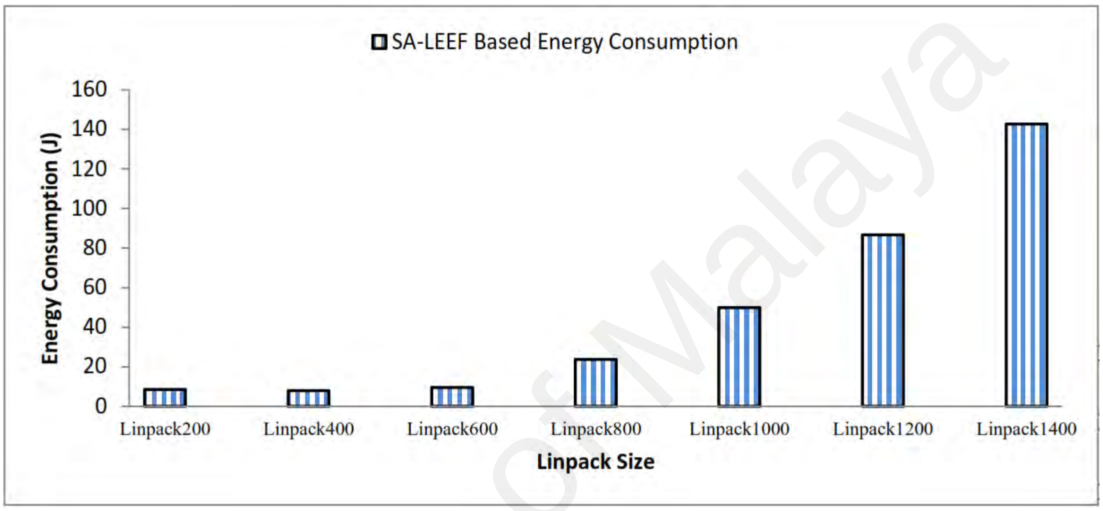

Figure 6.20: Effect of Data Size on Energy Consumption of LinpackSP using SA-LEEF

University of Malaya

Fig. 6.21 highlights energy consumption of different variants of LinpackSP benchmark application using the Power Tutor energy estimation tool. The energy consumption of LinpackSP for different data sizes is noticed different. The energy consumption of different variants of LinpackSP application such as Linpack200, Linpack400, Linpack600, Linpack800, Linpack1000, Linpack1200, and Linpack 1400, is estimated 8.99j, 9.06j, 10.12j, 24.99j, 52.8j, 91.6j, and 150.9 j, respectively. The standard deviation from the mean for chosen variants of LinpackSP is estimated 50.23. The high value of standard deviation from the mean is due to high variance in execution time of different variants of LinpackSP benchmark application.

Fig. 6.22 compares SA-LEEF and Power Tutor energy estimation tools based on their estimation accuracy for different variants of LinpackSP benchmark. In the

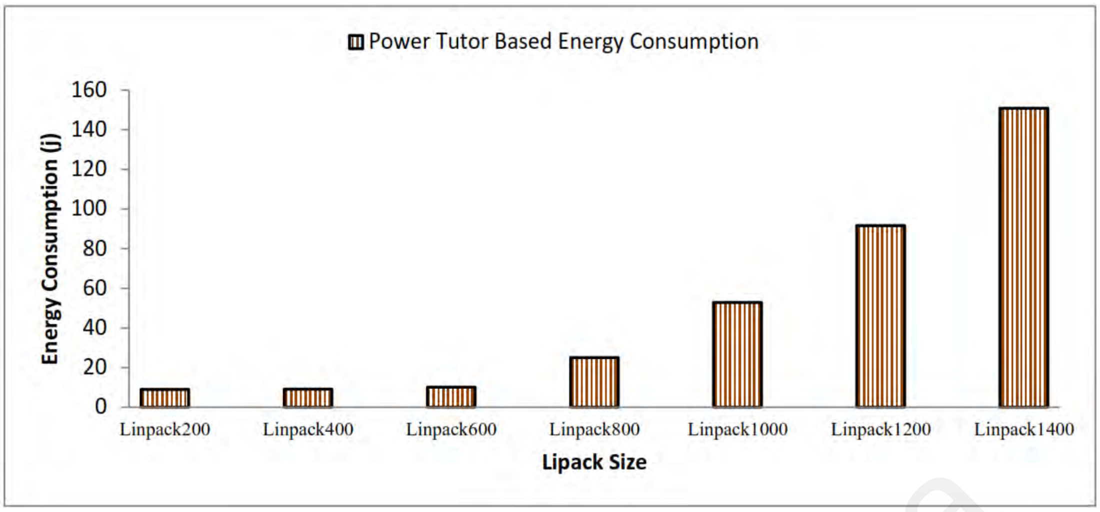

Figure 6.21: Effect of Data Size on Energy Consumption of LinpackSP using Power Tutor

said figure, X-axis describes LinpackSP variants whereas Y-axis states estimation accuracy across each variant of benchmark application while considering measurement based method as the measurement of ground truth value. Energy estimation accuracy for different variants of LinpackSP benchmark is approximately similar for both SA-LEEF and Power Tutor tool. The estimation accuracy for Power Tutor is noticed in the range of 90-92.1%. The standard deviation from the mean for chosen variants of LinpackSP benchmark using Power Tutor is estimated 0.52. Also, the variance from the standard deviation is noticed 0.27. The estimation accuracy for SA-LEEF framework is noticed in the range of 86.4-88.8% for different variants of LinpackSP benchmark application. The standard deviation from the mean is noticed 0.19 for SA-LEEF framework. The variance from standard deviation for different variants of LinpackSP benchmark using SA-LEEF is noticed 0.03 only. The main reason of low accuracy of SA-LEEF compared to Power Tutor is due to the fact that SA-LEEF does not consider the impact of the power state of smartphone components during estimation. Also, nested branching inaccurate the total energy consumption as SA-LEEF considers weighted probability for execution flow analysis.

University of Malaya

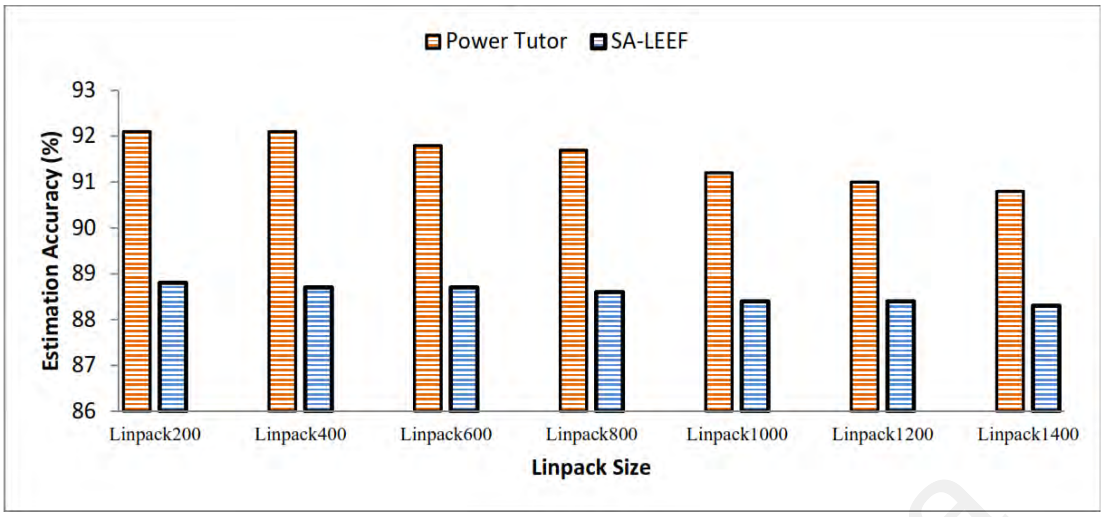

Figure 6.22: SA-LEEF vs. Power Tutor Energy Estimation Accuracy

6.3 Performance Comparison of SA-LEEF Framework (Remote Mode) to Existing Energy Estimation Methods

This section compares SA-LEEF framework to existing dynamic analysis based energy estimation tools when ARM-IS energy profile and assembly based version of benchmark application is not available on local smart-phone device. In this scenario, ARM-IS energy profile is hosted on a physical server. The physical server is located either within local premises of a user (i.e., cloud-let) or is located on a remote cloud server. This study has compared the performance of SA-LEEF to Power Tutor and external measurement methods for both scenarios.

6.3.1 Energy Estimation Time

This section compares two variants of SA-LEEF framework with dynamic analysis based energy estimation methods. It compares energy estimation methods (SA-LEEF, Power Tutor, and Measurement) based on average energy estimation time as discussed below in details.

Fig. 6.23 compares SA-LEEF framework with Power Tutor and measurement-based energy estimation methods. X-axis in Fig. 6.23 highlights a set of benchmark applications whereas Y-axis demonstrates average energy estimation time for each

University of Malaya

energy estimation method against chosen benchmark applications. The reported results are the average of 15 runs on the smart-phone. In Fig. 6.23, a scaling function is used to suppress the huge difference among findings of SA-LEEF and dynamic solutions for better presentation. For instance, the reported results for Power Tutor and measurement-based energy estimation methods are divided by 3. Two operational variants of SA-LEEF framework including SA-LEEF (LOC SER) and SA-LEEF (REM SER) are presented. SA-LEEF (LOC SER) represents the case where ARM-IS energy profile and assembly code of target smart-phone application is hosted on a local server. SA-LEEF framework accesses these files from the server to estimate energy consumption of the application. Alternatively, SA-LEEF (REM SER) states the case when ARM-IS energy profile and assembly code of the smart-phone application is hosted on a remote cloud server.

