1-1-2016

GreenC5: An Adaptive, Energy-Aware Collection for Green Software Development

Junya Michanan
University of Denver

GREENC5: AN ADAPTIVE, ENERGY-AWARE COLLECTION FOR GREEN SOFTWARE DEVELOPMENT

_____

A Dissertation

Presented to

the Faculty of the Daniel Felix Ritchie School of Engineering and Computer Science

University of Denver

_____

In Partial Fulfillment

of the Requirements for the Degree

Doctor of Philosophy

_____

by

Junya Michanan

June 2016

Advisors: Matthew J. Rutherford and Rinku Dewri

Author: Junya Michanan
Title: GREENC5: AN ADAPTIVE, ENERGY-AWARE COLLECTION FOR GREEN SOFTWARE DEVELOPMENT
Advisors: Matthew J. Rutherford and Rinku Dewri
Degree Date: June 2016

ABSTRACT

Dynamic data structures in software applications have been shown to have a large impact on system performance. In this paper, we explore energy saving opportunities of interface-based dynamic data structures. Our results suggest that savings opportunities exist in the C5 Collection between 16.95% and 97.50%. We propose a prototype and architecture for creating adaptive green data structures by applying machine learning tools to build a model for predicting energy efficient data structures based on the dynamic workload. Our neural network model can classify energy efficient data structures based on features such as the number of elements, frequency of operations, interface and set/bag semantics. The 10-fold cross validation result show 95.80% average accuracy of these predictions. Our n-gram model can accurately predict the most energy efficient data structure sequence in 19 simulated and real-world programs—on average, with more than 50% accuracy and up to 98% using a bigram predictor. Our GreenC5 prototype demonstrates how a green data structure can be implemented. With a simple decision making technique, the data structure can efficiently adapt for energy efficiency with low overhead. The median of GreenC5's potential energy savings is more than 60% and ranges from 18% to 95%.

ACKNOWLEDGEMENTS

I would like to thank the government of Thailand and the Royal Thai Army for providing me the full scholarship and the unimaginable opportunity to complete my study here at the University of Denver. Especially, I am very grateful to my advisors, Dr. Matthew J. Rutherford and Dr. Rinku Dewri, whose expertise, understanding, generous guidance and relentless support made it possible for me to archive the final goal. It was a pleasure working with two of them.

I would also like to thank all of my colleagues and friends in Denver and other cities and states that help motivate and provide me any support while I am away from home. I would like to thank my mom, dad, brother, sisters and relatives at home for their love, motivation and moral support. Finally, and most importantly, I owe an enormous debt of gratitude to my wife, Niruemon, and my two children, Lita and Tim, who have been a part of my journey. Through the struggles and trials of this dissertation, they have been a constant source of love, joy and motivation. Thank you.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

In modern computing systems, energy consumption has becoming increasingly important among systems that rely on battery power. Moreover, the proliferation of global computing devices has pushed IT energy consumption higher and higher, raising concerns among many environmentalists. As reported in an article by Digital Power Group [1], information and communication technology ecosystems have approached 10% of the world's electricity generation. The number has been estimated to range from 1,100 to 1,800 TWh annually. As worries about global warming issue are increasing, this trend inspires depression among environmentalists. The need for reducing IT energy consumption and research efforts in every sector is now necessary, not just for energy saving and extending battery life, but also for the environment.

There are many areas in the layers of computer systems that can be optimized for energy efficiency—from hardware [3, 4], operating systems [5] to application layers [12, 13]. At the software application layer, green or "unpolluted" design of software applications can also help minimize the system energy consumption [2]. However, there are also many areas in the software that affect the system performance and can be optimized for energy efficiency—from user behaviors/application workloads [29, 30, 31], coding styles and algorithms [6, 29, 32], to design patterns [33]. Like hardware engineers, software developers too can now join the rest of the communities in the fight against climate change.

One way to improve the energy efficiency of software applications is by optimizing the program code or modifying some of their properties so that the overall energy consumption is reduced—either by making the software run faster, consume less power or both. The main goal is that the overall energy consumption of the optimized program is less than that of the non-optimized one. Energy consumption is a product of execution time and power consumption. Power and performance (execution time) are considered conflicting attributes and are often traded off in order to achieve energy efficiency improvement [41]. One may argue that energy consumption can be reduced by just making the software applications run faster. However, this is not always the case. Abdulsalam et al. conduct a thorough study to show that faster code does not always lead to more energy efficiency code [102]. From their study, the energy efficiency codes can lead to either faster and lower power code, faster and higher power code, slower and higher power code or slower and lower power code. The authors are able to find program examples for all eight of their red (waste energy) and green (save energy) program categories. The authors also suggest that "judging software energy efficiency by time analysis or power usage alone is a deficient vision, which will bring in uncertainties and sometimes cause confusion." However, in our main study, we look at only the energy dimension. Our study aims to improve the energy efficiency of software applications by building a software tool that can help software application to consume less energy.

Today, sustainability has been gaining importance among software engineering communities. There have been many studies that include greenness, carbon footprint and sustainability in the existing green software engineering life cycles, to promote green and sustainable software development. For example, Naumann et al. propose GREENSOFT, a

conceptual reference model for sustainable software [24]; Mahmoud and colleagues propose a new two-level green software model that covers the sustainable life cycle of a software product and the software tools promoting green and environmentally sustainable software [25]; Lago et al. develop a sustainability analysis framework that enables software developers to specifically consider environmental and social dimensions in their green software projects [26].

Most research is concentrated more on the conceptual and higher level but not much is focused at the lower level, in particular the coding and implementation phases. There is still a lack of easy-to-use tools that can help programmers develop green software. These tools could help promote the green software development and are an important factor for software engineers and developers to consider the role of software engineering in the environmental impact of our computing technologies. In this study, we focus on the application layer, primarily object-oriented software with "interface-based" implementation [6], where there are multiple choices of classes that implement the same interface. Our goal is to explore how the choice of these classes impacts the energy consumption of software applications and to find ways to intelligently and dynamically switch between implementations for energy efficiency.

1.1 Green Data Structure Introduction

Our vision is to see a new generation of software applications composed of smart/green objects and components. We envision that these adaptive objects and components have the ability to intelligently adapt themselves to the workloads and environments for energy efficiency, and become the main building blocks in developing green software applications. The green objects are smart because they can learn, classify

and predict their workload, and can decide when and how to dynamically adapt for energy efficiency.

As a case study, we investigate dynamic data structures because they have been shown to have a large impact on performance and are considered key components of many object-oriented software applications [7, 8]; many applications are implemented using interface-based design [6]. By using a "select the right data structure for the right workload" approach, we apply machine learning tools and enhance these classes to be adaptive green data structures that can dynamically adapt for energy efficiency without creating more work for developers.

We propose a working prototype of adaptive green data structures called GreenC5, demonstrating how a green data structure can be implemented and how the models for predicting energy efficient data structures are integrated and used. Together with simple decision making and transformation techniques, we demonstrate that GreenC5 can accurately and efficiently adapt to the workload helping the software applications to perform better and the base system to consume less energy than necessary. The evaluation results also show that our models for predicting energy efficient data structure are accurate and the a priori knowledge is potentially be universal because it can be used in multiple platforms with similar performance results.

1.2 Programming Problems

Dynamic data structures have been around since the beginning of computer programming. They are major components of many programs and widely used in many software algorithms. They are also considered the key performance and energy consumption factors of software applications and computing devices. In programming, the

selection of a data structure implementation is normally done at the development phase. Once a program is completed, the data structure choice is fixed. Dynamic data structure selection and switching are not normally done at runtime. Moreover, programmers tend to choose their favorite data structures [8] for their programs, often without taking performance and/or energy consumption into consideration, or knowing whether there are better choices. As a result, the energy consumption of computer systems when running the application can be higher than necessary. The process of selecting the most energy efficient data structure should be automatic and switching to a different implementation should be dynamic, without creating more work for programmers. Our long-term goal is to develop an architecture for adaptive green data structures that allows programmers to replace their existing data structure with a universal "green" data structure, and expect the programs to function the same, with minimal overhead and configuration. It is the same as a drop-in replacement of existing collections.

Dynamic selection of the right data structure for the right workload, algorithm and program is sometimes not an easy task and many times difficult at the program level done by programmers. It would be much more convenient if the selection is left to the program. As shown in Figure 1(a), traditionally, programmers have to select data structures during the implementation phase. Once a data structure is selected, it is normally not changed at runtime. Our ultimate goal is to have a green data structure to be used as shown in Figure 1(b)—only one green data structure is declared; and the automatic switching of internal data structure implementations is done by the data structure itself. There is no need for programmers to select an energy-efficient data structure for their programs. The GreenC5 changes its internal data structures automatically and dynamically at runtime, for energy

efficiency. The idealized green data structure should also preserve all capabilities of the original data structures and the usage should be similar to the original ones so that the learning curve for programmers is at the minimum level.

```
ArrayList<string> ds = new ArrayList<string>();
ds.Add("Hello");
ds.Update("Hello");
ds.Find("Hello");
ds.Remove("Hello");
```

(a)

```
GreenC5<string> ds = new GreenC5<string>();
ds.Create("Hello");
ds.Update("Hello");
ds.Retrieve("Hello");
ds.Delete("Hello");
```

(b)

Figure 1. (a) Code Example of Traditional Data Structures; and (b) Code Example of Our Green Data Structure

1.3 Research Questions and Key Contributions

Our main research questions are related to interface-based dynamic data structures in object-oriented programming. In particular, the main topic in this dissertation focuses on the development of a smart and adaptive energy-aware dynamic data structure for green software development, called GreenC5. The key challenges and main research questions reside throughout the research and development phases of the GreenC5—from finding solutions for the problems of dynamic data structures, to developing software energy measurement tools and energy profiling process, to creating the predictive models and

training datasets, and to implementing and validating the green data structure. The following are the key data structure research questions to be answered in this dissertation:

1) Does the choice of data structures impact a system's energy consumption?
2) Can we create models for classifying and predicting energy efficient data structures for use in object-oriented programs?
3) Can we intelligently switch between different data structure implementations to improve energy efficiency?
4) Can a green data structure be implemented?

To answer the questions, we implement a working prototype of an adaptive green data structure for green software development, the GreenC5. Along with the implementation process, we also identify key performance/energy features of dynamic data structures and develop a predictive model for dynamic selection and switching of energy-efficient data structures. The study also provides some in-depth concept, interesting research insights and the design, architecture and actual implementation of our adaptive green data structure. Along with the software-layer study, we also include one of our initial explorations in green computing research at the hardware layer, in particular the energy impact from the cache system and the power-performance tradeoffs using live power data. The key contributions of this dissertation are fivefold:

1) Empirical evidence that energy saving opportunities exist in interface-based, object-oriented dynamic data structures.
2) Development of a predictive model based on artificial neural networks and n-gram inference to predict energy efficient data structures for use in object-oriented programs.

3) An architecture for building an adaptive green data structure.
4) A working prototype of GreenC5 that is lightweight, smart, adaptive and easy-to-use.
5) Understanding of power-performance tradeoff of live power data using the Pareto optimality principle.

1.4 Dissertation Outline

This dissertation contains two main studies in green computing research area. In particular, our interest areas are at the two main computer system layers—hardware and software layers. Our first study concentrates on the cache system, a key component of computer systems at hardware layer. The second study aims at the application layer, in particular the data structures. The two components are selected mainly because they both have shown to have large impact on the performance of computer systems. In these studies, we want to further explore their energy impacts and find ways to optimize and minimize the energy consumption. Our first study (Chapter 4) is presented as an initial and additional work to the second study (Chapter 5, 6, 7 and 8). The green data structure in the second study is the main topic and contains answers to our key research questions in this dissertation.

The remainder of this dissertation is organized as follows: Chapter 2 and 3 present a literature review and background materials. They include some introduction and discussions about green computing research, power measuring and profiling, green software development and machine learning for energy efficient computing. Chapter 4 presents our first study about the energy impact from the cache system. The goal is to understand the power-performance tradeoffs on computer systems using Pareto

optimization method. The remaining chapters present our second study of the green data structure. Chapter 5 describes the background and our design of the adaptive green data structure. Chapter 6 describes in-depth details of our methods for predicting data structures for energy efficiency and the evaluation results. Chapter 7 and 8 discuss the implementation, evaluation results and additional analysis of our green data structure prototype, respectively. In this chapter, we also present some examples found in the experiment to demonstrate how the GreenC5 performs dynamic transformations for energy efficiency. The last chapter summarizes our conclusions and describes the future work.

CHAPTER 2: GREEN COMPUTING BACKGROUND

As the obligation to reduce environmental impact is becoming crucial in the fight against climate change and global warming issue, the energy efficiency of computer systems are increasingly becoming one of the most important research topics for researchers, system designers, architects and software developers. This chapter explains the background and related work in green computing research and our green data structure for green software development.

2.1 Green Computing Research

The research in this dissertation is in the area of Sustainable/Green Computing or Green IT. As stated in a paper by Murugesan [86], Green Computing is defined as:

> "the study and practice of designing, manufacturing, using, and disposing of computers, servers, and associated subsystems such as monitors, printers, storage devices, and networking and communications systems efficiently and effectively with minimal or no impact on the environment."

The paper also states that "Green IT benefits the environment by improving energy efficiency, lowering greenhouse gas emissions, using less harmful material, and encouraging reuse and recycling". The "Green Computing" or "Green IT" concept was introduced in 1992 when Energy Star was launched by US Environmental Protection Agency (EPA). The main purpose of the Energy Star labeling is to give a recognition to electronic equipment that meets the EPA's energy-efficiency standard [53]. Green

computing normally takes all product life cycles that can directly or indirectly impact the environment into consideration, from manufacturing to usage to disposing and recycling [54]. Nonetheless, the main focus of the Energy Star program and Green Computing initiatives are primarily on energy efficiency of computer hardware devices, servers and data centers, not on software applications which run and operate the hardware components. However, recently the green initiatives start to also include developing green software applications as part of the main focus to minimize carbon footprint of computing technology.

Based on Zomaya and Lee [87], Green IT can be made up of three parts: (1) "designing product that are less polluting, less energy consuming and easier to recycle"; (2) "building more efficient data centers"; and (3) "working on innovative projects that will enable, via IT contributions, in building of a more sustainable world". In more detail, Murugesan [86] also explains that the main research topics in Green IT include:

1) Design for environmental sustainability—making business operations, buildings and other systems energy efficient by balancing energy and resource savings by Information and Communication Technology (ICT) infrastructures.
2) Energy-efficient computing—the efficient use of computing resources such as energy-aware algorithms, green compliers, parallel programming and energy efficient software development.
3) Power management—the use of hardware/software power management systems that help optimize the performance, energy consumption and manage the power resources in computer systems.

4) Data center design—environmental-friendly designs that improve energy efficiency and energy conservation of data centers.
5) Virtualization—the recreation of an entire system in software, which provides a virtual version of a machine to all software application to run on.
6) Disposal and recycling management—managing e-waste and developing plans for disposing, upgrading and replacing devices in a more sustainable and environmental friendly manner.
7) Regulatory compliance— legislative actions and regulatory requirements that can force acceptance of a technology or practice.
8) Green metrics, tools and methodology assessments—software tools for collecting, reporting and analyzing energy consumptions of computer systems and platforms for managing the environmental risks, environmental impact and greenhouse gas emissions, emission trading and ethical investing, etc.

Many organizations, such as the UN, WEF, GATT, G8 and G20 and governments of many countries, are now realizing the importance of environmental impact and climate protection and support the idea of three Rs: Rethink, Redesign and Rebuild [87] for mitigating the environmental impact. It has been estimated that by the year 2030, if the current trends continue, worldwide electricity consumption by ICT infrastructures will grow by a factor of 30 [88]. The trends are demonstrated by a fast growing number of worldwide Internet users and connected devices, the advancement of broadband Internet, as well as the usage and exchange of online information and rich media such as high definition video streaming among the users and devices. According to a recent report on

data center energy efficiency from the Natural Resources Defense Council (NRDC) [89], an environmental action organization, nationwide data centers in total used 91 billion kilowatt-hours of electrical energy in 2013. They estimate that the number will reach 139 billion kilowatt-hours by 2020, a 53% increase.

2.1.1 Improvement Opportunities

There are many energy efficiency improvement opportunities in all layers of computer products—from hardware, middleware to software application layers. At hardware layer, the semiconductor industry is well aware of energy issues and the environmental impacts. Several energy saving solutions for the hardware or chip level have already been developed such as voltage and frequency scaling, hardware accelerators, on-chip power domains, biometric components, efficient standby modes, etc. [90]. At the software level, however, there is much more room for improvement. One main problem is that software and middleware layers are normally not energy-aware. Software is not originally designed to support and not yet ready for the new energy-efficient hardware features. There is a need for standard and well-defined interfaces between hardware and software that allow the systems to better manage power domains in hardware that could provide significant improvements in energy efficiency [90].

One possible solution at the functional level that has potential to improve energy efficiency of computing devices is by using "good-enough" computing or by relaxing the strict accurate requirements. Some classes of applications, such as user interfaces, analytics and media processing do not require high precision results. These type of applications are well suited for the "good-enough" or adequate precision computing, where inaccurate

result or errors could be accepted and managed and approximate results are sufficient for performing the tasks. For example, to improve performance and energy efficiency, some frames in a video application can be dropped just enough that the quality degradation cannot be detected by human eyes. This is known as Approximate Computing [32]. For traditional energy-efficient computing, it is about the tradeoff between performance and energy consumption. However, for general-purpose approximate computing, it explores a third optimization objective—accuracy or error. The tri-objective optimization process is the tradeoffs of the accuracy of computation for gains in both energy and performance. In addition, there are several promising approaches to use adequate precision computing, such as EnerJ, the language of good-enough [32] and Quora, an energy efficient, quality programmable vector processor for approximate computing [92]. However, at the time of writing this dissertation, none of them is mature enough for use in actual software implementations. This is just one of the improvement opportunities. There are also many other aspects of software applications that can be improved for energy efficiency and are also open for future research.