University of Malaya

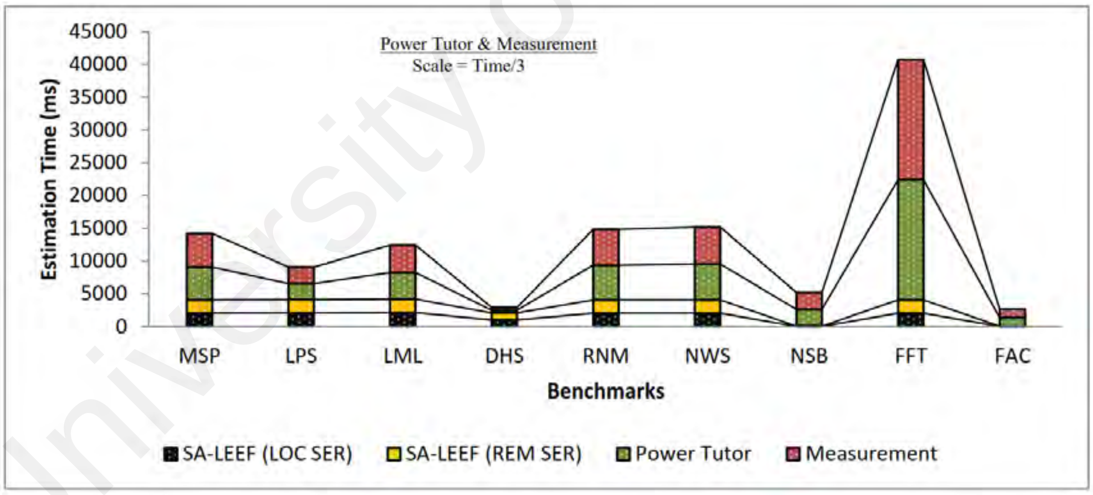

Figure 6.23: Comparison of SA-LEEF to Existing Methods based on Energy Estimation Time

Energy estimation time of measurement based method is noticed highest among all chosen energy estimation methods. Total energy estimation time of measurement based method consists of application execution time and offline analysis on time-stamped power profile for noise suppression (neighborhood operation). Within total energy estimation time of measurement based method, application execution time is superior to offline analysis time (98.99%). The estimation time of Power Tutor too is much higher than SA-LEEF framework. In comparison to SA-LEEF (REM SER), energy estimation time of SA-LEEF (LOC SER) is lower. On average, the energy estimation time of measurement based method is 2-4% higher than Power Tutor energy estimation tool. The estimation time difference between Power Tutor and measurement based method is minimal when the execution time of target benchmark application is very small. For the selected benchmark applications such as MemSpeedi, LinpackSP2, LivermoreLoops2, Dhrystone2, RandMemi, NativeWhetstone2, Nested Branches, FFT1, and Factorial, SA-LEEF (LOC SER) is 86.78, 73.05, 83.23, 21.38, 88.70, 88.02, 99.33, 96.32, and 99.46 times faster than measurement-based estimation method. Similarly, the energy estimation time of SA-LEEF (LOC SER) is 86.43%, 72.32%, 82.81%, 21.38%, 87.39%, 87.73 %, 99.30%, 96.32%, and 99.46% lower than Power Tutor energy estimation tool for aforementioned benchmark applications. The similar behavior is noticed when SA-LEEF (REM SER) is compared with measurement and Power Tutor energy estimation tools. For instance, considering MemSpeedi, LinpackSP2, LivermoreLoops2, Dhrystone2, RandMemi, and NativeWhetstone2 benchmark applications, SA-LEEF (REM SER) has reduced total energy estimation time of Power Tutor by 86.43%, 72.32%, 82.81%, 21.38%, 87.39%, and 87.73%, respectively.

University of Malaya

6.3.2 Energy Estimation Overhead

In this section, two variants of SA-LEEF framework with Power Tutor and measurement-based energy estimation methods are compared.

Fig. 6.24 presents a comparison among SA-LEEF, Power Tutor, and measurement-based energy estimation tools based on energy overhead during the estimation process. In Fig. 6.24, X-axis highlights a set of benchmark applications whereas Y-axis demonstrates total average energy estimation overhead for each energy estimation method against chosen benchmark applications. The scaling function in Fig. 6.24 divides the energy overhead of Power Tutor and measurement based method by 5 to improve the presentation of results. In Fig. 6.24, the presented results are the average of 15 runs on smart-phone. The energy overhead of SA-LEEF framework is presented in terms of two settings including SA-LEEF (LOC SER) and SA-LEEF (REM SER). Power Tutor runs benchmark application on smart-phone and it profiles execution behavior of application for its energy estimation. Therefore, the energy estimation overhead of Power Tutor is estimated based on energy overhead of Power Tutor and overhead of benchmark application running on smart-phone. Alternatively, energy estimation overhead of measurement based method is estimated based on the energy overhead of application executing on smart-phone.

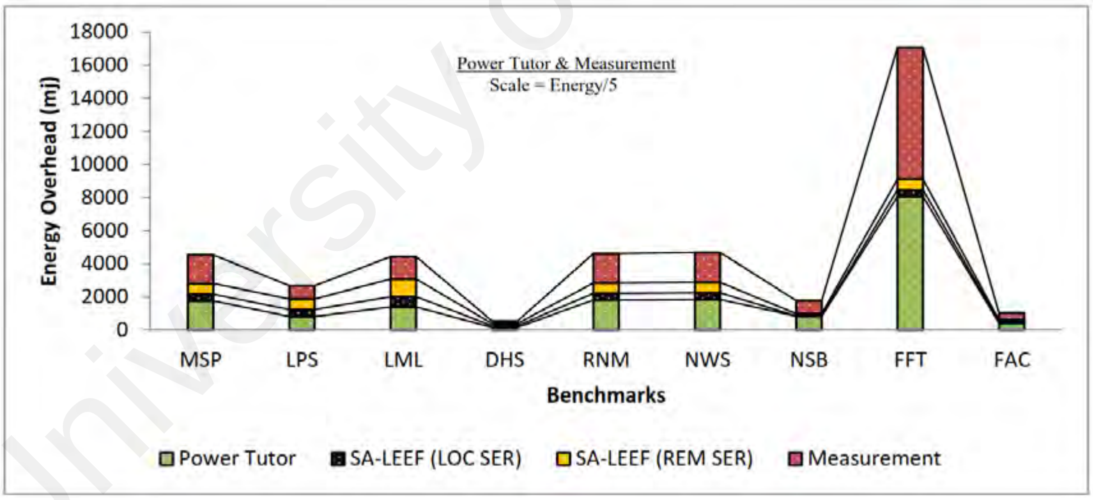

Figure 6.24: Comparison of SA-LEEF to Existing Energy Estimation Methods for Energy Estimation Time

The energy estimation overhead of Power Tutor is noticed highest among all chosen energy estimation methods. Alternatively, energy estimation overhead of SA-LEEF is noticed lowest among all chosen energy estimation methods. In terms of energy estimation overhead, the performance of SA-LEEF is much better than measurement-based energy estimation method. For the selected benchmark appli-

University of Malaya

cations such as MemSpeedi, LinpackSP2, LivermoreLoops2, and Dhrystone2, SA-LEEF(LOC SER) is 95.30, 89.32, 90.61, and 87.55 times energy efficient than measurement based method. Also, for RandMemi, NativeWhetstone2, Nested branches, FFT1, and Factorial benchmarks, SA-LEEF (LOC SER) has consumed 95.46%, 95.51%, 99.25%, 98.96%, and 96.54%, less energy than measurement-based energy estimation method. Alternatively, while considering SA-LEEF (REM SER) remote profile setting, SA-LEEF has consumed 92.72, 83.66, 84.4, and 82.65 times less energy while estimating energy consumption of MemSpeedi, LinpackSP2, LivermoreLoops2, and Dhrystone2 benchmark applications, respectively. Also, it consumes 92.88%, 93.01%, 96.78%, 98.39%, and 94.51% less battery charge during energy estimation of RandMemi, NativeWhetstone2, Nested branches, FFT1, and Factorial benchmark applications, respectively.

In comparison to Power Tutor, SA-LEEF (LOC SER) local mode has reduced energy consumption budget by 95.39%, 89.53%, 90.78%, and 87.81% for MemSpeedi, LinpackSP2, LivermoreLoops2, and Dhrystone2, benchmark applications, respectively. On the other hand, for RandMemi, NativeWhetstone2, Nested branches, FFT1, and Factorial benchmarks, it has consumed 95.56%, 95.60%, 99.26%, 98.97%, and 96.61% less energy than Power Tutor energy estimation tool. SA-LEEF (REM SER) remote setting consumes more energy than SA-LEEF (LOC SER) setting due to high RTT value while downloading assembly file of benchmark application. While considering energy estimation overhead of estimation tools, SA-LEEF (LOC SER) has consumed 92.85%, 83.99%, 84.69%, and 83.01% less energy than Power Tutor while estimating energy consumption of MemSpeedi, LinpackSP2, LivermoreLoops2, and Dhrystone2, respectively. Alternatively, for RandMemi, NativeWhetstone2, Nested branches, FFT1, and Factorial benchmarks, SA-LEEF (REM SER) has reduced the total energy consumption of Power Tutor by a factor of 93.04%, 93.15%,

University of Malaya

96.85%, 98.42%, and 94.62%, respectively. For the selected benchmark applications such as MemSpeedi, LinpackSP2, LivermoreLoops2, Dhrystone2, RandMemi, NativeWhetstone2, Nested branches, FFT1, and Factorial, energy estimation overhead of SA-LEEF (LOC SER) is 35.45, 34.63, 39.78, 28.24, 36.20, 35.74, 76.72, 35.16, and 37.04 times lower than SA-LEEF(REM SER) framework. The energy estimation accuracy is not affected by the remote profiling mode that is why results are not added here.

6.4 SA-LEEF Resource Consumption Comparison to Existing Energy Estimation Methods

This section compares SA-LEEF energy estimation framework with Power Tutor in terms of its resource consumption behavior. This study has evaluated performance behavior of SA-LEEF for two smart-phone resources including CPU and RAM usage.