2.1.2 Green Computing Benefits

We categorize three major areas and groups of people who might obtain benefits from energy-aware computing practices—the environment, the organizations and businesses, and the individuals. The details are discussed in this section.

2.1.2.1 Environmental Benefits

Information and Communication Technology (ICT) sector contributes about 3% of worldwide electricity usage and the same percentage of greenhouse gases [53]. In a recent

study in 2014, the study estimates that 2% of carbon emissions is from the ICT equipment and services and household electronic sector. The ICT total electricity consumption is forecast to reach 1,100 million tons by 2020. Its share of the global carbon footprint is estimated to increase from 1.3% in 2007 to about 2% in 2020. For fixed ICT networks, it is estimated that the share of greenhouse gas (GHG) emissions will be 1.4% in 2020, as shown in Figure 2, due to the increasing number of devices as well as due to network expansion [92]. Even though, these percentage numbers seem to be small, it is expected to get higher since the sector is getting larger and larger as technology is doubling every couple of years according to Moore's laws. A Smart 2020 report [53] states that the number of PCs (desktops and laptops) globally is expected to increase from 592 million in 2002 to more than 4 billion in 2020. Also in a recent study by Juniper Research, it has revealed that the number of IoT (Internet of Things) connected devices will reach 38.5 billion in 2020, up from 13.4 billion in 2015: a 285% jump [128]. When not include other computing devices in the parameters, if each PC is running 10 software applications, there will be 40 billion software applications running on the planet; imagine how much energy can be saved if all the software is green and energy efficient. According to a Climate Group report, despite of making the ICT more energy efficient, applications of those systems to electricity grids, logistic chains, intelligent transportation and building infrastructure could reduce greenhouse gases (GHG) emissions by as much as 15% by year 2020.

According to the NRDC report [89], if companies adopted data center best practices, the reduction in energy use could reach 40% and that the economic benefits would also be substantial. The 40% reduction in energy use would equal to $3.8 billion in

savings for businesses. In 2014, this number represents a savings of 39 billion kilowatt-hours annually—equivalent to the annual electricity consumption of nearly all the households in the state of Michigan. The report also states that the number is only half of the technically possible reduction. The need for solving climate change and global warming problems has led to a growing realization that climate impact must inform everyone including software engineers and programmers to be aware of the threats and try to use all resources to help solve the problems.

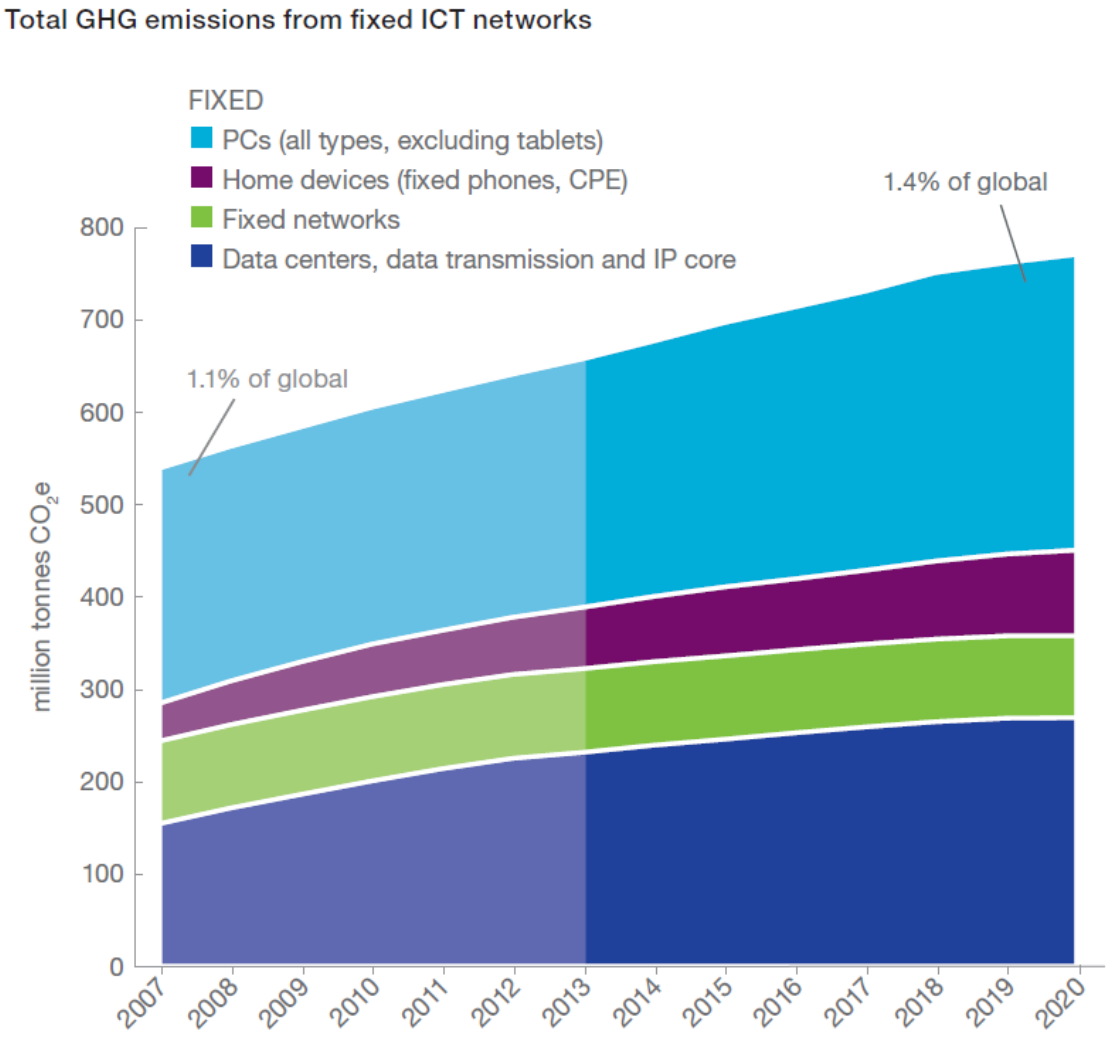


Figure 2. ICT Carbon Footprint Outlook (million tones CO_2e) (from [92])

According to a paper by Smarr [55], the emissions are divided into three component parts—emissions resulting from fixed and mobile telecommunications/internet infrastructures, data centers and the edge of the network. Even though, much attention in Green IT focuses on data centers, but in 2020, the amount of carbon emissions for data centers will account less than 20% of the total ICT emissions. The majority (57%) will come from the edge of the network—such as, phones, tablets, notebooks, PCs, peripherals, printers and IoTs. Even though, the edge devices refer to both hardware and software components. Our main study, however, focuses more on the applications that execute on those devices. The focus on green software can also contribute much impact on reducing the greenhouse emissions since software applications run on and operate all the network edges and the computing devices, which will be on every part of every person's life. Green Computing, Green IT and Green Software Development research must be driven at a faster pace to help the rest of the sectors solve the environment threats before it is too late.

2.1.2.2 Business and Organization Benefits

Going green can directly benefit companies and organizations in many ways. First, it can influence the consumers, the shareholders, and the company perception in the market. Based on an actual study performed jointly by University of San Diego and CB Richard Ellis Group and printed in a Business Week magazine on December 2009, the result from 2,000 business participants shows that businesses with green initiatives had 5% average increase in business net worth and 74% increase in business image and name branding [56]. The "greenness" of computing and software products can also be used as a part of

marketing campaigns and can help build better image and name of their brands and drive profits.

As part of the technological advancement, businesses and organizations like governments play major roles in driving the future technology. The green computing and energy-aware software development initiatives cannot be successful without the participation of businesses and organizations since the majority of software applications are developed by them. In addition to the better business perception and brand image, companies and organizations going green might also gain benefits from the IRS tax breaks [58]. For example, a tax credit for 30 percent of the cost of a residential fuel cell and microturbine system, up to $500 per 0.5 kilowatt of power capacity, is available through December 31, 2016. And, tax credits for all-electric cars range from $2,500 to $7,500, based on the vehicle's battery capacity, etc.

Currently, there is no tax credits available for building green software. However, in the future it is possible that similar tax breaks could be issued to promote green software development if governments realize the importance of the green practices in software application development. Not only just commercial benefits, businesses and organizations, in particular their IT departments, might have to be forced to follow and be compliance with International Standardization Organization (ISO) because green quality is being considered as a standard quality metric for software applications. One evidence can be found in a paper by Lago et al. [26]. The paper mentions that a working group on software architecture (WG42, working on ISO/IEC 42030) is considering to include Kern et al. [24] who developed a quality model for green software that refers to quality factors from

ISO/IEC 25000 based on direct and indirect software-related criteria. Also, Calero et al. [94], who considered sustainability in 2013 as a new factor affecting software quality, presented a quality model based on ISO/25010.

Today, consumers are becoming more educated and environmentally conscious. According to a Nielson's marketing research [58], "66% of global consumers say they're willing to pay more for sustainable brands—up 55% from 2014. 73% of global millennials are willing to pay extra for sustainable offerings—up from 50% in 2014". A new emerging trend of consumers is increasingly asking for products and services from businesses that are green and care about the environment. As a result, software companies with green software in their product lines could drive more green oriented customers leading to more sales, increased revenue, higher gross margin and higher profit.

2.1.2.3 Our Benefits

For individuals installing green software applications or having computing devices with an energy management component, some direct benefits are the energy savings and longer battery life. Many studies show that power-aware or green software applications and a good energy management component can help reduce the energy consumption of computer systems and lengthen battery life of mobile devices. This is the main reason why big companies such as Apple, Google and Windows have battery management as one of the key functionality in their operating systems and hardware platforms. As the result, there are also many research studies focusing on this area. For example, applications built using energy-efficient GUI (E^2GUI) technique for its graphical interface can significantly lengthen battery life [59]. The design technique involves reducing system energy through

optimization of human-computer interaction. The researchers demonstrate that the techniques can improve the average system energy of three benchmarks (text-viewer, personal viewer, and calculator applications) by 26.9%, 45.2% and 16.4% respectively. Moreover, the use of parallel programming such as MapReduce [60] and the code transformation techniques in [61] can be applied and used in green software construction phase when a code transformation tool can help reduce the system's energy consumption. The authors of the research paper claim that the merging process of code transformations in parallel using MapReduce can contribute nearly 29.6% in energy saving. The energy reduction corresponding to the final optimized source code after transformation is 37.9% when compared to the un-optimized ones.

Even though, the amount of energy saving from running green software applications in a device seems to be small for individual, many people seem to be fine buying green products for a higher price. According to a Green Buying research survey [62], "82 percent of consumers buy green products." The main reason from buying green products is that it is good for the environment. This clearly indicates that people now are more and more environmental conscious. In the future, there will be billions of PCs, laptops, tablet PCs, mobile phones, digital photo frames, printers, gamming consoles, TVs, set top boxes, refrigerators, and other future computing devices driven and controlled by software applications and interconnected to the world network exchanging richer and more interactive data like 3D movies, high definition images and voices, and large data files. If each individual installs green software applications in their devices, there will be an

immense amount of energy saving combined. This small energy saving for each individual can have a tremendous contribution in the fight against climate change at a global scale.

2.2 Energy-Efficient Software Research

Software applications can have a large impact on the system energy consumption. Accordingly, many research studies focus on different areas in the software application layer. Software applications have different architectural designs depending on application type and hardware platform. For one example, many modern software applications can have multiple layers such as a 3-teir architecture design [63]. The 3-tier software application architecture consists of Tier 1—Presentation layer, Tier 2—Business Logic layer, and Tier 3—Data layer. For simplicity, this section presents energy-efficient software research in the 3-tier architecture.

2.2.1 Presentation Layer

The presentation layer of software applications normally refers to the graphical user interface (GUI), located at the top part of the multi-tier layers. This layer interacts directly with the users. If software components at presentation layer are designed poorly, they give the users poor view of the system and sometimes can cause the computer systems to consume energy higher than necessary. In addition, by designing the user interface properly, we can reduce power consumption of the whole system at a great level. For example, in an early study by Qu and Potkonjak [64], the study presents techniques for minimizing energy of a software application with guaranteed quality of service (QoS), one of the key features for new Internet-based multimedia and other applications. The paper presents how to satisfy QoS requirements and minimize the system's energy consumption.

The key contributions in their paper include formulation of the energy minimization with QoS guarantee and development of the dynamic programming (DP) procedure for solving the general energy minimization (EM) problem. Their simulations show an average of 38.7% energy saving over the system shut-down technique. At the presentation layer, image, voice and movie rendering can be optimized using their techniques to minimize the power/energy consumption of a software application.

Moreover, energy-efficient GUI (E^2GUI) design techniques [59] are proposed to improve system energy efficiency without sacrificing application performance, ease of use of aesthetics. The optimization techniques of E^2GUI include power reduction techniques (low-energy-color scheme and reduced screen changes), performance enhancement techniques (hot keys, user input caches and content placement) and facilitators (paged display and quick buttons). For the first technique, the researchers show that low-energy color schemes of GUI can help reduce display energy. The second power reduction technique, reducing screen changes, can also minimize the energy by reducing the switching activity and computation required for screen generation. The authors also give some examples; user-perceived responsiveness is different from real responsiveness. By using a progress bar, it makes a user to feel that the computer is more responsive. But, it actually slows down the system and increases energy consumption. Animations give users a natural feeling for screen changes. However, these animations waste energy and add little or no functionality, but increase the power consumption. To enhance performance, the focus is on improving user productivity by eliminating the time and energy waiting for user responses. For example, hot keys instead of menus styles can reduce the amount of energy

consumption. Also, user input caches which store the most recent use, most frequent use or most common use inputs by users can also reduce amount of computation and screen generation and help save system energy. Content placement techniques focus on reducing the user interaction time for frequent inputs by strategically laying out the GUI content by using perception capacity, motor speed and cognitive speed as the main consideration for the layouts. Lastly, the facilitators are a pair of techniques that enable or enhance the effects of the other techniques. The paper demonstrates that paged display and quick buttons can reduce energy consumption. Paged display enables increased user interface functionality by increasing the effective display size. Page navigation buttons are designed to enhance the user interaction speed. Quick buttons use the available hardware buttons to increase user interaction capabilities. Holding down a hardware button can act as a <SHIFT> or <Control> key, which can facilitate the use of key combinations that are traditional impractical for GUI design. Similarly, the same authors in [65] also conduct energy efficiency improvements of handheld computer interfaces and study the human sensory/speed limits and characterization and proposed some practices to low energy consumption at the GUI level.

There are also many other studies about color scheme impacting the display power consumption. For example, in an article by Whitman [66], use of different colors, color patterns and color sequences in LCDs, OLED-based displays and AMOLED displays of flagship phones, consumes different amount of power and can be exploited for energy efficiency. Thin film transistor (TFT) LCDs consume more power when white than when black. OLED-based displays consume the power proportional to the number of on pixels

and their luminance. The article also states that a black screen does indeed consume 41% lower overall power when using the predominantly black interface in Reddit Sync. By using AMOLED-friendly apps most of the time, users can actual gain an extra of 15 or 20% screen time. Also, according to Williams and Curtis [67], displays and graphic cards consume 42% of all power consumption in an average laptop (33% for display and 8% for graphic cards). Therefore, by redesigning GUIs to be displayed in the different types of displays, it can have a great impact on the amount energy consumption of an overall software application.

2.2.2 Business Logic Layer

A business logic layer contains business functionalities that control the applications by performing detail processing. This layer can be viewed as the main computation and processing layer to manage and control above presentation components and the data layer components. It mainly acts as the middle layer between those two layers. Therefore, most of the coding and complex computation processes are normally laid in this layer. These normally include data access layer because this tier acts as an interface to the data tier, retrieving and storing information both remotely and locally (from databases, file systems or storage systems such as CD, DVD or memory sticks). For remote communication, data transmission can greatly impact the performance and power consumption of the software application. Therefore, the efficiency of all computational components can also be optimized here. This includes choosing energy-efficient algorithms and libraries, compression techniques, decoding-encoding schemes, data structures and applying multi-threading techniques to some computations to minimize power consumption, for example.

The following are some of the examples that can be used and applied in the business layer of this software architecture. Some of the examples are also discussed in a greater detail in the later sections.

A study by Benini and Micheli [68] states that algorithms can have approximate implementations. For example, certain operations may be implemented with limited accuracy to reduce energy cost. This is the same as Quality of Service (QoS) and energy consumption tradeoff analysis. For example, given by the authors, a *cos(x)* function can be approximated as a Taylor expansion $1-x^2/2 + x^4/24$. Furthermore, it can also be approximated as $1-x^2/2 + x^4/32$, which is simpler because the division of the last term by 32 can be done by a *Shift* operation. In addition, Eckerson [69] also mentions that the choice of algorithms and data structures can make a large difference in the performance of an application. Using an algorithm that compute a solution in $O(n \log n)$ time is going to perform better than one that does the job in $O(n^2)$ time. For some applications, a stack may be better than a queue and a B-tree may be better than a binary tree or a hash function. A study of the problem and a careful consideration of the architecture, design, algorithms, and data structures can lead to an application that performs better and consumes less energy. These will be more useful in the future applications if the consideration and selection process also include energy consumption into the performance and quality metrics. Or even better, the selection of the choices can be done automatically and dynamically by the system itself.

In addition, Barr and Asanović [70] had studied the wireless data transmission and concluded that wireless transmission of a bit can require over 1,000 times more energy than

a single 32-bit computation. They suggest that number of bits of data should be reduced by some computations before transmitting. The key is the energy required to compress/compute the data. If it is less than the energy required to transmit it, there is a net energy saving that can contribute to a longer battery life for mobile devices. The study shows that energy use from data compression and decompression can be minimized through smart use of memory—including efficient data structures and/or sacrificing compression ratio for "cacheability". The result from the study shows that by choosing the lowest-energy compressor and decompressor on the test platform, rather than using default levels of compression, overall energy to send compressible web data can be reduced 31%. Energy to send harder-to-compress English text can be reduced 57%.

2.2.3 Data Layer

A data layer is responsible for retrieving, storing and updating information; therefore, this tier can be ideally represented through a commercial database, local or remote file systems, disk drives or other storage systems. For a database system, stored procedures are considered as a part of the data tier. Currently, we can optimize energy efficiency at the data layer to some degree based on types of storage systems of the applications. For database system, we can utilize the tools provided by the database. For example, the usage of stored procedures can increase the performance and code transparency of an application since the processes are done at the database process and can minimize the computation at the business layers. A survey paper by Jones et al. [71] has mentioned about minimizing power consumed per transaction through embedded indexing for mobile applications. The idea is to combine the index information together with data

on the single broadcast channel in order to minimize access time. The author also mentions about the energy efficient query optimization for database systems. They show that query statements too can be optimized similar to the code optimizations.

Moreover, Mathur et al. introduce Capsule, an energy-optimized log-structured object storage system for memory-constrained sensor devices that enables sensor applications to exploit storage resources in many ways [72]. The idea of the storage system is to employ a hardware abstraction layer that hides the NAND flash memories for the application and support energy-optimized implementations of commonly used storage objects such as stream, files, arrays, queues and lists etc. The authors claim that Capsule can provide platform-independence, greater functionality, more tunability, and greater energy efficiency than existing sensor storage solutions. Also, the experiments not only demonstrate the energy and memory efficiency of I/O operations in Capsule but also show that Capsule consumes less than 15% of the total energy cost in a typical sensor application. This is an example of a storage system that can be used for development of green sensor software applications.

On the other hand, Ousterhout et al. see that disk-oriented approaches are becoming increasingly problematic in term of its scalability and access latency and bandwidth constraints. They introduce a new approach to datacenter storage called RAMCloud, where information is kept entirely in DRAM and large-scale systems are created by aggregating the main memories of thousands of commodity servers [73]. They claim that RAMClouds can provide durable and available storage with 100 - 1000x the throughput of disk-based systems and 100 - 1000x lower access latency. The new storage system is specially

designed for data intensive applications and has the following properties—low latency, large scale, durability, powerful data model and easy deployment. From their energy efficiency evaluation results, when measured in terms of cost per operation or energy per operation, RAMClouds are 100 - 1000x more efficient than disk-based systems and 5 - 10x more efficient than systems based on flash memory. Thus for systems with high throughput requirements, a RAMCloud can provide not just high performance but also energy efficiency.

2.2.4 Energy-Efficient Programming Practices

Steigerwald et al., a team of researchers at Intel, have conducted a thorough study on data efficiency and its energy saving and developed a white paper guiding programmers on how to create energy-efficient software [130]. In this section, example guidelines are mainly based on recommendations made by the research team. The team proposes that data efficiency can be achieved by designing software algorithms that minimize data movement, memory hierarchies that keep data close to processing elements, and application software that efficiently uses cache memories. They have conducted experiments on DVD playback, disk I/O and file transfer over wireless applications. Their test results and recommendations are in great details and practical for programmers and developers to use. For example, the following proposed three guidelines can help minimize the system energy consumption during a DVD playback:

1) Buffering—the study shows that DVD playback implemented with buffering techniques can help reduce DVD power consumption by 70% and overall platform consumption by about 10%.

2) Minimize DVD drive use—they recommend that reducing DVD spin-up, spin-downs, and read access can also help reduce energy consumption.
3) Let the OS manage the CPU frequency—they do not recommend changing the CPU scheme to run the processor at the highest available frequency and let the OS manage the frequency by itself.

For disk I/O, the test result shows that disk spin-up takes the most time and consume the most power. Some of their recommendations include:

1) For reading a large file, use block sizes of 8KB or greater for improved performance.
2) Utilize a buffering strategy in multimedia playback to minimize disk reads and save energy.
3) Avoid by pre-allocating large sequential files when they are created.
4) Use *NtfsControlFile()* function to help in defragmenting files.
5) Applications that deal with random I/O or I/O operations with multiple files should use asynchronous I/O to take advantage of Native Command Queuing (NCQ, an extension of the Serial ATA protocol allowing hard disk drives to internally optimize the order in which received read and write commands are executed).
6) Queue up all the read requests and use events or callbacks to determine if the read requests are complete.

7) For multiple threads competing simultaneously for disk I/O, queue the I/O calls and utilize NCQ. Reordering may help optimize the requests, improve performance, and save energy.
8) When multiple threads competing for the disk causes significant disk thrashing, consolidate all the read/write operations in a single thread to reduce read/write head thrashing and reduce frequent disk spin-ups as well.

For file transfer over wireless application, the study focuses on how the compression ratio or size of the file affects power consumption. The researchers recommend the following practices to improve data efficiency and energy efficiency:

1) Data sets with higher compression ratios (more than 3.0x), are recommended to be compressed before uploading/downloading data. This practice provides better power savings as compared to transmitting uncompressed data. For data sets having higher compression ratios, it is recommended that applications transmit compressed data.
2) Data sets with lower compression ratios (~1.2x in this case which is hardly to compresses) is not recommended for compressing the data before uploading uncompressed. This practice adds extra overhead. They instead recommended uploading/downloading uncompressed data.
3) Data sets with compression ratio around 2.5-3.0x provide minimal difference in the power saving between uploading/downloading compressed data and uncompressed data.

There are also many other techniques that can be used and applied at the design level to optimize the application performance and energy efficiency. For example, the utilization of parallelism (parallel programming and multi-threading) can help reduce energy consumption of computing devices [60]. There are also some energy frameworks that can be used to develop green software applications. For example, green compiler that can be considered as the main compiler of green software projects [95]. Today, there are so many choices of algorithms, components and libraries available for selection. One challenge is how the selection of a proper choice can be made in solving a programming problem, while the performance and energy efficiency are also improved. It will be helpful if there is a tool that can aid programmers and designers in the decision making.

2.3 Power Measuring and Profiling

Among all the areas of green computing research, power measurement and energy profiling are considered the key research areas. From a famous quote by Deming, "if you can't measure it, you can't manage [and improve] it" [131]. For green software development and research, power measurement and profiling tools are required in order to conduct the power-performance evaluation and energy consumption analysis of software applications and are used to trigger energy-aware mechanisms and evaluating the effectiveness of these mechanisms. According to Calandrini et al., power profiling can be categorized into hardware-based and software-based method [96]. Hardware-based power profiling mainly uses different types of instruments to directly measure the power of a device. The accuracy is normally higher than that of software-based profilers. However, they are not suitable for software applications since a power meter is required. It is also

expensive to integrate with software and can provide data with low granularity at low latency. Hardware-based methods are usually used to evaluate the effectiveness of power saving techniques at the system level. However, some modern computer boards start to include power metering tools that can measure power consumption of different hardware components. For software-based power profiling, the methods try to estimate the power of different levels by designing a group of power models. Software-based tool is normally more user-friendly and more suitable for software applications; can measure power consumption at the component and software application level; and can provide real-time power data at finer granularity. However, they are normally not as accurate as the hardware-based tools. In this section, we first describe the power metrics and benchmarks before going back into details of the power measuring and profiling with some examples of the existing tools.

2.3.1 Energy Metrics and Benchmarks

2.3.1.1 Energy Metrics

Energy is measured in Joule (J) or Watt-Hour (Wh). Power is measured in Joule/second (J/s) or Watt (W). Energy can be calculated by using the formula: $E = P \times t$; where E is Energy unit in joules, P is Power unit in watts, and t is a time unit in seconds. Both metrics are widely used to characterize power consumption of IT and ICT systems. In computer systems, energy is used more in research related to mobile platforms and data centers. For mobile devices, energy is strongly related to the battery lifetime. For data centers which consumes a large amount of energy, energy is used as the concern of electricity costs. Usually, research in these areas uses energy efficiency, such as PUE

(power usage effectiveness) [97], as a metric to evaluate their work or productivity. Power metric is used to reflect the current delivery and voltage regulator of the circuits. In green computing research, power may also be used for the abstract concepts of power consumption as effectuated by system process, operating system and software applications. Based on papers by Chen and Shi [98] and Ardito [99], the authors provide some of following useful terminologies:

1) "Energy is the electricity resource that can power the hardware devices to do computation" [98].
2) "Power is the dissipate rate of energy" [98].
3) "Efficiency is the ratio of useful energy and total energy used" [99].
4) "Productivity is output/resource on a time interval of a production process," where the output is computational work and the resource is energy. Examples of computational work include operations performed, network bits transmitted and a web application hits and more [99].

There are several metrics being used to measure the energy cost and productivity of computer systems in green computing research. For example, gathered by Ardito's dissertation [99], a million instructions per second per watt (MIPS/Watt), number of floating point operation computed per watt (MFLOP/Watt, FLOP/Watt) and Useful Work/Watt are used for measuring a computation capability and productivity of high performance computers and data centers. For networking, Environmental Performance Index (EPI) [100] uses 100(MI)/M, where I is energy consumption at idle state and M is maximum energy at active state, as a metric to measure the network efficiency. There are

also studies that use KB/Joule and Gbps/Watt to measure the rate of data transferred per joule or watt over a network channel [101].

In a recent paper, Abdulsalam et al. propose the Greenup, Powerup, and Speedup metrics (GPS-UP) to categorize software implementation and optimization efficiency [102]. Using the three metrics, the researchers can categorize software optimizations into one of eight categories. Speedup is defined as the ratio of non-optimized code runtime over optimized code runtime. Similarly, Greenup is defined as the ratio of the total energy consumption of the non-optimized code over the total energy consumption of the optimized code. Greenup and Powerup are analogous to Speedup as it reflects how green the optimized code is in term of energy and power consumption, respectively. Powerup implies the power effects of an optimization. In our green data structure study, the energy efficiency metric is the same as the Greenup metric. Our study mainly looks at how much energy consumption is reduced as a whole when using our methods and tools, by using the product of both the power and runtime data into consideration.

2.3.1.2 Energy Benchmarks

At the time of writing this dissertation, there are few benchmarks for energy efficiency in the green computing research area. However, with the growing trends in this area, we expect to see more benchmarks to be developed in the near future. For the existing benchmarks, JouleSort [75] is an external, system-level, I/O centric benchmark for evaluating energy efficiency across many types of computer systems. It is an extension of the sort benchmarks, which are used to measure the performance and cost-performance of computer systems [132]. The idea behind the Joulesort benchmark is to sort a predefined

number of randomly permuted 100-byte records with 10-byte keys under a controlled condition. The goal is to sort the records with minimum energy use.

Another example for commercial use, SPECpower_ssj 2008 [76] is a benchmark suite to evaluate energy efficiency of server-class computer equipment. It is used to compare power and performance among different servers and serves as a toolset for use in improving server efficiency. The suite is implemented in Java. Therefore, it can be executed on almost all operating systems and platforms. Advantages of SPECpower_ssj 2008 compared to JouleSort are the more advanced features such as the automated power measurements, the power measurement for different load levels and the possibility of considering temperature as an important environmental factor.

Moreover, the EnergyBench [76] is another industry-standard energy benchmark that provides data on the amount of energy consumption of a processor when running EEMBC's performance benchmarks. EnergyBench provides design engineers with comparable information regarding energy consumption and insights into the power budget cost of a device's performance by allowing a performance/energy number to be derived using the consolidated performance score in each benchmark suite. The existing benchmarks are designed specifically for evaluating hardware computing devices and data centers. However, at the time of writing this document, there is no benchmark designed specifically for energy efficiency of software applications. Also, we do not use any of the mentioned benchmarks in our study because they are incompatible with our study. Our study focuses more on the dynamic data structures specially designed for C# programming language and .NET technology.

2.3.2 Software Power Measurement

In any computing device, although the main drivers of energy consumption is always at the hardware, the way the energy is consumed is also influenced by the software. In a paper by Chen and Shi [98], the authors state that software power measurement is "state of the art" because we cannot directly measure the energy consumption of software applications just like directly measure the energy of computer systems using a plug load meter. In fact, the software power measurement can be explained and drawn by a theoretical model of the energy consumption, which depends both on hardware specifications and the way in which they are used by software artifacts. In the paper, an abstract model underlying the power consumption can be summarized as:

$$Power = Idle + \sum_{c \in Components} Hw_c \times Sw_c \quad (1)$$

The total power consumption of a device, when turned on, is composed of an *Idle* part that is present even when the device is sitting idle. The additional consumption depends on the individual hardware components maximum consumption, which is driven by what the software forces it to do. Depending on the software requests, the hardware component may run at full throttle or remain idle. Accordingly, to measure energy consumption of a particular software application, we just control other variables in the above model and vary the software parameters. The energy consumption data can give developers the energy visibility for energy efficiency improvements of software applications. For example, from one of our research studies, Figure 3 presents power/execution time visibility when executing an ADPCM program, from the start to the end, on an FPGA Atlys development board. Each line/legend of different voltages in the

graph represents power consumption and execution time of different parts of the Atlys board—from the top legend down, IO/video/USB port, memory, CPU and Ethernet port, respectively. With this visibility, developers can see the interactions between power and performance of software applications and use the information to improve the performance and energy efficiency of their programs.

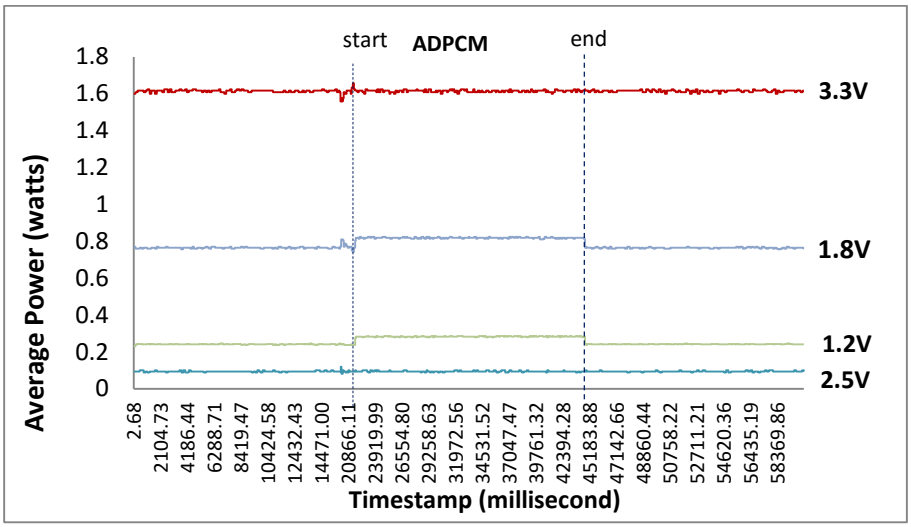


Figure 3. An Example of Power/Runtime Profile Collected by an FPGA Atlys Board Executing an ADPCM Benchmark Program

At the time of writing this dissertation, modern operating systems of mobile devices such as Apple iOS, Google Android and Microsoft Windows, have already integrated the energy measuring tools into their mobile phone operating systems. The tools provide device and application battery usage information to users and can be used to aid developers in developing battery-aware applications for mobile phones. However, these tools are mainly based on software-based power monitoring since the power/energy data are not derived from built-in current sensors, battery monitor unit (BMU) or power monitors attached to the devices. Instead, they are mainly estimated from mathematical formula of application performance counters provided by the OS and/or energy counters provided by

the hardware components such as CPU and power models of the display, Cellular modem, Wi-Fi, GPS, Bluetooth, battery and more [50, 136]. For example, Apple iOS provides an energy diagnostic tool to monitor application and mobile device energy usage. An Energy Impact instrument is integrated in XCode development environment (IDE) for iOS applications to help identify and address energy problems for application development [137]. The Energy Impact gauge in XCode displays a report of application's energy impact from user interactions with the application. Similarly, the Android OS also provides the similar capabilities along with many other 3rd party applications such as Trephen profiler [134] and PowerTutor [135] that can provide the energy information of individual application and total power usage of the mobile device and be used in Eclipse IDE for power aware software development. For Windows OS, the same capability is also available in Windows 8 and 10 operating systems and later versions of their Visual Studio IDE.

In the research community, many researchers have also been trying to improve computer power consumption affected by software system, many monitoring and measurement methods, tools and power benchmarks have also been developed either for specific or general-purpose use. These growing efforts indicate high possibility of developing standard software power measuring methods in the near future. Many companies such as, Apple, Google, Microsoft, Qualcomm, Intel, and others also join the research and develop their own metering tools, which we can use for a green software project and software testers can also use for their software quality measurement process. However, it is also very important that the programmers and testers have some background in green software development and adequate knowledge in software energy consumption.

The following sections present some additional examples of hardware and software-based power measurement and profiling tools that can be used for research purpose.