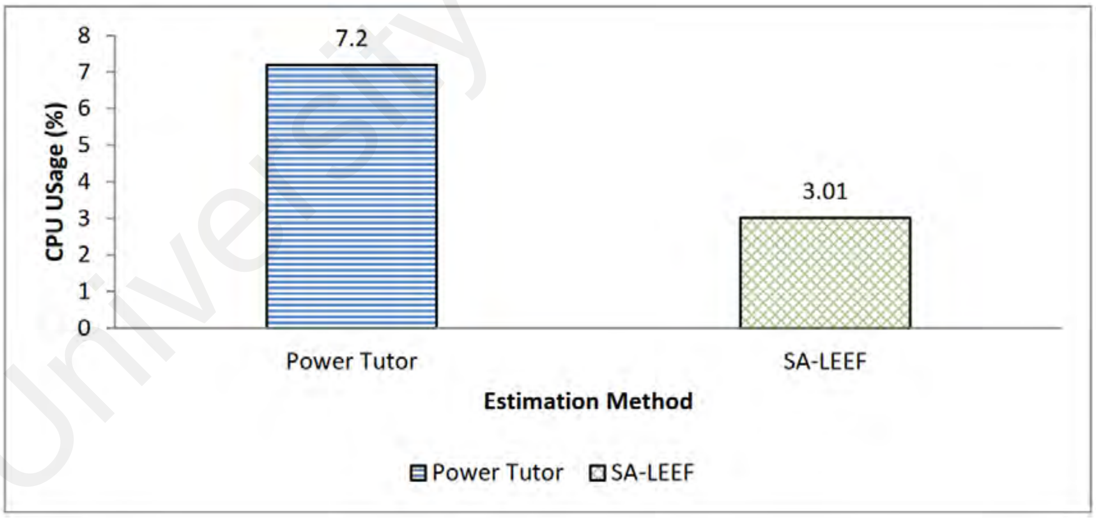

Figure 6.25: Comparison of Energy Estimation Methods based on CPU Resource Consumption

Fig. 6.25 highlights CPU resource consumption of SA-LEEF in comparison to Power Tutor energy estimation tool. The X-axis of Fig. 6.25 highlights chosen energy estimation tools whereas Y-axis states average CPU usage. This study has used *top* Linux utility to estimate the CPU usage for energy estimation tools. It is

University of Malaya

noticed that SA-LEEF has consumed less CPU capacity than Power Tutor during energy estimation process. SA-LEEF has consumed 58% fewer CPU resource than Power Tutor energy estimation method. The main reason for this behavior is the static analysis approach that makes SA-LEEF lightweight in terms of its CPU usage. Power Tutor continuously monitors the smart-phone components power states. Due to continuous monitoring of smart-phone components state, the CPU usage of Power Tutor is higher than SA-LEEF. The reported results in Fig. 6.25 are an average of 15 runs. It was observed that the standard deviation is 0.55 from the mean for 15 runs. Also, for 95% percentile, the confidence interval is observed 0.46. For power Tutor the standard deviation from mean is observed 0.91. Moreover, for 95% percentile, the value of confidence interval is estimated 0.73.

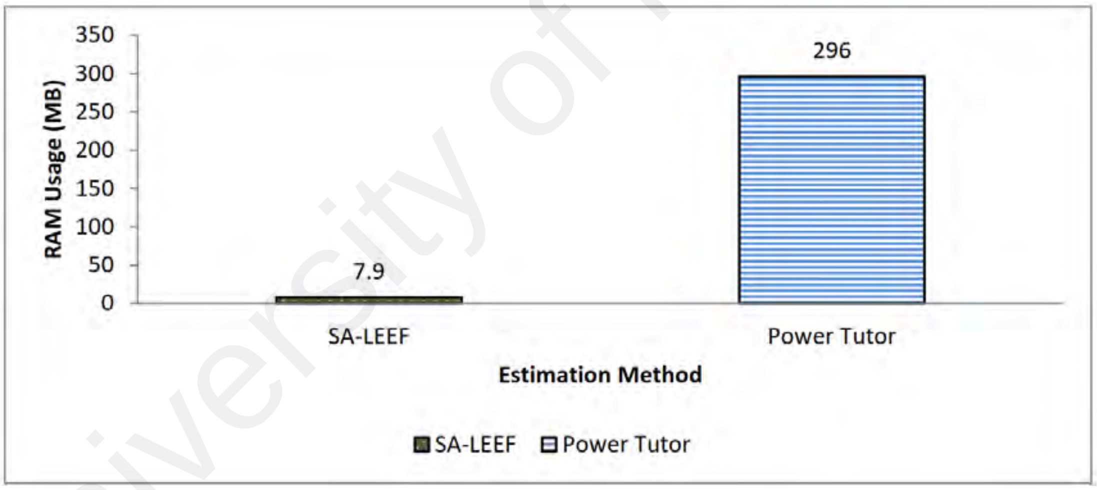

Figure 6.26: Comparison of Energy Estimation Methods based on RAM Resource Consumption

Fig. 6.26 compares SA-LEEF and Power Tutor energy estimation methods for RAM usage during the energy estimation process. The X-axis of Fig. 6.26 highlights chosen energy estimation tools whereas Y-axis states average RAM usage. To estimate the RAM usage of an estimation tool, this study has considered *Pmap* utility of Linux. It is noticed that SA-LEEF is much resource efficient in terms of RAM usage than Power Tutor energy estimation tool. SA-LEEF consumes up to 97% less RAM

University of Malaya

capacity during energy estimation. The main reason for this behavior is the static analysis approach of SA-LEEF. The design of Power Tutor is heavyweight as it follows dynamic analysis for estimation. This study has run the experiment 15 times to report the average RAM usage for energy estimation methods. The standard deviation from the mean is noticed 0.08 whereas, for 95% percentile, the confidence interval is estimated 0.11. For Power Tutor energy estimation tool, the standard deviation from the mean is observed 1.81. Alternatively, for 95% percentile, the value of confidence interval is estimated 2.25.

6.5 Qualitative Comparison of SA-LEEF Framework

The code analysis based energy estimation schemes consider energy cost of software operations to estimate energy consumption of an application. Due to the absence of a simulator and LEAP power measurement equipment for smart-phone devices to deploy the same system for comparison, a qualitative analysis of SA-LEEF is performed to the relevant studies. Table 6.1 compares SA-LEEF to existing code analysis based energy estimation schemes based on a set of parameters including Flow Analysis Method, Storage Analysis, Concurrent Execution Overhead, Instruction Granularity, Application Type, Power Measurement Equipment, and Power Profile Availability parameters.

Existing energy estimation methods such as eLens (Hao et al., 2013), eCalc (Hao et al., 2012a), Ins-Ener (D. Li et al., 2013), and Eco-Droid (Jabbarvand et al., 2015) has considered dynamic analysis approach to estimate execution paths of an application. Dynamic analysis category of energy estimation schemes instruments the application to capture the execution flow across a set of application use cases. However, the test cases for many of the smart-phone applications are not available usually that makes this approach impractical. Also, dynamic analysis based

University of Malaya

application profiling is a time and resource consuming process. Alternatively, SA-LEEF is unique as it considers static analysis approach (weighted probability based estimation) to estimate energy consumption of an application. Also, in existing energy estimation schemes, cache analysis is overlooked (eLens (Hao et al., 2013), eCalc (Hao et al., 2012a), Tiwari et al. (Tiwari et al., 1994), Ins-Ener (D. Li et al., 2013), and Eco-Droid (Jabbarvand et al., 2015)). SA-LEEF estimates energy consumption of an application based on the set of instructions fetched either from the RAM storage or system cache.

For energy estimation of an application, an energy estimation tool runs on the smart-phone device in parallel to several others smart-phone applications. Due to resource sharing, cache eviction owing to high context switching leads to high energy overhead. Existing energy estimation schemes such as eLens (Hao et al., 2013), eCalc (Hao et al., 2012a), Ins-Ener (D. Li et al., 2013), Tiwari et al. (Tiwari et al., 1994), and Eco-Droid (Jabbarvand et al., 2015) has overlooked energy overhead due to cache eviction. Alternatively, SA-LEEF has considered energy overhead due to concurrent program execution while estimating energy consumption of an application. Instruction granularity states the level at which energy cost of software operations is profiled. High-level granularity attribute defines the energy cost of software libraries, system calls, and APIs. Existing energy estimation schemes such as eLens (Hao et al., 2013), eCalc (Hao et al., 2012a), Ins-Ener (D. Li et al., 2013), and Eco-Droid (Jabbarvand et al., 2015) has considered energy profile of high level software operations. On the other hand, SA-LEEF has profiled energy consumption of low-level assembly based instructions. Energy estimation of an application based on the fine granular operations is more effective as it is not affected by bugs of the system (sleep, wake-lock).

Existing energy estimation schemes including eLens (Hao et al., 2013), eCalc (Hao

University of Malaya

Table 6.1: Qualitative Comparison of SA-LEEF

Study	Flow Analysis Method	Storage Analysis	Concurrent Execution Overhead	Instruction Granularity	Application Type	Power Measurement Equipment	Power Profile Available?
eLens (Hao et al., 2013)	Dynamic	NO	NO	High Level	Source Code	LEAP	NO
eCalc (Hao et al., 2012a)	Dynamic	NO	NO	High Level	Source Code	LEAP	NO
Ins-Ener (D. Li et al., 2013)	Dynamic	NO	NO	High level	Source Code	LEAP	NO
Eco-Droid (Jabbarvand et al., 2015)	Dynamic	NO	NO	API	Source Code	Monsoon	NO
Tiwari et al. (Tiwari et al., 1994)	N/A	N/A	N/A	Low Level	Assembly Source Code	Ammeter	Incomplete
SA-LEEF	Static	YES	YES	Low Level	Objdump	EM6000 multimeter	YES

et al., 2012a), Ins-Ener (D. Li et al., 2013), and Eco-Droid (Jabbarvand et al., 2015) requires java source code of the application for its energy estimation. However, the source code of many applications is not available due to the privacy issues. SA-LEEF is unique in comparison to aforementioned energy estimation schemes as it estimates energy consumption using obj dump of the application. SA-LEEF provides energy profile of ARM-IS instructions online for its users. Existing energy estimation schemes such as eLens (Hao et al., 2013), Ins-Ener (D. Li et al., 2013), and Eco-Droid (Jabbarvand et al., 2015) have not published the energy profile of software operations either scholarly or via web services.

6.6 Conclusion

In this chapter, SA-LEEF is validated and compared to Power Tutor and measurement-based energy estimation methods. The performance of SA-LEEF is compared to existing estimation tools with dissimilar application code and data sizes. Moreover, the performance of SA-LEEF is compared to existing schemes in two of its operational mode, (a) Local ARM-IS profile hosting, and (b) Remote ARM-IS profile hosting mode. The estimation of system model for SA-LEEF is observed 97.37% accurate to its empirical evaluation.