2.3.2.1 Hardware-Based Power Measurement

For hardware-based power measurement, there are many power meters in the market that are available for use and integrating with special-purpose devices. The meters are used to measure power consumption and understand the power dissipation of devices or different parts of devices. Many studies [3, 27, 79] rely on power meters to measure the real power and use it to validate their research work and their power models. Moreover, some research studies [3, 37] measure the power of hardware components and break it down into sub components based on some indicators that could reflect the activity of these lower level units. The differences in these methods are in the type of meters used to do the measurement and in the place the measurement is done. Many meters are not suitable for computer research since they do not provide public interface or APIs that can be used for computer access and developing power profiling tools. However, there are a few tools that have this capability. The following examples are those that we think are practical and suitable for green software research and development.

First, Watts Up? plug-load meters [16] are among the popular ones because they are cost effective, simple-to-use and accurate. The manufacturer claims that the wattage accuracy of their meters are within 1.5%. The meters are used by many researchers for measuring power consumption of computers, computing devices and servers [27, 79, 103]. Watts Up? meters come with public APIs that allow users to develop power profiling tools

to be used in green software research and development. We also use the meter to create training sets, validate and evaluate our adaptive green data structure in one of our studies.

With the growing popularity of energy efficiency research for mobile devices and the increasing need for low power/energy computing systems, there are several companies starting to integrate power meters, current sensors and energy profiling capabilities into their systems and embedded boards to make energy aware application development easier. For example, Xilinx Atlys FPGA board [37] integrates the power meters inside the boards that can provide power consumption data of different parts/components of the development board, such as Video/USB port, Ethernet port, CPU and memory components. We also use the tool in one of our studies and it will be discussed in more detail in Chapter 4.

Similarly, an affordable and energy-friendly Wonder Gecko development board by Silicon Labs [77] provides a quick and easy way for engineers to evaluate their microcontroller chips. Each board includes an on-board SEGGER J-link SWD debugger programming and debugging via a MiniUSB connector. The board is equipped with on-board energy sensors that can be accessed via its low energy sensor interface (LESENSE). The Wonder Gecko features an advanced energy monitoring system, allowing programmers to program, debug and perform real-time current profiling of their application without using external tools. The company also provides SDKs and tools for programmers to develop software applications on the board and evaluate the energy consumption of the programs.

For large chip companies such as Intel, AMD and Qualcomm, they are also realized the importance of the computer energy efficiency and the need for power measurement tool

for software applications. These can be seen from many of their energy efficiency projects from their research websites. Some companies also start to integrate the power/energy measurement capability to some of their chips and boards. For example, Intel has groups of researchers and labs for energy efficiency research that focus on technologies for the efficient future—low power circuit innovations, platform power management and efficient I/O and memory, for example. One research direction of their platform power management research is to shift the focus from the OS to hardware management platform [138]. Also, in recent versions of their famous core processors, Intel includes energy counters in their Intel Core processor chips and provides drivers, APIs and an SDK tool called Intel Power Gadget for developers of major operating systems to create real-time energy profiler for their green software projects [80]. Similarly, AMD has developed CodeXL [139], a debugging, profiling and analysis tool for taking advantages from their CPUs, GPUs and APUs (accelerated processing units). The tool also includes a power profiler that can provide power consumption of CPU, APU and GPU components in real-time, as well as frequencies and thermal trend. This tool can also be used to provide power visibility in making software greener. On the other hand, Qualcomm has also introduced Trepn Power Profiler for Android mobile devices with their Snapdragon processors [134]. With the power profiling tool, Android developers can have power/energy visibility of their applications and better understand the impact of their programming choices on both power and performance.

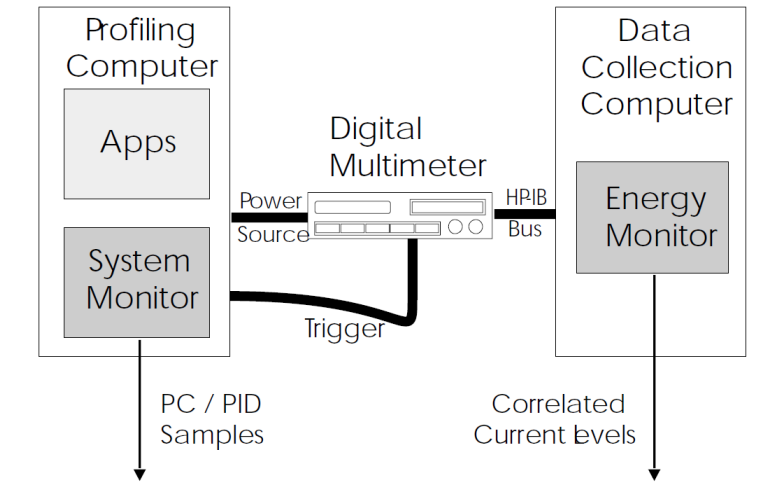
2.3.2.2 Power Models

Power models are considered software-based power measurement tools because they use a mathematical formula to estimate the power consumption of a system. Based on Chen and Shi [98], power models are built to estimate the power dissipation of different levels, such as instruction level, program block level, process level, hardware component level and system level. The methods capture power indicators that could reflect the power consumption of software applications or hardware components. The models are then built with these power indicators and fine-tuned the parameters of the power model for highest accuracy possible. The accuracy is normally verified by comparing with the result measured by a power meter or by applying the power information into a power-aware strategy to test its usability. Power models are more popular in energy-aware software research because it is more practical and suitable for use with or within software applications. However, its complexities and errors in the models can introduce inaccuracy making software-based power measurement less accurate than hardware-based [3, 42]. Also, many models are considered premature to be used in real-world applications. This section provides some examples of software/modeling-based power measurement tools.

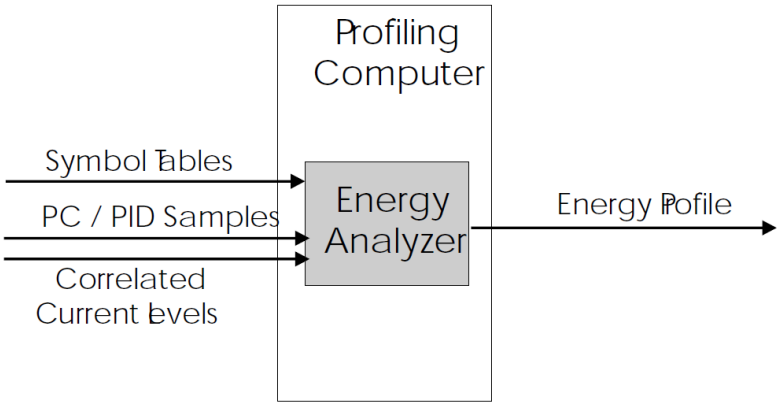
2.3.2.2.1 PowerScope

PowerScope [78] is an early-developed energy profiler that combines both hardware measurement and a statistical sampling method of system activity with kernel software support. The energy profiler maps energy consumption to program structure, in much the same way that CPU profilers, such as *prof* and *gprof,* map processor cycles to specific processes and procedures. Figure 4 is the architecture of PowerScope depicting

how a digital multimeter is integrated and how it generates an energy profile. As applications execute on the profiling computer, the System Monitor component samples system activity and records the value of the program counter (PC) and the process identifier (PID) of the executing process. At the same time, the Energy Monitor component samples voltage and current readings from the digital millimeter at the same corresponding rates controlled the multimeter. Subsequently, the Energy Analyzer component uses this information to generate an energy profile.



(a) Data Collection



(b) Offline Analysis

Figure 4. PowerScope Architecture (from [78])

The PowerScope Energy Analyzer component generates activity-based profiles by integrating the product of instantaneous current and voltage over time. The energy value is approximated by sampling the voltage V_t and current I_t at regular intervals of length Δt. The energy usage of the executing process over *n* samples is calculated by multiplying the summation of current samples with the measured voltage (V_m) and the sample interval (Δt) as shown in the following equation:

$$E \approx V_m \sum_{t=0}^{n} I_t \Delta t \quad (2)$$

The authors claim that, by using PowerScope, they are able to reduce the energy consumption of an adaptive video playing application by 46%.

2.3.2.2.2 Joulemeter

For a more recent energy profiling tool, Joulemeter [79] is a Microsoft's software-based energy profiler that estimates the power consumption of a computer. It tracks computer resources, such as CPU utilization and screen brightness, and estimates power usage. Joulemeter estimates the energy usage of a virtual machine (VM), computer, or software by measuring the hardware resources being used, such as CPU, disk, memory, screen, etc. The tool then converts the resource usage to actual power usage based on automatically learned realistic power models. This calibration process can be done with either the device's internal battery module or a Watts Up? meter [16]. Joulemeter is used for gaining visibility into energy use and for making several power management and provisioning decisions in data centers, client computing, and software design. At the time of writing this paper, Joulemeter is no longer available for download and supported by Microsoft. The company informs that similar energy profiling tool has been integrated in

the later versions of Visual Studio, a popular integrated development environment (IDE) from Microsoft.

2.3.2.2.3 Intel Power Gadget

The Intel Power Gadget [80] is one of the most recent power estimation tools that can also be used for energy-efficient software development projects. It can be considered as both hardware and software-based power usage monitoring tools since the power consumption data are derived and calculated from energy counters provided by Intel Core processors. The motivation for the tool is to assist end-users, ISV's (independent software vendors), OEM's (original equipment manufacturers), developers, and others interested in a more precise estimation of power from a software level without any hardware instrumentation. The latest version 3.0 tool is supported on Windows, Mac OS X and Linux, and includes an application, driver, and libraries to monitor and estimate real-time processor package power information in watts using the energy counters in the processor. With the latest release, the tool provides functionality to evaluate power information on various platforms including notebooks, desktops and servers. The callable APIs allow programmers to extract power information at finer granularity and within sections of program codes. The optimal sample rate is suggested at 100 milliseconds per sample. The data provided by the tool include processor power, package power limit, current processor frequency, base frequency, GT or GPU frequency, current temperature, maximum temperature, proc hot (when package temperature exceeds max temperature), timestamps and elapsed time. The data are obtained from the CPU's model specific registers (MSRs) and energy counters that are available only in 2nd-generation Intel Core or later processors.

There are also many other power/energy estimation tools—JouleTrack, a web based tool for software energy profiling [81]; PowerTop [82] that allows users to find programs that are consuming power when the computer is idle; PowerDail [83], a system for dynamically adapting application behavior to execute successfully in the face of load power fluctuations; Green Tracker [84], a tool that estimates the energy consumption of software in order to help concerned users make informed decisions about the software they use; and the more recent tool, POSE [85], a mathematical and visual modelling tool to guide energy aware code optimization. . Figure 5 shows some mathematical formula of component-level power models used in AppScope framework, an application energy metering tool for Android smartphone developed by Yoon et al. [140]. There are also many other tools not included here.

Component	Model
CPU	$P^{CPU} = \beta_{freq}^{CPU} \times u + \beta_{freq}^{idle}$ u: utilization, $0 \le u \le 100$ $freq$: frequency index, $freq = 0,1,2 \cdots, n$
LCD	$P^{LCD} = \beta_b^{LCD}$ b: brightness level, $\text{MIN}(level) \le b \le \text{MAX}(level)$
WiFi	$P^{WIFI} = \begin{cases} \beta_l^{WIFI} \times p + \beta_l^{base}, & if\ p \le t \\ \beta_h^{WIFI} \times p + \beta_h^{base}, & if\ p > t \end{cases}$ p: packet rate, t: threshold
cellular(3G)	$P^{3G} = \begin{cases} \beta_{IDLE}^{3G}, & if\ RRC\ state\ is\ IDLE \\ \beta_{FACH}^{3G}, & if\ RRC\ \text{state}\ is\ FACH \\ \beta_{DCH}^{3G}, & if\ RRC\ state\ is\ DCH \end{cases}$
GPS	$P^{GPS} = \beta_{on}^{GPS},\ if\ GPS\ is\ on$

Figure 5. AppScope's Component-Specific Power Models (from [140])

2.4 Performance, Power and Energy Optimization of Software Applications

Like execution time or performance, power and energy consumption are considered non-functional properties of software applications. The two properties are becoming more important among battery-driven devices because improving the energy efficiency in software applications can help extend the device's battery life. However, there is still some confusion and questions about energy optimization of software applications. Based on the energy formula ($E = P \times t$), minimizing the energy consumption can mean either minimizing the execution time, the power consumption or both. One may argue that improving the energy efficiency can be done by just improving the performance or making the code run faster (minimizing the execution time). However, this is not always the case because optimizing the code to run faster can sometimes increase or decrease the power consumption of software applications and can result in higher or lower energy consumption. As mentioned in one of our published papers [3], optimizing the power and performance of a software system is not simple. This is mainly because the two properties are considered conflicting attributes and are often traded off. Our study results provide some evidences of counterintuitive results showing that some system cache configurations can result in faster execution but the power consumption sometimes remains unchanged or is lower or higher.

Moreover, Abdulsalam et al. conduct a comprehensive experiment on using different techniques to optimize software programs and propose the Greenup, Powerup and Speedup metrics (GPS-UP) to categorize the software implementation and optimization efficiency [102]. The definitions of the metrics are explained in Section 2.3.1.1. Figure 6

shows the GPS-UP software energy efficiency quadrant graph. The "green" categories (categories 1-4) represent energy savings and the "red" categories (categories 6-8) represent energy loss compared to the corresponding non-optimized code. For example, category 1 is specified as Powerup < 1 and Speedup > 1, meaning that the optimized code runs faster and consumes less power consumption. On the other hand, category 4 is specified as Powerup < 1 and Speedup < 1. This category indicates that the optimized code is greener, consumes less power but runs slower. The goal of their study is also to try to answer the following questions:

1) "Is performance efficiency equivalent to energy efficiency?"

2) "Is there a win-win situation for both performance and power consumption?"

3) "Is there any optimization that helps energy more than performance?"

4) "What are the correlations between performance, power and energy when optimizing software?"

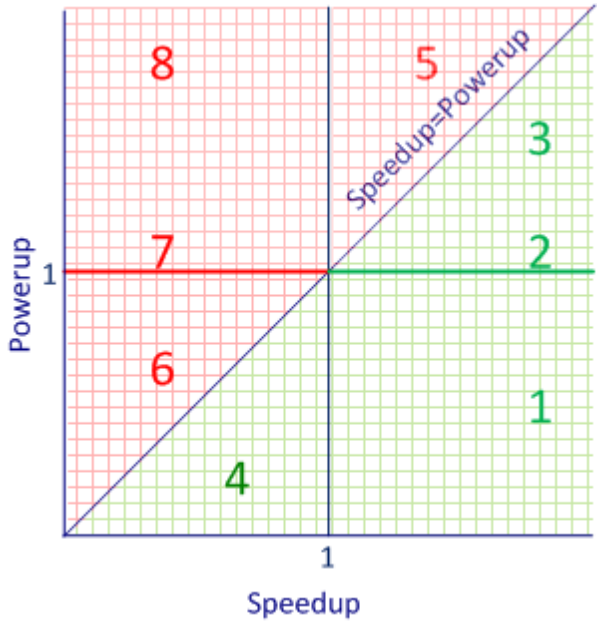


Figure 6. GPS-UP Software Energy Efficiency Quadrant Graph (from [102])

Using a number of different implementations of four selected algorithms, the authors observe the relationship between GPS-UP metrics and are able to find examples for all eight categories and show that some optimization techniques can help energy more than performance. These GPS-UP metrics can also be used to analyze the correlations of energy, power and performance when optimizing software applications. The ideal optimization techniques can be identified when the optimized code is in the category 1—less power consumption and faster execution. The authors also recommend that we combine the Greenup, Powerup and Speedup in analyzing the impact of language choices, compilers, and optimization techniques on execution time and energy consumption of software applications. However, in our adaptive data structure study in this dissertation, the energy optimization impact is analyzed using the Greenup metric, looking only at the overall energy impact from using our green data structure, without specifically analyzing the Powerup or Speedup metrics.

CHAPTER 3: GREEN SOFTWARE DEVELOPMENT

Originally, energy efficiency research in computer systems has been driven mainly by battery constraints of ubiquitous mobile devices and notebooks. Many energy efficiency tools and methods were developed primarily to help extend battery lifetimes of the devices. The main purposes were toward the saving of operational costs of data centers and extending of battery lifetime of mobile devices. However, today the trends and goals of the research are toward more environmental sustainability; reducing carbon footprints and the fight against climate change. Besides concentrating on the hardware components, aiming on software components and application layers of computer systems is also gaining popularity among many researchers. However, the research area is still in the early stage and there is much room for improvement. This chapter discusses the background and current development of green software research and some of the related fields, software adaptation for energy efficiency and developing green software using machine learning methods.

3.1 Sustainability in Software Engineering

In recent years, Sustainable or Green Software Engineering (SSE or GSE) has been gaining importance in the software engineering community. Many researchers are now realizing the direct and indirect effect of software applications to the system's energy consumption and the environment. Their efforts include developing software engineering methods and promoting green software developments among software engineers and

developers. The main purpose is to include sustainability and greenness into every phase of the software engineering process and consider it as a software quality metric or non-functional property of software applications.

3.1.1 Framing Sustainability as a Property of Software Quality

Based on the Collins dictionary [105], the definition of sustainability is "the ability to be maintained at a steady level without exhausting natural resources or causing severe ecological damage". Similarly, the Brundtland report from the United Nations (UN) [106] defines sustainable development as the ability to "meet the needs of the present without compromising the ability of future generations to satisfy their own needs" [62]. Based on the UN definition, sustainable development needs to satisfy the requirements of three dimensions, which are the society, the economy and the environment.

In addition, based on an IUCN's technical report by Adams [93], the author defines the three dimensions in sustainable development into three pillars of sustainable development that can also be applied in IT development:

1) **Pillar 1: Economic Development**—to ensure economic growth, maintain a healthy balance with our ecosystem and integrate environmental and social concerns into business.
2) **Pillar 2: Social Development**—to create a sustainable society which includes social justice or reducing poverty.
3) **Pillar 3: Environmental Protection**—to ensure that the environment is protected by human actions and help the environment to be able replenish itself; e.g., the use of recycled materials to help conserve natural resources.

In extending the above principle, Lago et al. [26] introduce the fourth pillar, the Technical pillar, for supporting long-term use and evolution of software-intensive systems at a level of abstraction closer to implementation. Their suggestion is that sustainability is achievable only when accounting for all dimensions. The authors also propose a framework to address the environmental dimension of software performance by demonstrating the use of the framework in a paper mill and a car-sharing service project. Figure 7 displays their framework for sustainability software-quality requirements. The framework addresses how these concepts relate to software and how to break down the respective concerns into software-quality requirements.

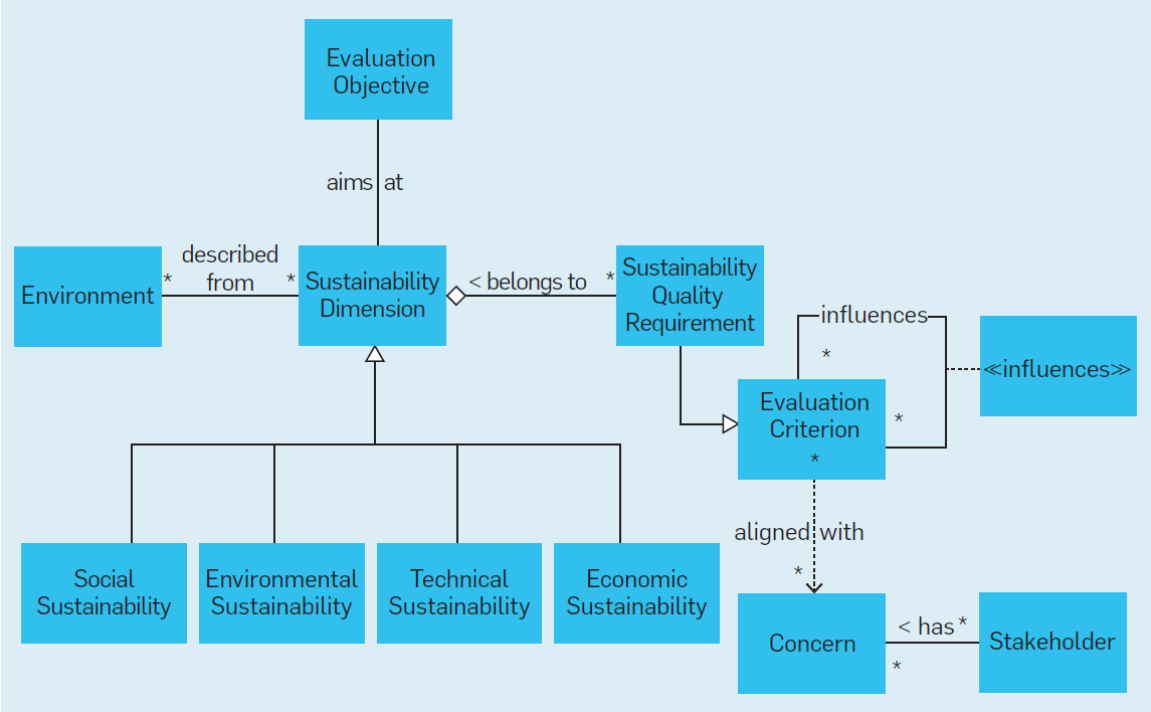


Figure 7. A Framework for Sustainability Software-Quality Requirements (from [26])

To frame software qualities, the framework positions the qualities into the four sustainability dimensions and relate them to the concerns of the relevant stakeholders.

When starting a project with a requirement on sustainability as shown in Figure 7, where the association aims to link the evaluation objective to the sustainability dimension, software developers have to resolve any concern and trade-offs among the various qualities classified as belonging to each of the four dimensions. In contrast with the traditional software decision making that considers trade-offs either between different technical sustainability criteria (such as performance versus availability) or between any of the four dimensions, the framework suggests sustainability-related software decision making involves trade-offs between environmental sustainability criteria (such as energy efficiency) and social, economic, and technical sustainability criteria. The authors also demonstrate the use of the framework in case-study examples.

In the paper-mill control system case study, there can be three main stakeholders that are concerned about different problems: for example, the surrounding community and society that are concerned about environmental sustainability like forest sustainability; customers that are concerned about economic sustainability like production savings expressing productivity and economic value creation; and producing organization, including managers and engineers, that are concerned about technical sustainability like optimization of configurability and performance. The interdependent quality requirements may influence one another, as in association/association class influences among sustainability quality requirements. In this case, performance and energy savings could influence each other, while increasing performance could demand more resources that consume more power and have a negative effect on energy savings. Using the framework can help designers of software-intensive systems appreciate the importance of the various

qualities and developers keep track of the elements captured by the framework when making the trade-offs among the various qualities in the four sustainability dimensions.

The term sustainability is sometimes analogous to “green” or “greenability” and also used in many sub areas of green computing research. For example, in a book by Calero and Piattini [107], the authors include the definitions for sub areas of green computing terms, such as information system (IS) Sustainability, ICT/IT sustainability and software sustainability. However, their main concepts are the same as the above definitions of sustainable development. This dissertation focuses more on the software sustainability and green software engineering, with the main goal to motivate green software development among researchers, programmers and other stakeholders that are involved in software development projects.

3.1.2 Sustainable and Green Software Engineering

Based on a book by Calero and Piattini [107], software engineering sustainability is considered a part of software sustainability. The authors state that software sustainability can be applied in many areas, such as software systems, software products, web applications, data center, etc. The book also states that the term sustainable software can be interpreted in two ways: (1) “the software code being sustainable, agnostic of purpose”, or (2) “the software purpose being to support sustainability goals”. Therefore, in this context, sustainable software is energy-efficient, minimizes the environmental impact of the processes it supports, and has a positive impact on social and/or economic sustainability. Naumann et al. also give a definition of sustainable software as “software, whose direct and indirect negative impacts on economy, society, human beings, and

environment that result from development, deployment, and usage of the software are minimal and/or which has a positive effect on sustainable development" [24]. The direct impacts are related to resources and energy consumption during the production and use of software, while indirect impacts are effects from the software product usage, together with other processes and long-term systemic effects.

In addition, Calero and Piattini also define green software as environment-friendly software that helps improve the environment. They also classify green software into four categories:

1) Software that is greener because it consumes less energy to run
2) Embedded software with smart operations that can assist other parts in going green
3) Sustainability-reporting software or carbon management software
4) Software for understanding climate change, assessing its effects and forming appropriate policy responses

For software engineering sustainability, Naumann et al. [24] subsequently define green and sustainable software engineering as the art of developing green and sustainable software engineering process, so that the negative and positive impacts on sustainable development that result and/or are expected to result from the software product over its whole life cycle are continuously assessed, documented, and used for a further optimization of the software product. However, as the research in green software engineering progress, there are also other definitions given by several researchers. For example, stated by Calero and Piattini [107], green and sustainable engineering is the enhancement of software

engineering, which targets; (1) the direct and indirect consumption of natural resources and energy, and (2) the aftermath that are caused by software systems during their entire life cycle, the goal being to monitor, continuously measure, evaluate and optimize these facts. The aims for sustainable software engineering are "to create reliable, long-lasting software that meets the needs of users while reducing the negative impact on the economy, society and the environment." Moreover, researchers from the University of California, Irvine, state that the aim of software engineering for sustainability (SE4S) is to support all dimensions of sustainability—human, social, economic, environmental, and technical—throughout the software lifecycle [108].

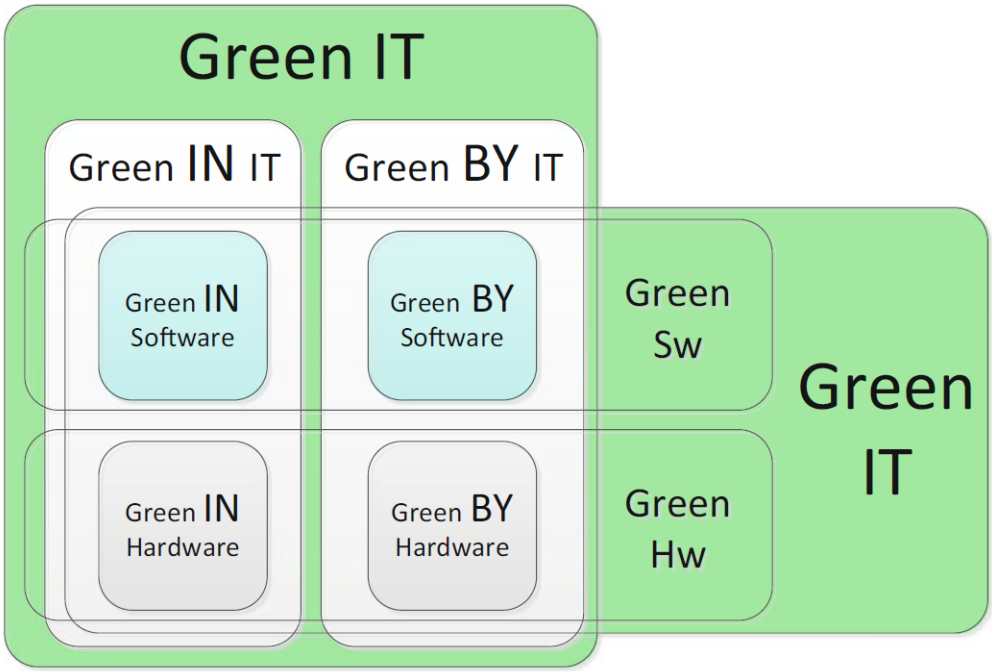


Figure 8. Green Software, Green Hardware and Green IT (from [107])

Because there are discrepancies between the concepts and the meanings given by many different authors, Calero and Piattini also present a simple diagram (Figure 8) to unify the different terminologies and definitions that together make up "Green IT." The

figure shows where software sustainability and green software engineering can be put in the green IT research area along with the distinctions from other areas in software and hardware layers. Based on the figure, our research area of green software development in this dissertation can be put in the "Green IN Software" research area, in which the main goal is to develop software applications that consume less energy to run.

3.2 Green Software Development Life Cycle

The trend of green computing has been changing in the last few years, and new pieces of work related to the area of green software development are emerging. There are several research studies that attempt to standardize the green software engineering practices. This section demonstrates two examples of the green software engineering models.

3.2.1 GREENSOFT

In order to classify and sort some aspects of green and sustainable software and its engineering, Naumann et al. develop a conceptual reference model named GREENSOFT model for sustainable software [24], shown in Figure 9. The model contains four parts that cover; (1) the life cycle model of a software product; (2) criteria and metrics that represent sustainability aspects that are directly and indirectly related to the software product; (3) procedure models for the different phases; and (4) recommendations for action, as well as tools for different stakeholders. The four-part model supports software developers, administrators, and software users in creating, maintaining, and using software in a green manner.

In the first part, the life cycle of software products, the authors follow the concept found in the Life Cycle Thinking (LCT) according to the "from cradle to grave" principle [109]. The life cycle of green software products can also be considered at the development and the usage phases, up until the end of life of software products.

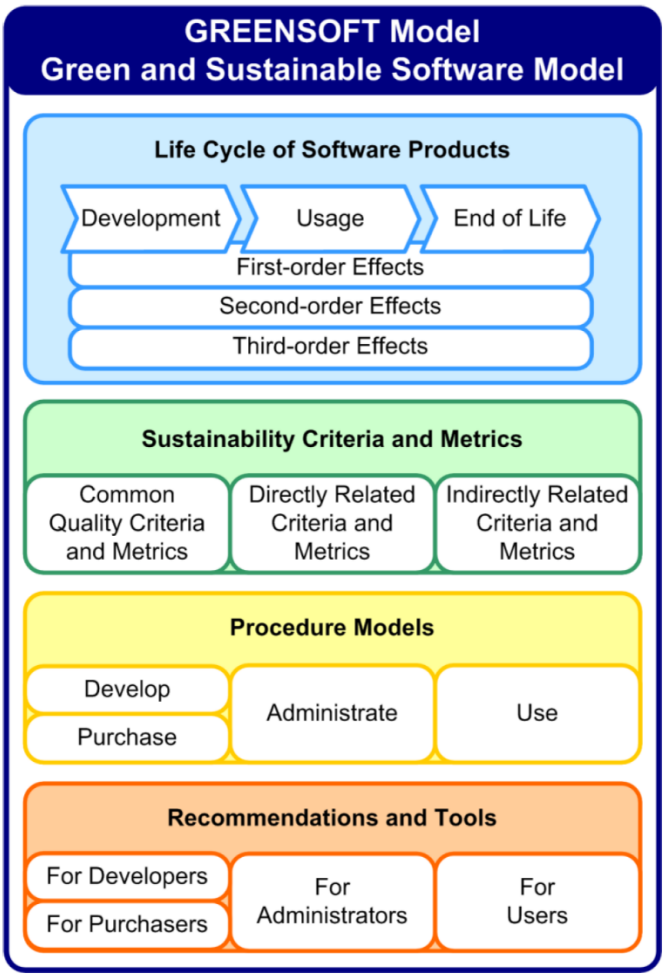


Figure 9. GREENSOFT Reference Model (from [24])

The intention is to estimate the ecological, social, and economic impacts that already occur in early stages during the software's whole life cycle. The sub model also shows an overview for the life cycle of software and its relationship to different levels of

effects—first order effects (effects resulting directly from the product, e.g. energy consumption), second order effects (usage results, e.g. effects of dematerialization by software), and third order or rebound effects (e.g. when an energy-efficient product leads to more energy consumption in total).

In the second part, the authors present some sustainability criteria for software products. The model categorizes the criteria into three categories—common quality criteria and metrics, directly related and indirectly related criteria metrics. The part involves mainly measurements of common effects of software products which are considered important for developing green software. In this part, the authors also propose a quality model and introduce some quality criteria terminologies such as, *Efficiency*, *Energy Efficiency*, *Runtime Efficiency*, *CPU-Intensity*, *Memory Usage*, *Peripheral Intensity*, *Idleness*, *Number of Methods, Framework Entropy, Functional Types*, and so forth.

For the third part, the sub model contains procedure models, based on the different usage types of different stakeholders such as, developers, purchasers, administrators, and users. The proposed models suggest that software engineering should become green and sustainable in its production, support, and application processes. Lastly, the forth sub model comprises recommendations for action and tools for the different stakeholders, such as checklists, guidelines, best practice examples, software tools, as well as other tools that speedup and improve the green software development processes. These support stakeholders with different professional skill levels in applying green or sustainable techniques in general, when developing, purchasing, administrating, or using software products. Our proposed green data structure in this dissertation can also be considered as

one of the support tools, particularly for programmers to use in developing green software applications.

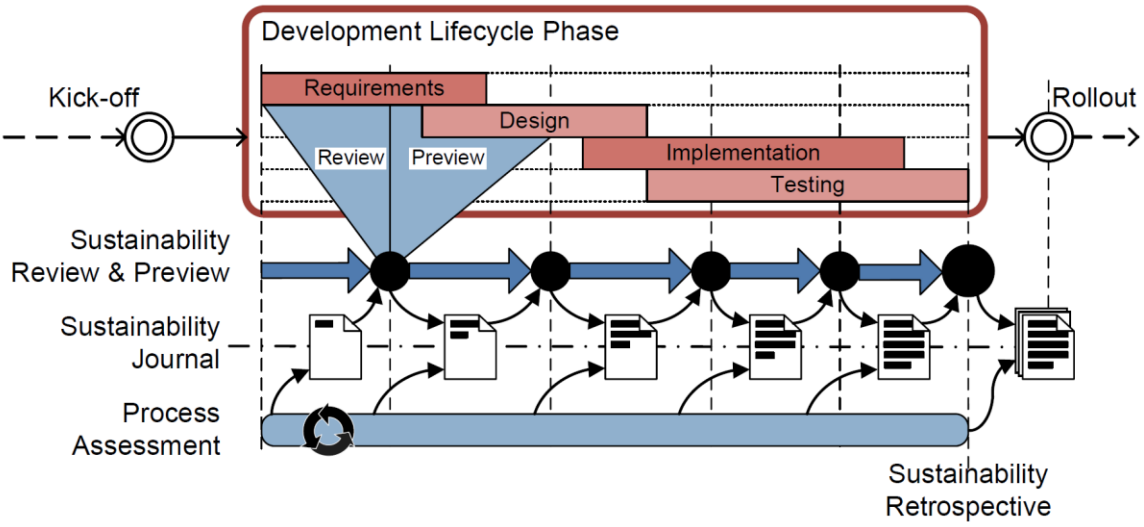


Figure 10. An Overview of Green Software Engineering Process Model (from [24])

As additional detail of the first sub model, software product life cycle, the authors also give an overview of a green software engineering process model, shown in Figure 10, which incorporates the green software development practices with the traditional software development life cycle (SDLC). This general process is enhanced by several activities that have the objective to enable sustainable software engineering. The sustainability reviews & previews mainly consider impacts on sustainability which are expected to arise from distribution and future use of the software products. The sustainability journal is the information hub of the process enhancements. It is a well-structured report, which evolves together with the software project. Its purpose is to document sustainability reviews & previews, process assessment and the sustainability retrospective. Finally, after the project has finished, it reports the assessed impacts on sustainability.