The energy estimation time of SA-LEEF is compared to Power Tutor and measurement based energy estimation methods. Energy estimation time of SA-LEEF is noticed up to 98% lower than Power Tutor and measurement based methods

University of Malaya

for large data sized applications. While considering small data sized applications, energy estimation time of SA-LEEF is 6% and 6.1% lower than Power Tutor and measurement based methods, respectively. In terms of energy estimation overhead, SA-LEEF has consumed up to 97% less energy than Power Tutor during energy estimation of large data sized smart-phone application. However, for small data sized applications, the energy overhead of SA-LEEF is 11.5% lower than the Power Tutor energy estimation tool. In terms of estimation accuracy, findings of SA-LEEF framework are accurate up to 88% to the ground truth energy estimation values captured through measurement based method. However, energy estimation accuracy of SA-LEEF is lower than Power Tutor by a marginal range of 3-4%. For remote ARM-IS profiling mode, the performance of SA-LEEF is 99% and 96% better than Power Tutor in terms of energy estimation time and overhead, respectively. SA-LEEF has significantly reduced energy estimation time and overhead of existing energy estimation tools while offering acceptable accuracy for various applications such as MCC, IoT, and WSN. In terms of resource consumption rate, SA-LEEF consumes 58% less CPU resource capacity than Power Tutor energy estimation tool during estimation process. For RAM usage, SA-LEEF requires up to 97% less smart-phone RAM during energy estimation of smart-phone applications. However, the performance of SA-LEEF in terms of its estimation time and energy is badly affected if the wall clock execution time of an application is lower than 1.5s.

University of Malaya

CHAPTER 7: CONCLUSION

This chapter reexamines the research objectives set in the first chapter of this thesis and draws a conclusion of the whole research work. It discusses limitations, research contributions, and presents future research directions of this work.

The organization of the chapter is as follows. Section 7.1 discusses reappraisal of research objectives, the procedure to achieve the objectives, and presents research findings. Section 7.2 highlights the limitations and applications of this research work. Section 7.3 highlights the main contributions. Lastly, Section 7.4 highlights several possible research directions from this work for further research to solve the problems in this field of research.

7.1 Reappraisal of Research Objectives

This research work aimed to solve the problems of dynamic analysis based energy estimation schemes. To achieve research objectives set in section 1.5, the given below research road-map is followed.

Objective 1: To explore existing energy estimation schemes to gain insights to the performance limitations of current state-of-the-art solutions.

The first objective of this research work was to study and critically analyze existing energy estimation schemes to highlight their limitations. To accomplish this research objective, the current study thoroughly reviewed existing energy estimation schemes to highlight their performance limitations. It has thoroughly reviewed design, approaches, and methods for energy estimation of smart-phone applications. It has performed an extensive search on online databases such as ACM, Elsevier, Web of science, and Springer, to review existing literature in energy estimation of smart-phone applications research domain. It has proposed thematic taxonomies

University of Malaya

to classify the existing literature based on parameters which were common in the majority of the literature. Furthermore, existing schemes are compared based on thematic taxonomies to highlight the commonalities and variances among existing schemes. The issues in existing energy estimation schemes are highlighted to propose possible research directions in this domain of research.

An analysis on open research issues is performed to investigate the feasibility of research in each direction. It was found that existing energy estimation schemes inefficiently utilizes underlying resources of smart-phone while estimating energy consumption of smart-phone applications. It was also noticed that existing energy estimation schemes consider dynamic analysis for application execution path finding and overlooks storage location of instructions during energy estimation. Therefore, to minimize the overhead of existing energy estimation schemes, an efficient lightweight energy estimation framework was needed in this domain of research.

Objective 2: To investigative performance overhead of dynamic analysis based energy estimation schemes to reveal inefficiencies of existing methods.

The second objective of this research was to analyze and investigate overhead of existing energy estimation schemes. The proposed research has considered dynamic analysis energy estimation schemes to examine their performance. The parameters considered for investigating the performance of dynamic analysis based estimation methods include energy estimation time, overhead, accuracy, and resource consumption during energy estimation process of smart-phone applications. It was noticed that the energy overhead and estimation time of dynamic analysis schemes is very high and highly depends on the size of data within an application. The energy estimation time of Power Tutor was noticed 14.3s, 7.8s, and 10.1s for NativeWhetstone2, LinpackSP2, and LivermoreLoops2 benchmark applications. The energy

University of Malaya

overhead of Power Tutor was noticed 187mJ. It was noticed that energy overhead and estimation time increases with increase in total execution time and operations within a smart-phone application. The energy consuming elements within a smart-phone application are also highlighted. It was noticed that up to 89% energy budget of an application (LinpackSP2) is consumed by the code written within loops of an application. It was noticed that Power Tutor consumed 2-10% of the total CPU capacity during energy estimation of smart-phone application. It was also noticed that because of dynamic analysis approach existing energy schemes exhibit high energy estimation time and overhead.

Objective 3: To develop a system to estimate energy consumption of assembly based instructions within ARM-ISA for both system cache and RAM storage access.

The third objective of this research was to estimate energy consumption of fine granular assembly based instructions for ARM-7 ISA. To estimate energy cost of ARM based assembly instructions, EM6000 multimeter is used. In the designed experimental setup, a resistor of one ohm resistance was attached to the battery terminals. Multimeter was interfaced to the sense resistor and desktop server. A set of test programs were designed to estimate energy consumption of instructions fetched from the RAM and cache storage. Moreover, energy is estimated for instructions performing integer and floating point operations. During test program execution on smart-phone device, multi-meter capture and records timestamped voltage drop at the desktop server. During post processing phase, timestamped voltage profile is processed to estimate power consumption of the test program. Also, weighted filter is applied to suppress the noise from the power profile to eliminate the effect of background activities on the power profile of the test programs. The timestamped power profile and size of the test program was used to estimate energy consumption

University of Malaya

of single ARM assembly instruction. It was observed that power consumption rate while running different type of test programs vary a little. It was also observed that energy consumption of each instruction highly depends on its CPI. The energy consumption of branch instructions was noticed very high. It was concluded that arithmetic operations based instructions consume more energy than logical operation based instructions. Moreover, the energy cost of multiply and divide operations was also noticed higher compared to arithmetic operations. The energy cost of floating based operations was noticed much higher than integer based operations because of their higher CPI. The energy cost of accessing instructions from RAM is higher that accessing it from local cache. Moreover, it was noticed that energy consumption of system libraries is very high.

Objective 4: To design and develop a lightweight energy estimation framework that proposes weighted probability based application execution flow estimation and static cache analysis to estimate energy consumption of smart-phone applications.

The fourth objective of this research was to propose a lightweight static analysis based energy estimation framework. The lightweight feature in proposed framework is achieved by eliminating the need to run the application on smart-phone for its energy estimation. The proposed energy estimation method analysis native smart-phone application and employs cost models for different operations of an application to estimate its energy consumption. However, recent smart-phone applications are non-deterministic by nature. The proposed work considered weighted probability theory to handle the non-deterministic nature of smart-phone applications. SA-LEEF employs probabilistic theory to predict the run time execution behaviors of a smart-phone application. For instance, it estimates execution path of an application based on a weighted probability function. It employs a slicing based approach to

University of Malaya

estimate the loop bounds within an application. For the data and storage location of instructions within the native smart-phone application, it considers cache distance based storage location prediction method. Two of main modules of proposed energy estimation framework includes Application Analyzer and ARM Instruction Energy Profiler. Application Analyzer and ARM Instruction Energy Profiler are responsible for resolving the issues of non-deterministic nature of the application and to find the operational cost of activities in a smart-phone application, respectively. The proposed framework hosts operational cost of activities in the smart-phone application on a cloud server to offer ARM-ISA energy profile as a service.

Objective 5: To evaluate the proposed energy estimation framework for energy estimation time, overhead, resource consumption, and accuracy, and to compare it with state-of-the-art energy estimation schemes.

The fifth objective of this research was to evaluate and validate the system model for the proposed framework. The viability of SA-LEEF is evaluated by testing benchmark applications on the real smart-phone device. SA-LEEF has shown remarkable performance in comparison to existing dynamic analysis based estimation methods. Energy estimation time of SA-LEEF is noticed up to 98% lower than Power Tutor and measurement-based energy estimation methods for large data sized applications. While considering small data sized applications, energy estimation time of SA-LEEF is 6% and 6.1% lower than Power Tutor and measurement-based energy estimation methods, respectively. In terms of energy estimation overhead, SA-LEEF has consumed up to 97% less energy than power tutor during energy estimation of large data sized smart-phone application. However, for small data sized applications, the energy overhead of SA-LEEF is 11.5% lower than Power Tutor energy estimation tool. In terms of estimation accuracy, findings of SA-LEEF framework are accurate up to 88% to the ground truth energy estimation values cap-

University of Malaya

tured through measurement-based method. However, energy estimation accuracy of SA-LEEF is lower than Power Tutor by a marginal range of 3-4%. For remote ARM-ISA profiling mode, the performance of SA-LEEF is 99% and 96% higher than Power Tutor in terms of energy estimation time and overhead, respectively. SA-LEEF has significantly reduced energy estimation time and overhead of existing energy estimation tools while offering acceptable accuracy for various application domains such as mobile cloud computing. However, the performance of SA-LEEF in terms of its estimation time and energy is badly affected if the wall clock execution time of applications is lower than 1.5s. In terms of resource consumption rate, SA-LEEF consumes 58% less CPU resource capacity than Power Tutor energy estimation tool during energy estimation process. For RAM usage, SA-LEEF requires up to 97% less smart-phone RAM during energy estimation of smart-phone applications.

7.2 Limitations and Applications

This research work is limited to the analysis of heaviness of existing dynamic analysis energy estimation methods and proposing a lightweight energy estimation framework to estimate energy consumption of the native smart-phone applications. SA-LEEF framework considers the static analysis of smart-phone application to minimize estimation time and energy overhead. The proposed framework is suitable for decision making in computational offloading frameworks for battery augmentation. Being a lightweight module, SA-LEEF can be integrated into SDK of a smart-phone application to empower smart-phone users to estimate energy consumption of their applications. Also, it is suitable for the external battery equipped small sized devices doing soft real-time decisions making. The main application areas for SA-LEEF includes MCC, IoTs, WSN, Body Area Network (BAN), and Internet of

University of Malaya

mobiles.

The scope of SA-LEEF is limited to native smart-phone applications only. It is applicable only to ARM processor as it has profiled operational energy cost of ARM-based assembly instructions only. In its current stand, it is applicable for energy estimation of those applications which does not require inputs from the users during their execution. Moreover, SA-LEEF lacks in considering the effect of smart-phone component's power states on estimation accuracy.

7.3 Contributions

In this section, the main contributions of this research are highlighted. A few contributions have already been published in scholarly articles as listed in publication section. Main contributions of this research are as follow.