3.2.2 A 2-Level Green Model for Sustainable Software Engineering

In addition to the GREENSOFT model, Mahmoud and Ahmad [25] propose another software model that covers all aspects of software related to green computing. The model is a two-level model in which the first level is a hybrid software engineering process between sequential, iterative, and agile software development processes that aims to create a green and sustainable software process; and the second level explains how software itself can be used as a tool to aid in green computing by monitoring resources in an energy efficient manner.

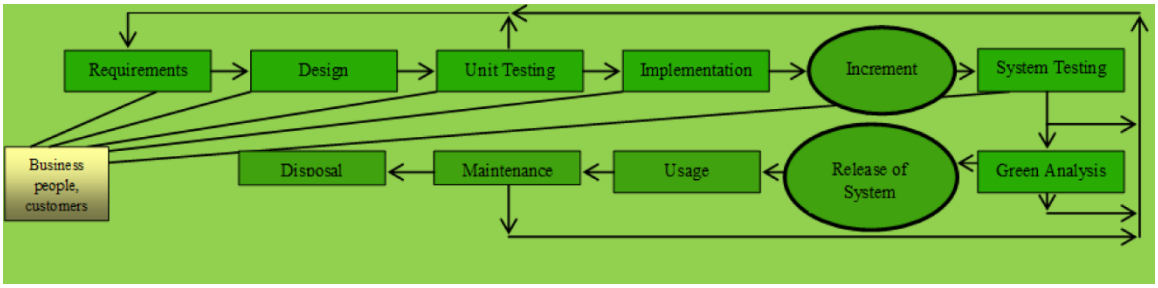


Figure 11. Level-1 Green Software Engineering Process (from [25])

In the first level shown in Figure 11, the Level-1 model consists of nine main stages that can aid in producing a green product and is designed to have environmentally sustainable stages—requirements, design, unit testing, implementation, system testing, green analysis, usage, maintenance, and disposal. This first level represents how to obtain a green and sustainable product. The "Business, people, customers" box in the figure indicates that the customers and the business people should be part of the requirements, design, implementation, and testing stages.

Each of the nine stages contains some sub stages and can sometimes be iterated back to the previous steps and repeated multiple times if necessary. For example, starting

with the green requirements engineering stage, this stage is the energy efficient requirement engineering process. It consists of a feasibility study to determine if the system to be built is relevant and useful to the business, including whether to include energy efficiency as a non-functional requirement. The requirement process also includes outlining and organizing the services in the order they should be developed and a risk analysis inspired from the iterative spiral model but in terms of energy. After the risk analysis, the process can go back to the requirement outlining stage to implement these changes if necessary. At the final stage of the requirements process, the requirement test stage is to be conducted to an environmentally sustainable requirements process. It is energy efficient to develop tests along with requirements because it provides a better understanding to the testers and developers of the requirements and mainly satisfies that there will be no changes when system and acceptance testing occurs.

In the green design stage, a system architecture is created based on the requirements. During this stage, fundamental software system abstractions and their relationships are defined. There are a number of design activities that form the sustainability level of the software component such as architectural design, abstract specification, data structure design, and algorithm design. The system is then implemented into a set of programs and program units based on the designs. Software developers should choose at this stage the most suitable programming patterns or algorithms to the application. The software testing process can emphasize on either discovering that the software does not meet its requirements or can emphasize on discovering faults or defects in the software where the behavior of the software is incorrect.

The green analysis stage is to promote energy efficiency and brings forth new ideas about environmental sustainability to any software engineering process. The green analysis stage determines the greenness of each increment of the system that is developing. This stage acts like a testing stage but for energy efficiency. Energy metrics are used in this stage to perform the analysis. The usage, maintenance and disposal stages are also conducted in a green manner.

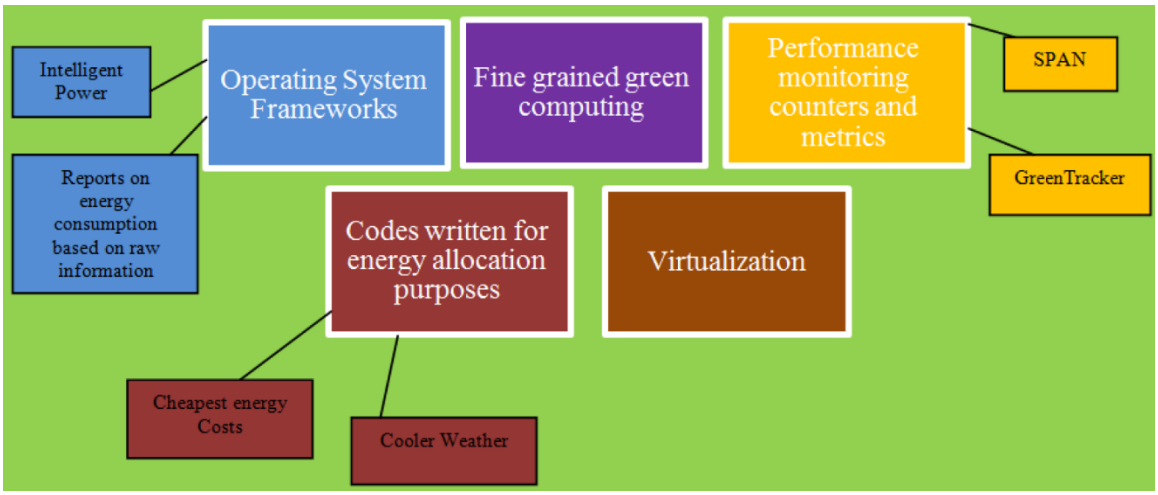


Figure 12. Level-2 Software Model that Promotes Green ICT (from [25])

In Figure 12, the Level-2 model indicates how software tools can play a major role in having energy efficient use of software applications thus promoting green computing. This model consists of five categories—operating system frameworks, fine grained green computing, performance monitoring counters and metrics, code written for energy allocation purposes, and virtualization. For example, the operating system frameworks that create intelligent power profiles are integrated into the operating system code to minimize power consumptions of computer systems. The energy profiles can be used to minimize the average work load of the CPU by shutting or hibernating applications not in use so that heat dissipation and power consumption is minimized. For performance monitoring

counters and metrics, the authors give Span [110] and GreenTracker [111] as examples of power estimation tools that can be used for energy efficiency improvement of computer systems. The fine grained green computing approaches are more specific to a running application such that power consumptions can be measured at component levels such as memory banks or I/O peripherals. The fine grained data can help the applications to improve energy efficiency better. The general approaches are codes written specifically for energy allocation purposes that can route traffic to locations such as data centers with the cheapest energy costs or ones with cooler temperatures. The virtualization also plays a role in green computing and is partly software. Virtual machines are partitioned based operating systems that allow for multiple applications to exist on a single system. This approach can reduce the number of systems needed and the amount of power required, thereby contributing to green computing.

In our main study, the proposed adaptive green data structure can be put in both levels of the green model. If we consider our green data structure as a green programming tool that programmers can use to develop green software application, then the approach can be viewed as the Level-1 model—in the design and implementation stages, in particular during the selection process of a green design architecture, tools and algorithms. If the green data structure is viewed as a software tool that aids in green computing and software development, it can also be put in the Level-2 model—possibly in a fine grained green computing group.

3.3 Software Adaptation and Energy-Aware Applications

Due to the growing numbers of modern and complex applications with the ability to adapt to different users, environments, platforms and/or screen sizes of mobile devices, software adaptation is emerging as a new discipline in the software engineering field. According a paper by Canal et al. [112], software adaptation refers to "a process, in which an interactive/adaptive system adapts its behavior to individual users based on information acquired about its user(s) and its environment". In a more specific definition in software engineering term, software adaptation promotes the use of adaptors—specific computational entities whose main goal is to guarantee that software components are able to interact in the right way not only at the signature level, but also at the behavioral, semantic, and service levels. There are many purposes for software adaptation—interoperability, usability and improving performance, quality of service or energy efficiency, among others.

For interoperability purpose, at the signature level, a software component is designed so that its interfaces or the name of the service, type of its arguments and return values, and the possible exceptions raised, that is, the full signature of the component can be reused many times in developing many other applications. At the behavioral level, the component has the behavior or protocol as expected or specified by the interfaces. The mismatched behavior can make the component to be incompatible or have no behavioral adaptability. At the semantic level, the compatible component has to be designed and implemented correctly with correct formal functional descriptions and language semantic. And at the service level, typical aspects that can be adapted are synchronization, security,

persistence, and so forth. For usability purpose, software components are to be designed/implemented so that it can adapt to the users and environments such as platforms, internet connectivity, screen size and other usability requirements. In addition, a study by Oreizy et al. [113] shows how an application can be adapted at runtime by manipulating its architectural model. In particular, the paper demonstrates how software connectors in aiding runtime change, provides an explicit architectural model fielded with the system and used as the basis for runtime change, and suggests architectural style in providing both structural and behavioral constraints over runtime change.

For green software development, software adaptation is the ability of software to adapt and be reconfigured, changed or transformed for energy efficiency. This adaptation can be either in a manual or automatic manner. For example, an adaptive online video player is designed/implemented so that it can be manually or automatically reconfigured for the video content to be streamed at different quality levels depending on the strength of the internet connection or the battery lifetime of a computing device. For manual reconfiguration, the process is normally done offline by users or programmers at design times. But, automatic reconfiguration is normally done at runtime so the software can adapt dynamically and automatically. Sometimes, the online or dynamic adaptation requires some types of intelligence for decision making of when and how to adapt.

For modern software systems, many studies focus on software adaptation for energy efficiency. The software adaptation capability sometimes is mechanically included in energy-aware applications. The energy-aware applications normally include energy monitoring capabilities (so they know the energy impacts from its processes) and runtime

adaptability for energy efficiency. For example, a study by Flinn and Satyanarayanan [13] demonstrates how mobile applications can dynamically modify their behavior to conserve energy. One experiment in the Linux operating system shows that the operating system can guide such adaptation to yield a battery life of desired duration. By monitoring energy supply and demand, it is able to select the correct tradeoff between energy conservation and application quality. Their evaluation result shows that this approach can meet goals that extend battery life by as much as 30%.

In a recent study by Hoffman [114], the author combines approximate applications and energy aware systems to create JouleGard, a runtime control system that coordinates approximate applications with system resource usage to provide control theoretic formal guarantees of energy consumption, while maximizing accuracy. JouleGuard is evaluated by testing on three different platforms (a mobile, tablet, and server) with eight different approximate applications created from two different frameworks. The result shows that JouleGuard respects energy budgets, provides near optimal accuracy, adapts to phases in application workload, and provides better outcomes than application approximation or system resource adaptation alone. For another study that focuses on software adaptation for energy efficiency, OpenMPE [115] is an extension to OpenMP designed for power management. OpenMP is a standard for programming parallel shared memory systems without any support for power control. The OpenMPE exposes per-region multi-objective optimization hints and application-level adaptation parameters, in order to create energy-saving opportunities for the whole system stack. The evaluation results demonstrate the

effectiveness of OpenMPE with geometric mean energy savings across 9 use cases of 15% while maintaining full quality of service.

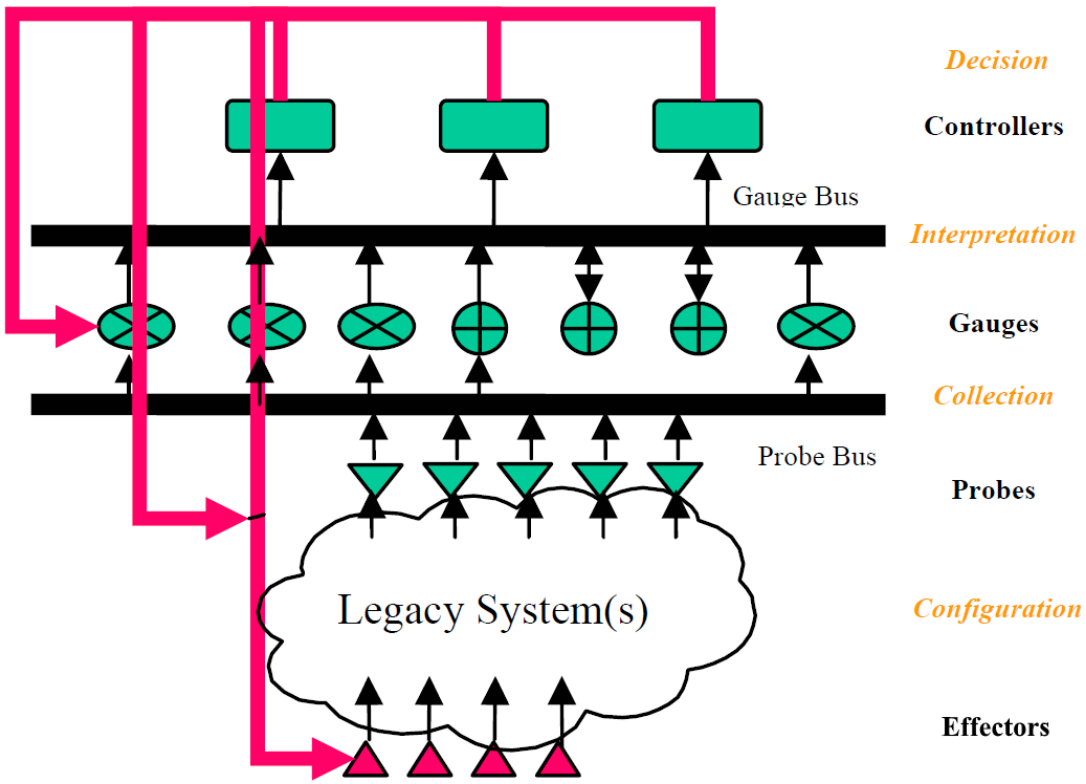


Figure 13. Idealized Infrastructure for Dynamic System Adaptation (from [116])

There are many types of system infrastructures and architectures for software adaptation; for example, Kinesthetics eXtreme (KX) for applying dynamic adaptation facilities "from the outside" of a given target system [116]; and KX Feedback-Control-Loop infrastructure that extends the KX architecture with a feedback-control-loop infrastructure. However, in general, an adaptive system composes of monitoring, dynamic analysis, decision making and feedback to reconfiguration components. As seen in Figure 13, it is an idealized infrastructure for system adaptation with the "Feedback-Control-Loop" infrastructure being added to the legacy systems. This infrastructure adds the ability

for dynamic adaption to the legacy systems. In the figure, initially, data is collected from the running target system. It is instrumented with non-invasive probes that report raw data to other layers via the probe bus. The data is then interpreted via a set of gauges that map the probe data into various models of the system. The gauges then report their findings to the gauge bus. Then the decision and control layer can analyze the implications of the interpreted data on overall system performance and make decisions on whether to: (1) introduce new gauges in the interpretation layer to analyze further, or disable some as unneeded; (2) deploy new probes to provide more detailed information to the remaining gauges, or turn some off to reduce "noise"; and/or (3) reconfigure the system itself, perhaps changing the running system's structure by introducing new modules or modifying system or component parameters.

The adaptive green data structure in our study and other energy-aware applications also have similar infrastructure. There is an energy monitoring component for energy profiling and collecting energy data. There is also a data analysis for data interpretation and a decision making component for making decisions on when and how to adapt. There is also a feedback mechanism to notify different parts of the system to reconfigure, transform or adapt for energy efficiency and extending battery life. Also similar to the idealized architecture, our green data structure adds the Green component to the existing dynamic data structures so that they have the ability to learn and adapt for energy efficiency. However, in software applications, the main challenges are (1) how to add power monitoring components since, at the time of writing this dissertation, there is no such tool for measuring energy impact at the software object level, and (2) how to add the

data analysis and decision making capabilities without introducing overhead due to the additional computation. Our approach to solving the problems and overcome the challenges are to use machine learning technology and make the decision mechanism as lightweight as possible.

3.4 Machine Learning for Energy-Efficient Computing

Machine learning is a field in computer science that is becoming more and more popular among many research areas such as gaming, natural language processing, data science and robotics. Based on a book by Bekkererman et al. [44], machine learning is about developing algorithms for making predictions from data. The purpose of a machine learning task is to identify (or to learn) a function $f: X \rightarrow Y$ that maps the input domain X (data) onto output domain Y (of possible predictions). The function f is selected from a certain function class, which is different for each learning algorithm. X and Y are the domain-specific representations of data objects and predictions, respectively. For learning algorithms, there are two main types—supervised and unsupervised learning. Supervised learning algorithms require training data to create a function f that produces accurate predictions on test data. Instead, unsupervised learning algorithms aim to construct predictive functions that generalize or describe hidden structure of unlabeled or unseen data.

Two famous examples of supervised learning tasks are classification and regression. Classification tasks have the output Y as discrete set of categories (or classes), $Y = \{c_1, c_2, c_3,..., c_k\}$, whereas, regression tasks has the output Y as real numbers. One famous example of the unsupervised learning is data clustering. The goal of data clustering

is to construct a function *f* that partitions an unlabeled dataset into clusters, with *Y* being the set of cluster indices. In our green data structure study, supervised learning technique is used for training our green data structure, and classification technique is used when predicting energy efficient data structures from the workload. In general, machine learning gives computers the ability to learn and make predictions without explicitly being programed.

Today, machine learning is being widely used in many computer fields and is another important methodology for sustainability as well. There are many research and products that use machine learning techniques for improving energy efficiency; for example, intelligent agents in smart meters and the Smart Power Grid [117]; ThinkHome [118] and Smart Buildings [119] for optimizing energy bills in homes and buildings; and smart cars [120] for optimizing fuel consumption and extending battery life of electric cars. For green computing research, there are also several research studies that make use of stochastic search for improving energy efficiency of computer systems and for developing green software applications. This section provides some research examples that uses such techniques for improving energy efficiency of computer systems and software applications. First, a research by Lorenz et al. [121] focuses on the compiler layer, in particular the code optimizer in embedded systems. The authors propose an energy-aware code generator (GCG) based on single population genetic algorithm. This code generator reduces the energy consumption by suitable instruction selection and instruction scheduling. Energy-aware compilation is done with respect to an instruction level energy cost model which is integrated into the code generator and simulator. Their method is to decompose a source

program into basic blocks of procedures and present them as nodes in a data flow diagram (DFG). The genetic algorithm module encodes the basic blocks into specialized chromosomes. Each gene of the chromosome represents an operation like a load or an addition. The values of a gene express information about used registers, performed processor instruction, execution cycle, and others, which are necessary for code generation. An objective function is defined as the consumed power or energy of a program. It is represented by values of average power dissipation of certain combinations of instructions. The authors used their method for SIMD instructions (SIMD refers to single instruction multiple data). The evaluation results show a 30% of energy reduction and 8% reduction of the application code.

Furthermore, similar code optimizer research by Azzemi [122] uses a simple multi-objective genetic algorithm (MOGA) in their optimization and achieve an energy reduction of about 17%. Meedeniya et al. [123] try to solve the redundancy allocation problem in the embedded systems by using the Markov Reward Model [124] for system representation. The authors use the non-dominated sorting GA (NSGA) algorithm to solve a bi-objective optimization problem between system reliability and energy consumption. The achieved empirical results show that the proposed method can significantly reduce the energy consumption for a very small trade-off of reliability.

For network data transmission, dynamic data compression in the application code seems to be a promising software tool for saving the energy used for data propagation in wireless sensor networks. Compression methods exploit the data structure and reduce the data size. Marcelloni and Vecchio [125] perform a data compression on a network (single)

node based on a differential pulse code modulation scheme with quantization of the differences between consecutive codes of the signal samples. The trade-off between a performance of compression algorithm and the amount of the lost information is determined by the set of quantization parameters. The authors employ the Non-Dominated Sorting Genetic Algorithm II (NSGA–II) for optimizing the combinations of these parameters corresponding to different optimal trade-offs. The evaluation analysis of the proposed method shows a 62% reduction of the energy consumed in data transmission.

Similarly, Liu et al. [126] propose a MAC layer solution called pushback that appropriately delays packet transmissions for energy efficiency in sensor networks. The method is to overcome periods of poor channel quality and high interference, while ensuring that the throughput requirement of the node is met. It uses a hidden Markov model (HMM) based channel model that is maintained without any additional signaling overhead scheme. The pushback algorithm is shown to improve the packet success rate by up to 71% and reduce the number of transmissions needed by up to 38% while ensuring the same throughput.

A recent article by MIT News [129] reports that MIT researchers have built an energy-friendly chip that can perform powerful learning tasks. The new chip is designed specifically to implement neural networks that enable future mobile devices to model the human brain. It is 10 times as efficient as a mobile GPU so it could enable mobile devices to run powerful learning algorithms locally rather than uploading data to the internet for processing. Also stated in the article, neural nets were widely studied in the early days of

machine learning research, but by the 1970s, they had fallen out of favor. In the past decade, however, they have come back under the name “deep learning.”

CHAPTER 4: A POWER-PERFORMANCE TRADEOFF STUDY

As our first study and an initial exploration in green computing research, this chapter presents a power-performance tradeoff study of the cache system in computer hardware systems. The purpose is to see how the cache system impacts the energy consumption of a computer system and how the energy efficiency can be improved by using a Pareto tradeoff method. In the study, we conduct an empirical evaluation of the power/performance impact of cache configuration on embedded systems. We gather live power consumption and execution time data for the programs in the CHStone benchmark suite on an embedded processor with configurable cache parameters and perform a Pareto analysis on these data to identify the optimal cache configurations. We observe that the optimal configurations are sparse in the design space, are inconsistent across the benchmark, and are counterintuitive in some cases. Our results reveal interesting, unexpected insights motivating the need for tools and methodologies that automate this process and operate directly on data gathered from the systems.

4.1 Introduction

Power-performance optimization is challenging and becoming increasingly important among modern computer systems, especially for those that rely on battery power. The sophistication of software applications and the increasing needs of rich media and big data have made today's computer systems power-hungry, while battery standards are not keeping pace with the demand [49]. Therefore, many researchers have been developing

optimization techniques to extend battery life and reduce power consumption while maintaining other performance characteristics at acceptable levels.

Many power reduction techniques are based on power models which might not represent the full complexity of the system being analyzed. Most computer systems are not originally designed to support power optimization so the onboard power monitoring systems are not included, or if included, they are not explicitly designed to measure the power consumption of software applications [42, 50]. Many power models have been developed to support power optimization [39, 45, 48, 50, 51]. They are mostly intended to evaluate a specific platform or specific technology [51]. As with all models, if there are errors with calibration or inaccuracies in the models, or if they are used incorrectly, the results can be skewed or different from those based on analysis of live power consumption data [42]. In order to avoid the use of power models, we focus on the use of live data.

Optimization with live data is difficult: the process of gathering and analyzing these data is tedious and understanding conflicting performance attributes is challenging. In software engineering, performance and power consumption are viewed as non-functional properties. They are considered conflicting attributes and are often traded off, making them difficult and time consuming to optimize [41]. Many researchers point out that high-level strategies can help in trading off the conflicting properties and solving the multi-objective optimization problem [33, 41]. Although their results are intuitive and feasible, there are still many open challenges and the strategies are far from being adopted into practice.

In this paper, as a case study, we conduct an experiment on an embedded hardware platform that can run a wide variety of software applications while providing live power consumption data. We investigate one aspect of the system, the cache system, because it has a major impact on both power consumption and execution time, and virtually all computer systems use caches [45]. Several other studies have shown that the cache has a large effect on the overall system performance and also accounts for a large amount of total energy consumption in embedded systems; up to 50% of total energy usage in some cases [48]. Also, there are many tunable parameters in most cache designs [46, 48].

We select Pareto optimality as the main principle to solve the bi-objective optimization problem because it is well-known and has been applied in many fields, including engineering and economics where optimal decisions need to be made in the presence of tradeoffs between two or more conflicting objectives [36]. Our goals are to demonstrate a detailed manual optimization process and to convey the basic concept of power-performance tradeoff in an energy-aware system and to understand the impact that different cache parameters have (or do not have) on the power/time tradeoff in order to better understand how an automated optimization methodology for performing this analysis might work.

We consider power consumption data (watts) for the analysis instead of energy (joules) because we want to look at the system's power consumption and performance as a whole and not the specific software being executed. We consider these properties to be independent from each other. Power consumption and execution time are just a few of the many performance and non-functional properties of a system [42]. Our goal is to observe

the interactions between the system's power consumption and execution time as effectuated by different cache system parameters when executing different benchmark programs.

The contributions of this study are threefold: (1) the demonstration of a detailed manual process for power-performance tradeoff analysis using Pareto optimality and how some unexpected insights can be discovered and categorized, (2) to provide evidence that some optimal configurations might not be as expected when analyzing the live power consumption data; our test results show that the optimal configurations can be sparse, inconsistent and in many cases counterintuitive, making automated optimization processes hard to implement without analysis from actual data, and (3) to provide some useful test results of FPGA cache configurations and to demonstrate that the optimal cache configurations do exist in the selected CHStone benchmarks.

4.2 Background

In the existing literature on power/performance tradeoffs, proposed techniques target improvements over the base system without using Pareto optimality. They often fail to address the overall space of possible solutions without knowing whether their chosen solution is optimal (where they are on the Pareto front). Much of the research is conducted without the understanding of the power-performance interactions at the system level. As stated in [43], observation of a lack of Pareto optimality is an alert to an opportunity to improve the design that might be missed, especially when no single engineer understands all the design dependencies. By applying the Pareto optimality principle with all possible

solutions for the development of an efficient energy-aware system, we come up with the following hypothesis for the experiment:

- There exists a Pareto optimal curve on a solution space so that power and performance can be traded off at different weights.
- If the curve is sparse, the development of the efficient energy-aware system is difficult.

Based on our hypotheses, without a Pareto optimal analysis, it is hard to demonstrate that an improved result is optimal. It is possible that the reported result might in fact be suboptimal, far from a Pareto optimal curve. In that case, the work done could be wasted as the solutions do not encompass all the necessary elements.

4.2.1 The FPGA Cache System

To study the cache parameters of an embedded system in our experiment, we select an Atlys development board, a complete, ready-to-use digital circuit development platform based on a Xilinx Spartan-6 LX45 FPGA [37]. All of the hardware platforms configured for the experiments are based on Xilinx's MicroBlaze, a FPGA soft processor core that includes advanced architecture options like AXI or PLB interface, Memory Management Unit (MMU), Floating-Point Unit (FPU), instruction and data cache among other capabilities [46]. For MicroBlaze, the AXI System Cache soft-peripheral system used for the study is viewed as a direct-mapped L2 Cache and is highly configurable. The available cache configuration options in the Atlys board include cache size, cache line length, number of stream buffers, number of victims and write-back storage policy (all options are listed in Figure 16). Note that the MicroBlaze Cache configuration parameters are preset

with some default values and the data cache write-back storage policy is disabled by default.

4.2.2 Pareto Optimality for a Typical Power-Performance Tradeoff

As an example of power-performance tradeoff analysis, the design process we consider can be viewed as solving a bi-objective optimization problem, where we seek a cache configuration that minimizes two objectives, namely the execution time of applications in the system, and the power consumption of the system. Choices in configuring the system are generated by varying multiple cache-related parameters. In Pareto optimality, all objectives are treated equal. The "optimal" solutions found in a Pareto analysis together form the Pareto set or the Pareto front [52]. Solutions in the Pareto set reflect tradeoffs in the achievement of the different objectives. The selection of these solutions is based on the concept of dominance—a solution is worse than another only if it is so in all the objectives in the problem [36].

A scatter plot of the objective values corresponding to Pareto optimal configurations (also called a Pareto curve) can give system designers and software developers an overview of how power and performance interact in the system. It can help them design optimization algorithms for an efficient energy-aware system that can handle a wide variety of power-performance requirements. With these algorithms, an energy-aware software system can have the ability to adapt its power consumption behavior at different stages during program execution. For example, when the battery level in a system is low, applications may be forced to run at a degraded performance level in order to induce a lower power consumption rate. Algorithms could navigate possible choices on the Pareto

curve so that the performance of the applications is minimally affected even with reduced power availability.

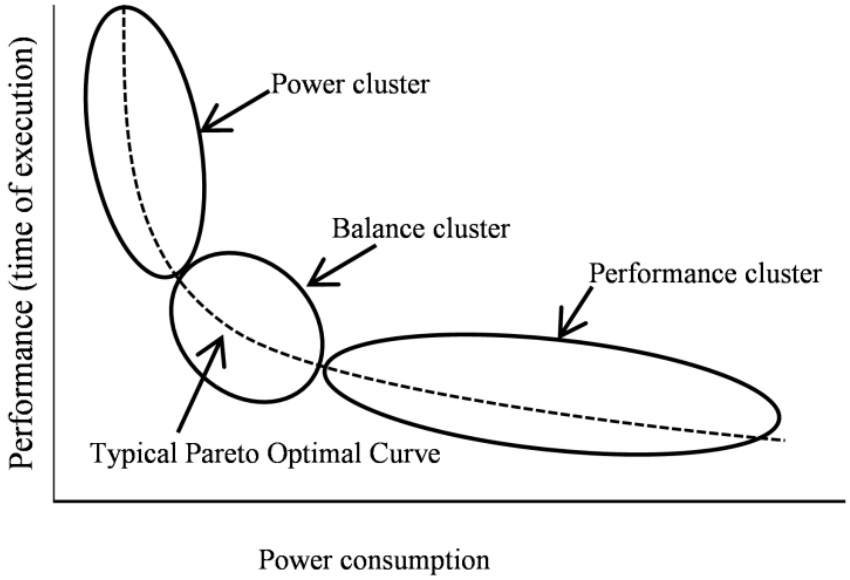


Figure 14. A Pareto Optimal Curve and Clusters for Typical Power-Performance Tradeoffs

We generally categorize the power-performance requirements of a system into three types—performance-favored, power-favored and balanced. The performance-favored type is a system that demands fast execution time over power consumption, while the power-favored type is a system that demands low power consumption over faster execution time. The balanced type is sought in a software system where both power consumption and performance are deemed equally important. Similarly, on a typical Pareto optimal curve, we can categorize the solutions into three clusters—performance, balanced and power (Figure 14). As can be seen in the figure, configurations in the power cluster allow flexibility in adjusting the performance of the applications, with no significant impact on the power consumption. While we hypothesize that a Pareto optimal curve will conform to this typical picture, the existence (or non-existence) of one or more cluster types is a

characteristic of the application(s) under test. Further, a cluster may be dense, including a large number of configurations to choose from, while another may be sparse, with a significantly fewer number of choices.

4.3 Related Work

Most related work either does not include the Pareto optimality principle or analyzes power data derived from power estimation models. For example, the research by Sahin et al. [33] focuses on high-level strategies through an approach for mapping software design to power consumption and exploring how high-level design decisions affect an application's energy usage. The results from this study show that applying design patterns can both increase and decrease the amount of energy used by an application; and design patterns within a category do not impact energy usage in similar ways. While this is just one study, the results imply that it is unlikely that impacts on the energy usage can be precisely estimated by only considering design-level artifacts. The research uses live power data but focuses on reducing energy consumption based on different software designs and does not address other non-functional properties, the tradeoff process, or include Pareto optimal analysis which are likely to be important in general.

Another interesting project related to our work is GISMOE [41]. This project sets out an alternative vision for a software development environment that can automatically generate a set of candidate program implementations, called Pareto program surface, with different non-functional attributes. At present, GISMOE is a proposed high-level architecture and set of principal features of the development environment. Although their concept is related to our research, unlike our study, their research is speculative rather than

based on empirical evidence. Their high-level abstractions of Pareto analysis might hide some unexpected insights producing an inaccurate Pareto surface. Unlike the approach proposed in GISMOE, we do not transform or change any of the benchmark program code. We instead focus on the cache system as that has been shown to have an impact on both the software performance and power consumption [48]. Although our manual process of gathering these data is tedious, the analysis results from CHStone benchmark suite yield useful information and provide a foundation toward efficient energy-aware systems and GISMOE.

There is also research related to energy efficiency and power-performance tradeoffs on the system cache [39, 45, 48]. However, the tradeoff techniques are either hardware/software specific or require additional hardware or features built in. The analyses are based on estimated power data using a power model of memory access. Their main focus is for designing optimal cache architecture and developing energy-efficient cache hardware, not for the whole system in general. In particular, the study in [45] is similar to ours but its main objective is to design power efficient cache hardware systems. The study also does not include Pareto optimality in their tradeoffs and their power data for the analysis are based on power models.

4.4 Experiment

4.4.1 Experimental Setup

To perform our experiment, we developed a custom power monitoring and profiling tool for the Xilinx Atlys FPGA board [37] using the APIs and drivers provided by the manufacturer. The board is equipped with four on-board power-supply monitors with

accuracy within 1%. The tool also annotates power consumption data with time stamps at a fine-grained resolution with an average of about one sample every two milliseconds. The monitoring tool simultaneously records power data in real-time for all four power-supply rails. In our experiments, we are primarily interested in the 1.2V and 1.8V rails which correspond to CPU and memory operations.

Table 1. Number of Pareto Optimal Solutions and Negligible Ones by Benchmark Program

Benchmark Programs	**Number of Pareto optimal solutions**	**Percentage of all cache configurations (out of 36)**	**Number of negligible Pareto optimal solutions**
ADPCM	11	30.56%	2
GSM	5	13.89%	2
MOTION	3	8.33%	1
AES	9	25.00%	4
BLOWFISH	11	30.56%	0
SHA	8	22.22%	2
DFADD	4	11.11%	3
DFDIV	4	11.11%	1
DFMUL	3	8.33%	0
DFSIN	11	30.56%	1
MIPS	4	11.11%	1
Total	**73**		**17**

The experiment uses the CHStone benchmark suite version 1.6 as standard workloads for creating power consumption profiles. The suite is designed for C-based high-level synthesis, and is easy to use since the programs are self-contained and require no external libraries [40]. The CHStone suite consists of 12 programs taken from widely-used applications in the real world from various application domains—four arithmetic

programs, four media applications, three cryptography programs, and one processor. We are able to compile and run 11 out of the 12 programs on the Atlys board (presented in Table 1 and Figure 16). For each hardware system with different cache configurations, the 11 programs were executed and profiled.

4.4.2 Experimental Method

There are three main steps in conducting the experiment—implementing the hardware platforms, profiling the software's power and performance, and analyzing the resulting data.