- **Taxonomies of Energy Estimation Schemes:** This research has proposed thematic taxonomies to classify existing state-of-the-art smart-phone application energy estimation schemes. The proposed taxonomies highlight the main categories in this domain of research. It highlights critical aspects and significant features of existing categories of energy estimation schemes that ultimately lead to new emerging research directions.

- **Performance Evaluation of Dynamic Analysis Schemes:** A detailed analysis using a set of parameters on existing dynamic analysis based estimation schemes is performed to investigate their performance overhead. The performance evaluations reveal insights to the issues in existing energy estimation methods.

- **Fine Granular Instruction Energy Profile:** This study has estimated energy consumption of ARM-7 ISA instructions. It proposes energy cost for

University of Malaya

fetching ARM assembly instructions from RAM and local Cache. It has improved the estimation accuracy by applying weighted filter based neighborhood operation on power profile for an instruction.

- **Static Analysis based Energy Estimation Framework (SA-LEEF):** A lightweight static analysis based energy estimation framework is proposed. The proposed framework proposes weighted probability based application execution flow estimation. It also estimates storage location of instructions based on the cache distance between different chunks of the code. It estimates energy consumption based on application's base cost energy, user-system interaction model, ARM-IS energy profile access cost, and concurrent program execution energy overhead.

- **Evaluation and Validation of SA-LEEF:** This study validated the proposed energy estimation framework against empirical results. It chose four parameters to compare SA-LEEF with Power Tutor and measurement-based estimation methods. Chosen parameters include estimation time, energy overhead, resource consumption, and accuracy. It compared SA-LEEF with existing solutions while considering the different sizes of application code and data. It has also evaluated SA-LEEF to estimate its resource requirements in terms of its CPU and RAM requirements.

7.4 Future Works

The future research work includes extending SA-LEEF model to consider user system interaction model to estimate energy consumption of interactive web and mobile games applications. For gaming mobile applications, the extended version of this work will profile energy cost for graphics libraries. Also, it will model energy

University of Malaya

consumption while user interact with the smart-phone device. The user system interaction model is a complex system as the interaction time varies from person to person. The power consumption behavior of smart-phone components vary with load and type of operations within an application. The future research work will enable application analyzer module of SA-LEEF to statically predict the power state of smart-phone components based on the analysis of operations within a smart-phone application. SA-LEEF converts the native smart-phone application into its equivalent assembly code to analyze the application. To empower SA-LEEF for suppressing the conventional conversion process, a mapper can be proposed to efficiently generate assembly based instructions for each operation within high-level constructs of native smart-phone application. The compilers such as GCC applies optimization to minimize the code size for effective storage resource usage. The impact of compiler optimization on the performance of SA-LEEF in terms of estimation accuracy and estimation time is also part of future research. In its current stand, SA-LEEF has only implemented its remote profile mode. However, SA-LEEF local profile mode can be implemented to generate ARM-IS energy profile on local smart-phone based on the test programs.

MCC considers code size as one of the parameters to decide the execution location of the smart-phone application. In contrast to code size, energy consumption of an application highly depends on the type of operations within it. In future research, SA-LEEF can be integrated with MCC computational offloading frameworks to increase accuracy in decision making for execution location of the application.

University of Malaya

REFERENCES

Abolfazli, S., Sanaei, Z., Gani, A., Xia, F., & Yang, L. T. (2014). Rich mobile applications: genesis, taxonomy, and open issues. *Journal of Network and Computer Applications*, *40*, 345–362.

Ahmad, A., Paul, A., Rathore, M. M., & Rho, S. (2015). Power aware mobility management of m2m for iot communications. *Mobile Information Systems*, *2015*.

Ahmad, R. W., Gani, A., Hamid, S. H. A., Xia, F., & Shiraz, M. (2015). A review on mobile application energy profiling: Taxonomy, state-of-the-art, and open research issues. *Journal of Network and Computer Applications*, *58*, 42–59.

Aleti, A., Buhnova, B., Grunske, L., Koziolek, A., & Meedeniya, I. (2013). Software architecture optimization methods: A systematic literature review. *IEEE Transactions on Software Engineering*, *39*, 658-683.

Allcott, H., & Greenstone, M. (2012). *Is there an energy efficiency gap?* (Report No. 17766). National Bureau of Economic Research.

Anand, A., Manikopoulos, C., Jones, Q., & Borcea, C. (2007). A quantitative analysis of power consumption for location-aware applications on smart phones. In *Ieee international symposium on industrial electronics* (pp. 1986–1991).

Autili, M., Cortellessa, V., Di Benedetto, P., & Inverardi, P. (2015). On the adaptation of context-aware services. *arXiv preprint arXiv:1504.07558*.

Ayewah, N., Hovemeyer, D., Morgenthaler, J. D., Penix, J., & Pugh, W. (2008). Using static analysis to find bugs. *IEEE software*, *25*(5).

Bandyopadhyay, A., Chakraborty, K., Bag, R., & Das, A. (2016). High density salt and pepper noise removal by selective mean filter. In *Foundations and frontiers in computer, communication and electrical engineering: Proceedings of 3rd international conference c2e2, mankundu, west bengal, india, 15th-16th january, 2016.* (p. 191).

Barroso, L. A., & Hölzle, U. (2007). The case for energy-proportional computing. *Computer*(12), 33–37.

Batyuk, L., Schmidt, A.-D., Schmidt, H.-G., Camtepe, A., & Albayrak, S. (2009). Developing and benchmarking native linux applications on android. In *International conference on mobile wireless middleware, operating systems, and applications* (pp. 381–392).

Bazzaz, M., Salehi, M., & Ejlali, A. (2013). An accurate instruction-level energy estimation model and tool for embedded systems. *IEEE Transactions on Instrumentation and Measurement,*, *62*(7), 1927–1934.

Ben-Zur, L. (2011). *Developer tool spotlight-using trepn profiler for power-efficient*

University of Malaya

apps.

Bhattacharya, S., Blunck, H., Kjærgaard, M. B., & Nurmi, P. (2015). Robust and energy-efficient trajectory tracking for mobile devices. *IEEE Transactions on Mobile Computing,*, *14*(2), 430–443.

Blazy, S., Maroneze, A., & Pichardie, D. (2013). Formal verification of loop bound estimation for wcet analysis. In *Verified software: Theories, tools, experiments* (pp. 281–303). Springer.

Blem, E., Menon, J., Vijayaraghavan, T., & Sankaralingam, K. (2015). Isa wars: Understanding the relevance of isa being risc or cisc to performance, power, and energy on modern architectures. *ACM Transactions on Computer Systems (TOCS)*, *33*(1), 3.

Brandolese, C., Fornaciari, W., Salice, F., & Sciuto, D. (2000). An instruction-level functionally-based energy estimation model for 32-bits microprocessors. In *Proceedings of the 37th annual design automation conference* (pp. 346–351).

C, A. D. (n.d.). Architecture reference manual. armv7-a and armv7-r edition [Computer software manual].

Carroll, A., & Heiser, G. (2010). An analysis of power consumption in a smartphone. In *Usenix annual technical conference* (Vol. 14).

Chaffey, D. (2016, April). *Mobile marketing statistics compilation.* online. Retrieved from www.smartinsights.com/mobile-marketing/mobile-marketing-analytics/mobile-marketing-statistics/

Chang, N., Kim, K., & Lee, H. G. (2002). Cycle-accurate energy measurement and characterization with a case study of the arm7tdmi [microprocessors]. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,*, *10*(2), 146–154.

Chang, W.-Y. (2013). The state of charge estimating methods for battery: a review. *ISRN Applied Mathematics*, *2013*.

Chen, R. Y., Irwin, M. J., & Bajwa, R. S. (1998). Architectural level hierarchical power estimation of control units. In *Proceedings of eleventh annual ieee international asic conference 1998.* (pp. 211–215).

Cornelissen, B., Zaidman, A., Van Deursen, A., Moonen, L., & Koschke, R. (2009). A systematic survey of program comprehension through dynamic analysis. *IEEE Transactions on Software Engineering*, *35*(5), 684–702.

Dementyev, A., Hodges, S., Taylor, S., & Smith, J. (2013). Power consumption analysis of bluetooth low energy, zigbee and ant sensor nodes in a cyclic sleep scenario. In *Wireless symposium (iws), 2013 ieee international* (pp. 1–4).

Ding, F., Xia, F., Zhang, W., Zhao, X., & Ma, C. (2011). Monitoring energy consumption of smartphones. In *Internet of things (ithings/cpscom), 2011 in-*

University of Malaya

ternational conference on and 4th international conference on cyber, physical and social computing (pp. 610–613).

Do, T., Rawshdeh, S., & Shi, W. (2009). ptop: A process-level power profiling tool. In *Proceedings of the 2nd workshop on power aware computing and systems (hotpower'09).*

Dong, G., Chen, Z., Wei, J., Zhang, C., & Wang, P. (2016). An online model-based method for state of energy estimation of lithium-ion batteries using dual filters. *Journal of Power Sources*, *301*, 277–286.

Dong, M., & Zhong, L. (2010). Sesame: Self-constructive system energy modeling for battery-powered mobile systems. *arXiv preprint arXiv:1012.2831*.

Emanuelsson, P., & Nilsson, U. (2008). A comparative study of industrial static analysis tools. *Electronic notes in theoretical computer science*, *217*, 5–21.

Ermedahl, A., Sandberg, C., Gustafsson, J., Bygde, S., & Lisper, B. (2007). Loop bound analysis based on a combination of program slicing, abstract interpretation, and invariant analysis. In *Oasics-openaccess series in informatics* (Vol. 6).

Ferry, D. K. (2012). Ohm's law in a quantum world. *Science*, *335*(6064), 45–46.

Filina, M. V., & Zubkov, A. M. (2008). Exact computation of pearson statistics distribution and some experimental results. *Austrian Journal of Statistics*, *37*(1), 129–135.

Fischer, B. (2012, Feburary). *How much energy a smartphone uses in a year (and what it means for your budget).* Online. Retrieved from lifehacker.com/5948075/how-much-energy-a-smartphone-uses-in-a-year-and-what-it-means-for-your-budget

Gavalas, D., & Economou, D. (2011). Development platforms for mobile applications: Status and trends. *Software, IEEE*, *28*(1), 77–86.

Gholizadeh, M., & Salmasi, F. R. (2014). Estimation of state of charge, unknown nonlinearities, and state of health of a lithium-ion battery based on a comprehensive unobservable model. *IEEE Transactions on Industrial Electronics,*, *61*(3), 1335–1344.

Gosain, A., & Sharma, G. (2014). A survey of dynamic program analysis techniques and tools. In *Ficta (1)* (pp. 113–122).

Grund, D. (2012). *Static cache analysis for real-time systems: Lru, fifo, plru.* epubli.

Guan, N., Yang, X., Lv, M., & Yi, W. (2013). Fifo cache analysis for wcet estimation: a quantitative approach. In *Design, automation & test in europe conference & exhibition (date), 2013* (pp. 296–301).

Hamzaoui, K. I., Benzekri, W., Grimaud, G., Berrajaa, M., & Azizi, M. (2016). Esti-

University of Malaya

mation and optimization of energy consumption on smartphones. In *Proceedings of the mediterranean conference on information & communication technologies 2015* (pp. 333–342).

Hans, R., Burgstahler, D., Mueller, A., Zahn, M., & Stingl, D. (2015). Knowledge for a longer life: Development impetus for energy-efficient smartphone applications. In *Ieee international conference on mobile services (ms), 2015* (pp. 128–133).

Hao, S., Li, D., Halfond, W. G., & Govindan, R. (2012a). Estimating android applications' cpu energy usage via bytecode profiling. In *Proceedings of the first international workshop on green and sustainable software* (pp. 1–7).

Hao, S., Li, D., Halfond, W. G., & Govindan, R. (2012b). Estimating android applications' cpu energy usage via bytecode profiling. In *Proceedings of the first international workshop on green and sustainable software* (pp. 1–7).

Hao, S., Li, D., Halfond, W. G., & Govindan, R. (2013). Estimating mobile application energy consumption using program analysis. In *35th international conference on software engineering (icse), 2013* (pp. 92–101).

Hardy, D., Puaut, I., & Sazeides, Y. (2016). Probabilistic wcet estimation in presence of hardware for mitigating the impact of permanent faults. In E. Consortium (Ed.), *Design, automation and test in europe.*

Harizopoulos, S., & Ailamaki, A. (2004). Steps towards cache-resident transaction processing. In *Proceedings of the thirtieth international conference on very large data bases-volume 30* (pp. 660–671).

Hasan, S., King, Z., Hafiz, M., Sayagh, M., Adams, B., & Hindle, A. (2016). Energy profiles of java collections classes. In *Proceedings of the 38th international conference on software engineering* (pp. 225–236).

He, H., Zhang, X., Xiong, R., Xu, Y., & Guo, H. (2012). Online model-based estimation of state-of-charge and open-circuit voltage of lithium-ion batteries in electric vehicles. *Energy*, *39*(1), 310–318.

He, Y., Liu, X., Zhang, C., & Chen, Z. (2013). A new model for state-of-charge (soc) estimation for high-power li-ion batteries. *Applied Energy*, *101*, 808–814.

Hohl, W., & Hinds, C. (2016). *Arm assembly language: Fundamentals and techniques* (W. Hohl, Ed.). Crc Press.

Hoque, M. A., Siekkinen, M., Khan, K. N., Xiao, Y., & Tarkoma, S. (2015). Modeling, profiling, and debugging the energy consumption of mobile devices. *ACM Computing Surveys (CSUR)*, *48*(3), 39.

Hu, C., Youn, B. D., & Chung, J. (2012). A multiscale framework with extended kalman filter for lithium-ion battery soc and capacity estimation. *Applied Energy*, *92*, 694–704.

Jabbarvand, R., Sadeghi, A., Garcia, J., Malek, S., & Ammann, P. (2015). Ecodroid:

University of Malaya

an approach for energy-based ranking of android apps. In *Proceedings of the fourth international workshop on green and sustainable software* (pp. 8–14).

Jayaseelan, R., Mitra, T., & Li, X. (2006). Estimating the worst-case energy consumption of embedded software. In *Proceedings of the 12th ieee real-time and embedded technology and applications symposium, 2006.* (pp. 81–90).

Jiang, C., Taylor, A., Duan, C., & Bai, K. (2013). Extended kalman filter based battery state of charge (soc) estimation for electric vehicles. In *Ieee transportation electrification conference and expo (itec), 2013* (pp. 1–5).

Jiang, F., Zarepour, E., Hassan, M., Seneviratne, A., & Mohapatra, P. (2015). Type, talk, or swype: Characterizing and comparing energy consumption of mobile input modalities. *Pervasive and Mobile Computing*, *26*, 57-70.

Joos, O., Silva, R., Amouzou, A., Moulton, L. H., Perin, J., Bryce, J., & Mullany, L. C. (2016). Evaluation of a mhealth data quality intervention to improve documentation of pregnancy outcomes by health surveillance assistants in malawi: A cluster randomized trial. *PloS one*, *11*(1). (e0145238)

Khan, A. R., Othman, M., Madani, S. A., & Khan, S. U. (2014). A survey of mobile cloud computing application models. *Communications Surveys & Tutorials, IEEE*, *16*(1), 393–413.

Khan, A. U. R., Othman, M., Xia, F., & Khan, A. N. (2015). Context-aware mobile cloud computing and its challenges. *Cloud Computing, IEEE*, *2*(3), 42–49.

Khoshbakht, S., & Dimopoulos, N. (2015). Investigating the effects of store value locality on processor power. In *Ieee pacific rim conference on communications, computers and signal processing (pacrim), 2015* (pp. 338–343).

Kjærgaard, M. B., & Blunck, H. (2011). Unsupervised power profiling for mobile devices. In *Mobile and ubiquitous systems: Computing, networking, and services* (pp. 138–149). Springer.

Kong, J. (2015). Variable latency l1 data cache architecture design in multi-core processor under process variation. *Journal of the Korea Society of Computer and Information*, *20*(9), 1–10.

Konstantakos, V., Chatzigeorgiou, A., Nikolaidis, S., & Laopoulos, T. (2008). Energy consumption estimation in embedded systems. *IEEE Transactions on Instrumentation and Measurement,*, *57*(4), 797–804.

Koudounas, V. (n.d.). *Mobile computing: Past, present and future.* Online. Retrieved from `www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/vk5/report.html`

Li, D., Hao, S., Halfond, W. G., & Govindan, R. (2013). Calculating source line level energy information for android applications. In *Proceedings of the 2013 international symposium on software testing and analysis* (pp. 78–89).

University of Malaya

Li, X., & Gallagher, J. P. (2016). An energy-aware programming approach for mobile application development guided by a fine-grained energy model. *arXiv preprint arXiv:1605.05234*, *59*, 46-54.

Lisper, B. (2014). Sweet–a tool for wcet flow analysis. In *International symposium on leveraging applications of formal methods, verification and validation* (pp. 482–485).

Lu, L., Han, X., Li, J., Hua, J., & Ouyang, M. (2013). A review on the key issues for lithium-ion battery management in electric vehicles. *Journal of power sources*, *226*, 272–288.

Lu, Q., Wu, T., Yan, J., Yan, J., Ma, F., & Zhang, F. (2016). Lightweight method-level energy consumption estimation for android applications. In *10th international symposium on theoretical aspects of software engineering (tase), 2016* (pp. 144–151).

Mandeep, K. (2008). *Cenzic application security trends report-q4, 2007, cenzic inc. 2008. whitepaper.*

Maroneze, A., Blazy, S., Pichardie, D., & Puaut, I. (2014). A formally verified wcet estimation tool. In *Oasics-openaccess series in informatics* (Vol. 39).

Mehta, H., Owens, R. M., & Irwin, M. J. (1996). Instruction level power profiling. In *Ieee conference proceedings of international conference on acoustics, speech, and signal processing, icassp-96. , 1996* (Vol. 6, pp. 3326–3329).

Monte, T. (2010). Energy debugging tools for embedded applications. *Simplicity Studio*, *2*.

Murmuria, R., Stavrou, A., Barbará, D., & Fleck, D. (2015). Continuous authentication on mobile devices using power consumption, touch gestures and physical movement of users. In *Research in attacks, intrusions, and defenses* (pp. 405–424). Springer.

Neglia, G., Carra, D., Feng, M., Janardhan, V., Michiardi, P., & Tsigkari, D. (2016). Access-time aware cache algorithms. In *Teletraffic congress (itc 28), 2016 28th international* (Vol. 1, pp. 148–156).

Nimmer, J. W., & Ernst, M. D. (2001). Static verification of dynamically detected program invariants: Integrating daikon and esc/java. *Electronic Notes in Theoretical Computer Science*, *55*(2), 255–276.

Noguchi, H., Ikegami, K., Takaya, S., Arima, E., Kushida, K., Kawasumi, A., ... others (2016). 7.2 4mb stt-mram-based cache with memory-access-aware power optimization and write-verify-write/read-modify-write scheme. In *2016 ieee international solid-state circuits conference (isscc)* (pp. 132–133).

Oliner, A. J., Iyer, A. P., Stoica, I., Lagerspetz, E., & Tarkoma, S. (2013). Carat: Collaborative energy diagnosis for mobile devices. In *Proceedings of the 11th acm conference on embedded networked sensor systems* (p. 10).

University of Malaya

Pang, C., Hindle, A., Adams, B., & Hassan, A. E. (2015). What do programmers know about the energy consumption of software? *PeerJ PrePrints*, *3*, e1094.

Pascu, T., White, M., Beloff, N., Patoli, Z., & Barker, L. (2016). Ambient health monitoring: The smartphone as a body sensor network component. *InImpact: The Journal of Innovation Impact*, *6*(1), 62.

Pathak, A., Hu, Y. C., & Zhang, M. (2011). Bootstrapping energy debugging on smartphones: a first look at energy bugs in mobile devices. In *Proceedings of the 10th acm workshop on hot topics in networks* (Vol. 14, p. 5).

PCs, U.-M. (2008). Emerging technologies mobile-computing trends: Lighter, faster, smarter. *About Language Learning & Technology*, *3*(12), 3-9.

Pejovic, V., & Musolesi, M. (2015). Anticipatory mobile computing: A survey of the state of the art and research challenges. *ACM Computing Surveys (CSUR)*, *47*(3), 47.

Peltonen, E., Lagerspetz, E., Nurmi, P., & Tarkoma, S. (2015). Energy modeling of system settings: A crowdsourced approach. In *Ieee international conference on pervasive computing and communications (percom), 2015* (pp. 37–45).

Peltonen, E., Lagerspetz, E., Nurmi, P., & Tarkoma, S. (2016). Where has my battery gone?: A novel crowdsourced solution for characterizing energy consumption. *Pervasive Computing, IEEE*, *15*(1), 6–9.

Pistoia, M., Chandra, S., Fink, S. J., & Yahav, E. (2007). A survey of static analysis methods for identifying security vulnerabilities in software systems. *IBM Systems Journal*, *46*(2), 265–288.

Plaza, I., MartíN, L., Martin, S., & Medrano, C. (2011). Mobile applications in an aging society: Status and trends. *Journal of Systems and Software*, *84*(11), 1977–1988.

Qian, H., & Andresen, D. (2015). An energy-saving task scheduler for mobile devices. In *Ieee/acis 14th international conference on computer and information science (icis), 2015* (pp. 423–430).

Rajan, A., Noureddine, A., & Stratis, P. (2016). A study on the influence of software and hardware features on program energy. In *Proceedings of the 10th acm/ieee international symposium on empirical software engineering and measurement* (p. 37).

Ren, R., Juarez, E., Sanz, C., Raulet, M., & Pescador, F. (2015). Energy estimation models for video decoders: reconfigurable video coding-cal case-study. *IET Computers & Digital Techniques*, *9*(1), 3–15.

Rice, A., & Hay, S. (2010). Measuring mobile phone energy consumption for 802.11 wireless networking. *Pervasive and Mobile Computing*, *6*(6), 593–606.

Rice, A. C., & Hay, S. (2010). Decomposing power measurements for mobile devices.

University of Malaya

In *Percom* (Vol. 10, pp. 70–78).

Sepasi, S., Ghorbani, R., & Liaw, B. Y. (2014). A novel on-board state-of-charge estimation method for aged li-ion batteries based on model adaptive extended kalman filter. *Journal of Power Sources*, *245*, 337–344.

Seward, J., Nethercote, N., & Fitzhardinge, J. (2004). *Cachegrind: a cache-miss profiler.*

Shao, Y. S., & Brooks, D. (2013). Energy characterization and instruction-level energy model of intel's xeon phi processor. In *Proceedings of the 2013 international symposium on low power electronics and design* (pp. 389–394).

Shin, D., Kim, K., Chang, N., Lee, W., Wang, Y., Xie, Q., & Pedram, M. (2013). Online estimation of the remaining energy capacity in mobile systems considering system-wide power consumption and battery characteristics. In *Design automation conference (asp-dac), 2013 18th asia and south pacific* (pp. 59–64).

Sinha, A., Ickes, N., & Chandrakasan, A. P. (2003). Instruction level and operating system profiling for energy exposed software. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,*, *11*(6), 1044–1057.

Stallings, W. (2000). *Computer organization and architecture: designing for performance.* Pearson Education India.

Stanley-Marbell, P., & Hsiao, M. (2001). Fast, flexible, cycle-accurate energy estimation. In *Proceedings of the 2001 international symposium on low power electronics and design* (pp. 141–146).

Sun, W.-T., & Cassé, H. (2016). Dynamic branch resolution based on combined static analyses. In *Oasics-openaccess series in informatics* (Vol. 55).

Thiagarajan, N., Aggarwal, G., Nicoara, A., Boneh, D., & Singh, J. P. (2012). Who killed my battery?: analyzing mobile browser energy consumption. In *Proceedings of the 21st international conference on world wide web* (pp. 41–50).

Thornton, P., & Houser, C. (2005). Using mobile phones in english education in japan. *Journal of computer assisted learning*, *21*(3), 217–228.

Tiwari, V., Malik, S., & Wolfe, A. (1994). Power analysis of embedded software: a first step towards software power minimization. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,*, *2*(4), 437–445.

Tsao, S.-L., Kao, C.-C., Suat, I., Kuo, Y., Chang, Y.-H., & Yu, C.-K. (2012). Powermemo: a power profiling tool for mobile devices in an emulated wireless environment. In *International symposium on system on chip (soc), 2012* (pp. 1–5).

Vasilakis, E. (2015). *An instruction level energy characterization of arm processors* (Unpublished doctoral dissertation). Department of Computer Science,

University of Malaya

University of Crete.

Wang, W., Nevatia, R., & Yang, B. (2015). Beyond pedestrians: A hybrid approach of tracking multiple articulating humans. In *Ieee winter conference on applications of computer vision (wacv), 2015* (pp. 132–139).

Watson, G. (2004). *Dmalloc–debug malloc library.*

Weiss, A. R. (2002). Dhrystone benchmark: History, analysis, scores and recommendations. *EEMBC White Paper*.

Wu, J.-H., & Wang, S.-C. (2005). What drives mobile commerce?: An empirical evaluation of the revised technology acceptance model. *Information & management*, *42*(5), 719–729.

Yetim, Y., Malik, S., & Martonosi, M. (2012). Eprof: An energy/performance/reliability optimization framework for streaming applications. In *17th asia and south pacific design automation conference (asp-dac), 2012* (pp. 769–774).

Zhang, L., Tiwana, B., Qian, Z., Wang, Z., Dick, R. P., Mao, Z. M., & Yang, L. (2010a). Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In *Proceedings of the eighth ieee/acm/ifip international conference on hardware/software codesign and system synthesis* (pp. 105–114).

Zhang, L., Tiwana, B., Qian, Z., Wang, Z., Dick, R. P., Mao, Z. M., & Yang, L. (2010b). Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In *Proceedings of the eighth ieee/acm/ifip international conference on hardware/software codesign and system synthesis* (pp. 105–114).

Zhang, W., Wen, Y., Wu, J., & Li, H. (2013). Toward a unified elastic computing platform for smartphones with cloud support. *IEEE Network*, *27*(5), 34–40.

Zhao, X., Guo, Y., Wang, H., & Chen, X. (2008). Fine-grained energy estimation and optimization of embedded operating systems. In *International conference on embedded software and systems symposia, 2008. icess symposia'08.* (pp. 90–95).

Zhong, L., Zhang, C., He, Y., & Chen, Z. (2014). A method for the estimation of the battery pack state of charge based on in-pack cells uniformity analysis. *Applied Energy*, *113*, 558–564.

Zhou, X., Guo, B., Shen, Y., & Li, Q. (2009). Design and implementation of an improved c source-code level program energy model. In *International conference on embedded software and systems, 2009. icess'09.* (pp. 490–495).

University of Malaya

LIST OF PUBLICATIONS

Ahmad, Raja Wasim, Abdullah Gani, Siti Hafizah Ab Hamid, Feng Xia, and Muhammad Shiraz. "A Review on mobile application energy profiling: Taxonomy, state-of-the-art, and open research issues." Journal of Network and Computer Applications 58 (2015): 42-59.

Ahmad, Raja Wasim, Abdullah Gani, Siti Hafizah Ab Hamid, Anjum Naveed, "A Case and Framework for Smart-phone Application Energy Estimation using Code Analysis." International Journal of Communication Systems (Accepted).

Ahmad, Raja Wasim, Abdullah Gani, Siti Hafizah Ab Hamid, " A survey of Power modeling and energy estimation schemes for smart-phone applications", International Journal of Communication Systems (Accepted)

Ahmad, Raja Wasim, Abdullah Gani, Siti Hafizah Ab Hamid, Muhammad Shiraz, Abdullah Yousafzai, and Feng Xia. "A survey on virtual machine migration and server consolidation frameworks for cloud data centers." Journal of Network and Computer Applications 52 (2015): 11-25.

Ahmad, Raja Wasim, Abdullah Gani, Siti Hafizah Ab Hamid, Muhammad Shiraz, Feng Xia, and Sajjad A. Madani. "Virtual machine migration in cloud data centers: a review, taxonomy, and open research issues." The Journal of Supercomputing 71, no. 7 (2015): 2473-2515.

Shiraz, Muhammad, Abdullah Gani, Azra Shamim, Suleman Khan, and **Raja Wasim Ahmad**. "Energy efficient computational offloading framework for mobile cloud computing." Journal of Grid Computing 13, no. 1 (2015): 1-18.

Shiraz, Muhammad, Abdullah Gani, **Raja Wasim Ahmad**, Syed Adeel Ali Shah, Ahmad Karim, and Zulkanain Abdul Rahman. "A lightweight distributed frame-

University of Malaya

work for computational offloading in mobile cloud computing." PloS one 9, no. 8 (2014): e102270.

Yousafzai, Abdullah, Abdullah Gani, Rafidah Md Noor, Anjum Naveed, **Raja Wasim Ahmad**, and Victor Chang. "Computational Offloading Mechanism for Native and Android Runtime based Mobile Applications." Journal of Systems and Software (2016).

Shuja, Junaid, Abdullah Gani, Shahaboddin Shamshirband, **Raja Wasim Ahmad**, and Kashif Bilal. "Sustainable Cloud Data Centers: A survey of enabling techniques and technologies." Renewable and Sustainable Energy Reviews 62 (2016): 195-214.

Ahmad, Raja Wasim, Sajjad A. Madani, Kashif Bilal, Junaid Shuja. "An Investigation of Video Communication Over Public Safety Network" The Malaysian Journal of Computer Science (Accepted).

University of Malaya

APPENDIX A: DATASET NOISE REMOVAL

Generally, there are various types noise removal approaches exists in literature. We normalize data by removing the noise from dataset (XYZ) inspired by image processing noise removal mean filtering method[1].

Algorithm 3 has presented a Mean Filter for removal of noise. Mean filtering (MF) is intuitive and easy to implement method of smoothing data. It reduces the amount of intensity variation between one data element and the next by simply mean ('average') value of its neighbors, including itself. This process eliminates the data element which is beyond the representation of their neighbors. The proposed pseudo code of applying MF over target data-set is shown below.

Algorithm 3 Noise Removal

```
1: Input: Grayscale Image
2: $SumKernal \leftarrow 0$
3: for $X = WL$ to $l - WL$ do
4:    $i \leftarrow 0$
5:    for $fx = 0$ to $WL - 1$ do
6:       $SumKernal += InputDataValue[x + fx + WL]$
7:    end for
8:    $i += 1$
9:    $AvgKernal = SumKernla/i$
10:   $OutputDataElement[x] = floor(AvgKernal)$
11: end for
```

For evaluation of propose noise removal, we have considered a case study as given below. Assume the inputDataValue = $[11, 12, 9, 8, 13, 8, 12, 109, 7, 11, 12, 190]$. Where L represent the length of inputDataValue, i.e. L = 14, and similarly the WL represent the window length i.e. WL = 8. Assume in the 8th element of inputDataValue is 109, considered the noise element. From step 2 to 7, we observed if x = 1, the SumKernal = 182 with inputDataValue $[11+12+9+8+13+8+12+109] = 182$, and AvgKernal = 22.75, So the inputDataValue 109 value would be

[1]http://homepages.inf.ed.ac.uk/rbf/HIPR2/mean.htm

University of Malaya

replaced by floored 22. And similarly the above process will be applied for other noise elements in data sets.

University of Malaya

APPENDIX B: TEST PROGRAM DESIGN

This section presents the design of the test program that is considered to estimate energy consumption for a single ARM based assembly instruction. The test program when compiled outputs the assembly based sequence for an ARM instructions.

```
#include <stdio.h>
#include <stdlib.h>
int main (int arg, char *argv[])
{
FILE *fp;
fp=fopen("CSTORE.s", "w");
fprintf(fp," .arch armv7t\n");
fprintf(fp," .fpu softvfp");
fprintf(fp," .eabi_attribute 20, 1 \n")
fprintf(fp," .eabi_attribute 21, 1\n")
fprintf(fp," .eabi_attribute 23, 3\n")
fprintf(fp," .eabi_attribute 24, 1\n")
fprintf(fp," .eabi_attribute 25, 1\n")
fprintf(fp," .eabi_attribute 26, 2\n")
fprintf(fp," .eabi_attribute 30, 6\n")
fprintf(fp," .eabi_attribute 34, 0\n")
fprintf(fp," .eabi_attribute 18, 4\n")
fprintf(fp," .file p̈pc1.c¨\n");
fprintf(fp," .text\n");
fprintf(fp," .align 2\n");
fprintf(fp," .global main\n");
```

University of Malaya

```
fprintf(fp,"main:\n");
fprintf(fp,"  args = 0, pretend = 0, frame = 24\n");
fprintf(fp,"  frame_needed = 1, uses_anonymous_args = 0\n");
fprintf(fp," str fp, [sp, #-4]\n");
fprintf(fp," add fp, sp, #0\n");
fprintf(fp," sub sp, sp, #28\n");
fprintf(fp," str r0, [fp, #-24]\n");
fprintf(fp," str r1, [fp, #-28]\n");
fprintf(fp," mov r3, #5\n");
fprintf(fp," str r3, [fp, #-12]\n");
fprintf(fp," mov r3, #0\n");
fprintf(fp," str r3, [fp, #-16]\n");
fprintf(fp," ldr r3, [fp, #-12]\n");
fprintf(fp," mov r3, #0\n");
fprintf(fp," str r3, [fp, #-12]\n");
fprintf(fp," ldr r2, [fp, #-16]\n");
int j,k;
for (i=0;i<1000;i++)
{
for(j=0;j<1521;j++)
{
for(k=0;k<10;k++)
{
n=rand()%4096;
if (n>4000)
{n=3471;}
```

University of Malaya

```
//fprintf(fp," add r3, r3, 1\n");
n=rand()
if (n>4000)
{n=3471;}
fprintf(fp," str r3, [fp, -%d]\n",n);
n=rand()%4096;
if (n>4000)
{n=3471;}
fprintf(fp," str r3, [fp, -%d]\n",n);
n=rand()%4096;
if (n>4000)
{n=3471;}
fprintf(fp," str r3, [fp, -%d]\n",n);
n=rand()%4096;
if (n>4000)
{n=3471;}
fprintf(fp," str r3, [fp, -%d]\n",n);
n=rand()%4096;
if (n>4000)
{n=3471;}
fprintf(fp," str r3, [fp, -%d]\n",n);
n=rand()%4096;
if (n>4000)
{n=3471;}
fprintf(fp," str r3, [fp, -%d]\n",n);
n=rand()%4096;
```

University of Malaya

```
if (n>4000)

{n=3471;}

fprintf(fp," str r3, [fp, -%d]\n",n);

n=rand()%4096;

if (n>4000)

{n=3471;}

fprintf(fp," str r3, [fp, -%d]\n",n);

n=rand()%4096;

if (n>4000)

{n=3471;}

fprintf(fp," str r3, [fp, -%d]\n",n);

n=rand()%4096;

if (n>4000)

{n=3471;}

fprintf(fp," str r3, [fp, -%d]\n",n);

n=rand()%4096;

if (n>4000)

{n=3471;}

fprintf(fp," str r3, [fp, -%d]\n",n);

n=rand()%4096;

if (n>4000)

{n=3471;}

fprintf(fp," str r3, [fp, -%d]\n",n);

n=rand()%4096;

if (n>4000)

{n=3471;}
```

University of Malaya

```
fprintf(fp," str r3, [fp, -%d]\n",n);

n=rand()%4096;

if (n>4000)

{n=3472;}

fprintf(fp," str r3, [fp, -%d]\n",n);

n=rand()%4096;

if (n>4000)

{n=3470;}

fprintf(fp," str r3, [fp, -%d]\n",n);

n=rand()%4096;

if (n>4000)

{n=3471;}

fprintf(fp," str r3, [fp, -%d]\n",n);

n=rand()%4096;

if (n>4000)

{n=3321;}

fprintf(fp," str r3, [fp, -%d]\n",n);

n=rand()%4096;

if (n>4000)

{n=3371;}

fprintf(fp," str r3, [fp, -%d]\n",n);

n=rand()%4096;

if (n>4000)

{n=3456;}

fprintf(fp," str r3, [fp, -%d]\n",n);

n=rand()%4096;
```

University of Malaya

```
if (n>4000)

{n=3471;}

fprintf(fp," str r3, [fp, -%d]\n",n);

n=rand()%4096;

if (n>4000)

{n=3171;}

fprintf(fp," str r3, [fp, -%d]\n",n);

n=rand()%4096;

if (n>4000)

{n=3131;}

fprintf(fp," str r3, [fp, -%d]\n",n);

n=rand()%4096;

if (n>4000)

{n=3471;}

fprintf(fp," str r3, [fp, -%d]\n",n);

n=rand()%4096;

if (n>4000)

{n=3411;}

fprintf(fp," str r3, [fp, -%d]\n",n);

n=rand()%4096;

if (n>4000)

{n=3471;}

fprintf(fp," str r3, [fp, -%d]\n",n);

n=rand()%4096;

if (n>4000)

{n=3401;}
```

University of Malaya

```
fprintf(fp," str r3, [fp, -%d]\n",n);

n=rand()%4096;

if (n>4000)

{n=3470;}

fprintf(fp," str r3, [fp, -%d]\n",n);

n=rand()%4096;

if (n>4000)

{n=3471;}

fprintf(fp," str r3, [fp, -%d]\n",n);

n=rand()%4096;

if (n>4000)

{n=3471;}

fprintf(fp," str r3, [fp, -%d]\n",n);

n=rand()%4096;

if (n>4000)

{n=3271;}

fprintf(fp," str r3, [fp, -%d]\n",n);

n=rand()%4096;

if (n>4000)

{n=3471;}

fprintf(fp," str r3, [fp, -%d]\n",n);

n=rand()%4096;

if (n>4000)

{n=3471;}

fprintf(fp," str r3, [fp, -%d]\n",n);

n=rand()%4096;
```

University of Malaya

```
if (n>4000)

{n=3471;}

fprintf(fp," str r3, [fp, -%d]\n",n);

n=rand()%4096;

if (n>4000)

{n=3471;}

fprintf(fp," str r3, [fp, -%d]\n",n);

n=rand()%4096;

if (n>4000)

{n=3471;}

fprintf(fp," str r3, [fp, -%d]\n",n);

n=rand()%4096;

if (n>4000)

{n=3471;}

fprintf(fp," str r3, [fp, -%d]\n",n);

n=rand()%4096;

if (n>4000)

{n=3471;}

fprintf(fp," str r3, [fp, -%d]\n",n);

n=rand()%4096;

if (n>4000)

{n=3456;}

fprintf(fp," str r3, [fp, -%d]\n",n);

n=rand()%4096;

if (n>4000)

{n=3471;}
```

University of Malaya

```
fprintf(fp," str r3, [fp, -%d]\n",n);

n=rand()%4096;

if (n>4000)

{n=3471;}

fprintf(fp," str r3, [fp, -%d]\n",n);

n=rand()%4096;

if (n>4000)

{n=3471;}

fprintf(fp," str r3, [fp, -%d]\n",n);

n=rand()%4096;

if (n>4000)

{n=3471;}

fprintf(fp," str r3, [fp, -%d]\n",n);

n=rand()%4096;

if (n>4000)

{n=3471;}

fprintf(fp," str r3, [fp, -%d]\n",n);

n=rand()%4096;

if (n>4000)

{n=3471;}

fprintf(fp," str r3, [fp, -%d]\n",n);

n=rand()%4096;

if (n>4000)

{n=3471;}

fprintf(fp," str r3, [fp, -%d]\n",n);

n=rand()%4096;
```

University of Malaya

```
if (n>4000)
{n=3471;}
fprintf(fp," str r3, [fp, -%d]\n",n);
n=rand()%4096;
if (n>4000)
{n=3471;}
fprintf(fp," str r3, [fp, -%d]\n",n);
n=rand()%4096;
if (n>4000)
{n=3471;}
fprintf(fp," str r3, [fp, -%d]\n",n);
n=rand()%4096;
if (n>4000)
{n=3471;}
fprintf(fp," str r3, [fp, -%d]\n",n);
n=rand()%4096;
if (n>4000)
{n=3471;}
fprintf(fp," str r3, [fp, -%d]\n",n);
n=rand()%4096;
if (n>4000)
{n=3471;}
fprintf(fp," str r3, [fp, -%d]\n",n);
}
}
fprintf(fp," sub sp, fp, #4\n");
```

University of Malaya

```
fprintf(fp," ldmfd sp!, {fp, pc}\n");

fprintf(fp," .size main, .-main\n");

fprintf(fp," .ident "GCC: (Ubuntu/Linaro 4.7.3-12ubuntu1) 4.7.3¨\n");

fprintf(fp," .section .note.GNU-stack,"",%%progbits\n");

fclose(fp);

return 0;

}
```

University of Malaya