4.4.2.1 Implementing the Hardware Platforms

In the first step, we use the Xilinx EDK tool to design and implement each individual hardware platform with different cache parameters for the experiments. The system design and specifications of the hardware platforms are based on the Atlys Base System specifications provided by the manufacturer [38]. The only non-default parameter in each platform is the cache configuration parameter. The idea is not to build all hardware platforms for every possible combination of cache configurations but to vary only the controlled parameters from the base system, while leaving the other parameter values at the defaults. The purpose is to see how the dependent variable impacts the power consumption and performance of the base system. Based on the FPGA cache properties, we implement 36 hardware platforms, each labeled with a code—for example, ICS-64B for Instruction Cache Size of 64B (the rest of the parameters are set to the default values), DCS-512KB for Data Cache Size of 512KB, DC-WB-VIC4 for Data Cache Write-Back

Storage Policy Enabled with 4 victims, DC-LL-4W for Data Cache Line Length of 4 words (see Figure 16).

4.4.2.2 Software Power/Performance Profiling

The second step is to create a software project for each CHStone benchmark program on each hardware platform using the Xilinx SDK. With the combination of all 36 hardware platforms and 11 benchmark programs, a total of 396 software projects are built for the experiment. Since most of the benchmark programs are small and have short execution time, we modify the main programs to execute each benchmark multiple times. This provides for longer execution time so that the power monitoring tool can capture enough power data for the calculations in the next step. Also, during the data collection, all the print commands have been commented to minimize the CPU overhead caused by the commands.

4.4.2.3 Analyzing the Result Data

Power/performance profile data collected from each benchmark execution contain raw sample readings of power and execution times. The average power of each power rail is calculated as the sum of all power readings divided by the number of samples read from the start to the end of the program execution. The time of execution per iteration is calculated by the total time of execution from start to end divided by the number of times a benchmark was executed. For 396 projects, we have collected pairs of average power and time of execution per iteration for each program execution. These values are used in the Pareto optimal analysis of each benchmark described next.

4.5 Results and Discussion

With the data gathered we identify the set of Pareto optimal configurations for each benchmark. Note that the difference in power/time values may be very small across certain configurations; accordingly, some points may look identical on the plots. By using the K-Means clustering algorithm [47], the Pareto front is divided into three clusters. These clusters may not exactly signify the three cluster types as displayed in Figure 14; we manually merged one or more clusters generated by the K-Means algorithm in order to retain the underlying meaning of the three cluster types. A total of 11 such scatter plots are produced, one for each benchmark program. Figure 15 shows plots and clusters for three programs (ADPCM, AES and DFDIV) in the CHStone benchmark suite. The result data is also translated into a profile table (Figure 16) to be used in the power-performance tradeoff analysis.

4.5.1 Power-Performance Tradeoff Result

For each of the plots shown in Figure 15, the red dots (solid) represent the Pareto optimal cache configurations, while the blue points (hollow) represent suboptimal cache configurations. The groupings depict the three clusters—power, balanced and performance. Across all 11 benchmarks, we summarize the experimental result as follows:

1) All graphs reveal promising Pareto optimal curves showing that the cache is a good candidate for power-performance optimization in an efficient energy-aware embedded system development.

2) The results reveal that only a small numbers of choices of cache configurations (as low as 3 choices from the total of 36 cache configurations) are optimal for power-performance tradeoffs (see Table 1).
3) The clusters show that a small set of configurations optimally satisfies all three types of system requirements (performance-favored, power-favored and balanced).

4.5.2 Optimal Cache Configuration Result

With some post-Pareto analysis, power/execution time data from the benchmarks are put into a tradeoff profile table, as shown in Figure 16. This figure demonstrates how tradeoff analysis result data can be displayed, providing the power/performance visibility and initial guidelines for tradeoff purposes. Other information can be added depending on the requirements and selection criteria. In Figure 16, in addition to the Pareto analysis results, there are also data from using the K-Means and Normalized Distance to Ideal Point [47] methods as the post-Pareto analysis for clustering the optimal points in the scatter plots (e.g., Figure 15). As an example, our tradeoff profile table contains 36 rows representing 36 cache configurations and 11 columns for 11 benchmark programs. The table cells highlighted with red (darkest in grayscale) are the Pareto optimal cache configurations. The green (gray in grayscale) and light gray cells are the 2nd and 3rd iterations of Pareto analysis respectively. The second iteration of Pareto analysis is done after removing the Pareto optimal configurations from the design space. The 3rd iteration is done after removing the configurations found optimal in the 1st and 2nd iterations.

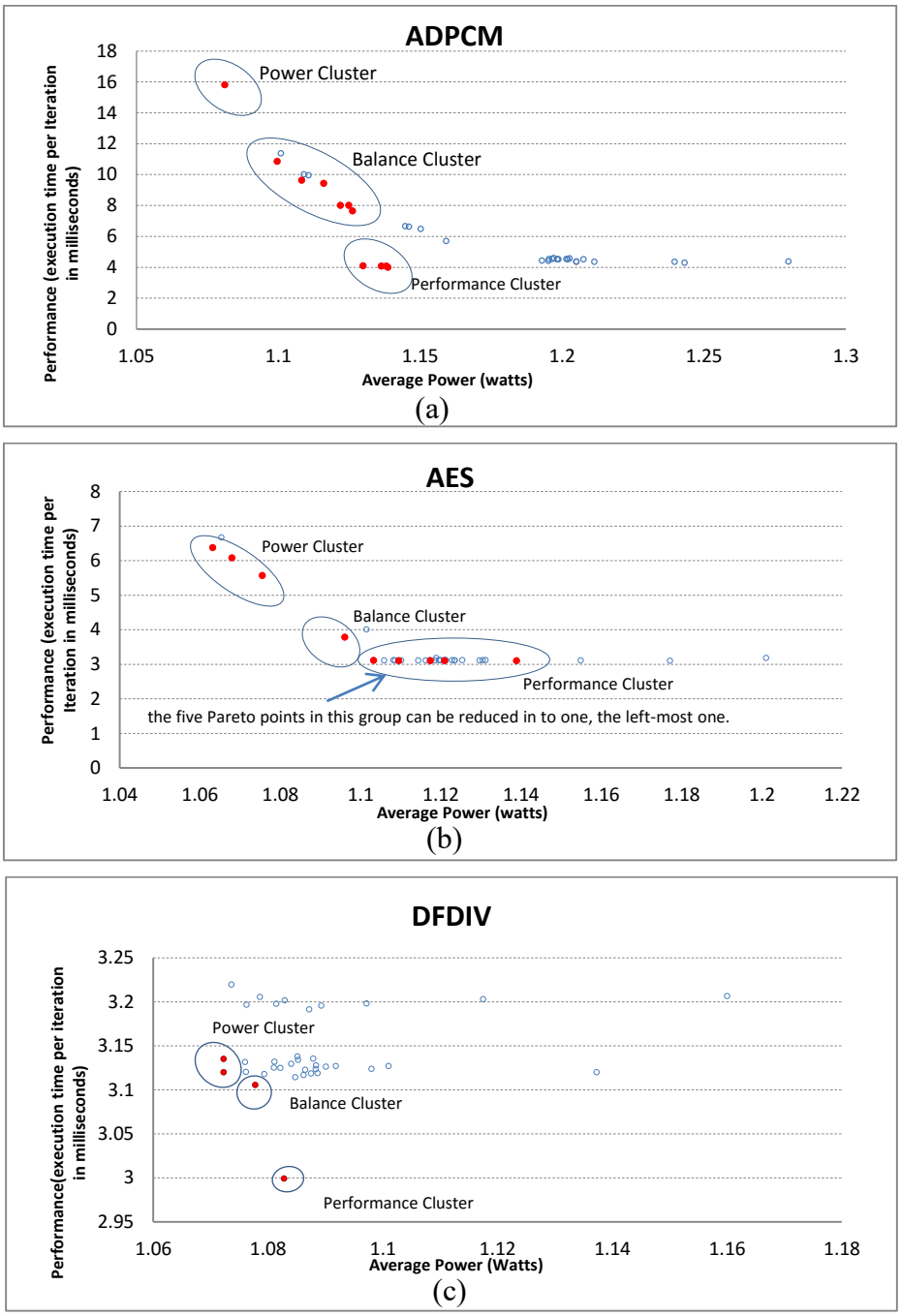


Figure 15. Examples of Pareto Optimal Cache Configurations and Clusters of Four Programs in the CHStone Benchmark

Multiple iterations of Pareto analysis is useful when the power/performance data exhibit low variability and points on the scatter plot are close together (as seen in Figure 15(a) inside the balanced cluster). Some configurations result in similar power-

performance outputs—we found that some values are equal up to the 3rd decimal place. The purpose of this analysis is to provide additional choices from the suboptimal solutions, which might be meaningful to developers or system designers. It also provides a better picture of how each cache property impacts power and performance on different benchmarks.

4.5.3 Insight Summary

Based on our observations from the experimental results, we summarize our insights as follows:

1) Only a few cache configurations are optimal for power-performance tradeoffs

All 11 benchmarks reveal only a small number of optimal (non-dominated) cache configurations from a relatively large cache configuration space. For example, as summarized in Table 1, the smallest number of optimal cache configurations are from the MOTION and DFMUL benchmarks, with only 3 Pareto points from 36. These account for about 8% of the cache configuration space that we evaluated and therefore the other 92% could have been ignored or removed from the solution space. On average, the benchmark programs produce 6.6 Pareto points or about 18% of the solution space. The number is relatively small and more than 80% of solution space can be disregarded or avoided on average. Smaller optimal choices can contribute to less complexity in the calculations of an optimization algorithm. In this case, there are fewer options for the system to switch the cache configuration parameters and adapt itself to its current state of power/performance requirements.

2) Optimal cache configurations are sparse, inconsistent and sometimes counterintuitive

The results are sparse and inconsistent because not all Pareto curves are in the expected Pareto shape. As seen in the examples, Figure 15(a), 15(b) and 15(c), some sections of the curves form vertical or horizontal lines. This means that by changing from one optimal configuration to another on the Pareto curve, there is no significant improvement on either power consumption or system performance. In this case, some optimal solutions can be ignored and one solution on each section can be selected as the representative optimal solution in the group. We call the ignored optimal solutions "negligible" because, by definition, they are insignificant or unimportant as to be not worth considering.

For example, in Figure 15(a) and 15(b) in the performance clusters, most of the optimal cache configuration points are flat and form horizontal lines on the performance axis (time of program execution). By changing from one optimal configuration to another, there is not a significant gain in performance. Therefore, only one optimal solution should be retained in this case, the one with the lowest power consumption (best solution in term of power consumption), which is the left-most one in the cluster.

In Figure 16, all decimal numbers in the cells are the Normalized Distance to the Ideal Point *(0, 0)* in normalized scatter plots—the modified version of scatter plots in Figure 15 but with both scales of execution time and power axis normalized to the scale from 0 -10 units. The ideal point *(0, 0)* signifies a system that can run a program at the fastest speed (lowest execution time) and consume the lowest possible power, both ideally

zeroes. The bold, italic and underlined numbers indicate a Pareto optimal configuration with the lowest normalized distance to the ideal point, among all optimal cache configurations for the same benchmark program. For example, in the ADPCM column, data cache with write-back policy enabled and number of victim 0 (row DC-WB-VIC0) is one of the optimal cache configurations. In addition, the bold italic and underlined notation ***CT(8.98525)*** implies that 8.98525 is also the lowest distance to the ideal point (0, 0) among the 11 optimal choices (red or darkest cells in ADPCM column)—meaning that when running program ADPCM on the system we just need to enable write back policy for the data cache and set the number of victim of the cache to 0 to get the optimal power-performance result.

In the red-highlighted cells (or darkest cells in grayscale), there are also the letters CT, CB or CP. The letters indicate clusters in the scatter plots or categories of the Pareto cache configurations. CT stands for performance cluster, CB for balance cluster and CP for power cluster. This means that, for our selected example, DC-WB-VIC0 is an optimal cache configuration categorized in the performance cluster (CT)—meaning that it is one of the optimal choices suitable for running ADPCM benchmark program in a system state where performance is more important than power, or execution time is more important than the power consumption.

		Media			Cryptography			Arithmetic				Processor
		ADPCM	GSM	MOTION	AES	BLOWFISH	SHA	DFADD	DFDIV	DFMUL	DFSIN	MIPS
I-Cache Size	ICS-64B	CP(12.10407)	CP(12.6106)	CP(13.29683)	CP(11.66424)	CP(12.034389)	CP(12.49671)	13.28304	CP(13.25121)	12.97542	CP(11.86953)	13.4829083
	ICS-128B	10.57308	CB(11.5302)	13.4135	11.9359	10.5664502	CP(11.28324)	CP(13.30432)	13.45107	12.97074	CP11.62412)	CP(13.318208
	ICS-256B	10.18704	***CB(11.3650)***	13.47432	CP(11.4419)	CB(10.505671)	9.479168		CP(13.21924)	***CB(12.8345)***	CP(11.58594)	CP(13.336081
	ICS-512B	10.18041	11.45963		CP(11.07766)	10.53867825	9.51649	13.51082	***CT(13.0144)***		CP(11.48474)	13.3917808
	ICS-1KB	CB(10.05921)	11.47495		10.14644	CB(10.450580)					CP(11.22462)	13.4487865
	ICS-2KB	CB(9.733527)	11.4697		9.73237	CB(9.4483732)				12.96558	CP(10.94869)	
	ICS-4KB	9.555242	11.4835	13.50794								
	ICS-8KB	9.50268			9.779774				13.29399			
	ICS-16KB	9.582617		13.44362	9.843192					13.0386	CT(9.766825)	
	ICS-32KB	9.629181	11.59119		CT(9.906587)						9.801238	13.3215064
	ICS-64KB	9.85685			10.19131						10.04601	
D-Cache Size	DCS-64B	CB(10.39008)		13.33218	CB(9.968029)	CB(10.018033)	CB(9.44259)	***CT(12.91722)***	CB(13.2192)	12.99141		***CT(13.271123***
	DCS-128B	CB(10.06734)	11.47932		9.670209	CB(9.8440253)	CB(9.458805		13.26656	CT(12.8473)		13.4113525
	DCS-256B	CB(9.715121)			9.692222	CB(9.7426903)	CB(9.448334		13.25611	12.98704		13.432871
	DCS-512B	CB(9.650008)	11.51831		9.696039	CB(9.6289988)	9.469237	13.31465	13.24286	CP(12.9463)		13.4231817
	DCS-1KB	9.554246				CB(9.4749122)		CB(13.15027)				
	DCS-2KB	9.463525	CT(11.4631)		9.745117	CB(9.4095792)	9.489417		13.27934			
	DCS-4KB	9.553731			CT(9.750549)				13.28821			
	DCS-8KB	9.518849	CT(11.4893)	CT(13.40506)								
	DCS-16KB	9.58346	11.5489									
	DCS-32KB	9.839793										13.404553
	DCS-64KB	10.14061										
D-Cache Write-Back Policy Enabled with different Number of Victim	DC-WB-VIC0	***CT(8.98525)***	11.46333	13.34831	***CT(9.648599)***	***CB(8.9756610)***	***CT(9.294251***	CP(13.26862)	13.41464	13.12777	***CB(9.26888)***	CB(13.277194
	DC-WB-VIC2	CT(9.031878)		13.37705	CT(9.693676)	9.000216265	9.337504		13.28224		9.31699	13.3225436
	DC-WB-VIC4	CT(9.044664)		13.40549	9.698613	9.055877747	CT(9.372639				CB(9.30665)	13.3281005
	DC-WB-VIC8	CT(9.037831)	11.47831	***CB(13.25647)***	9.689596	9.048775599	CT(9.355299)	13.32929			9.333707	13.3147547
Line Length	DC-LL-4W	9.554812										
	DC-LL-8W							13.37339				
	IC-LL-4W			13.49192	CT(9.776797)		9.508951					
	IC-LL-8W	9.555049									CT(9.62897)	
I-Cache String Buffer	IC-NUM-STR0	9.5362						13.36616				13.4715134
	IC-NUM-STR1			13.43333							CT(9.68828)	
I-Cache Number of Victim	IC-VICTIM-0	9.543207										
	IC-VICTIM-2											
	IC-VICTIM-4			13.43096				13.37133				
	IC-VICTIM-8									13.02388		

Optimal Cache Configuration Secondary Optimal Cache Configuration Tertiary Optimal Cache Configuration

Figure 16. A Power-Performance Tradeoff Profile of the Pareto Analysis Result (CT = performance cluster, CB = balance cluster and CP = power cluster; bold, italic and underlined numbers indicate the minimum normalized distance to ideal point of a benchmark)

By looking horizontally across all benchmarks in the power-performance tradeoff profile (Figure 16) to count the number of the red (or darkest) cells, and the data in Table 2, we can additionally summarize the optimal cache configuration result related to cache properties as follows:

(1) Data and instruction cache size, and write-back policy are the most influential cache properties in the power-performance tradeoff analysis. They account for most of the Pareto optimal configurations. In Figure 16, I-Cache, D-Cache size and D-Cache Write-Back rows have the highest combined number of red cells or darkest cells across all benchmark programs. Also in Table 2, the first three rows cover more than 95% of all optimal configurations. This means that when trying to configure the cache parameters for the optimal power/performance, we just look at these three cache properties as the main starting points for power-performance optimization.

(2) Data and instruction cache with larger size (2K and above), line length, instruction cache's string buffer and number of victims did not contribute significantly in the optimal achievement of a power-performance tradeoff. In Figure 16, rows of cache size 2K and larger and the last three cache configuration groups (line lengths, I-Cache string buffer and I-Cache number of victim) of the table do not have as many red cells as the previous three cache properties. Table 2 shows that the last three cache configuration groups only cover 4.1% of the optimal cache configurations. It seems that they can be ignored for most applications and are not the first candidate for power-performance optimization. This is counterintuitive.

(3) Manipulating the write-back policy on the data cache is recommended for performance-favored and balanced systems, while the instruction and data cache size are recommended for power-favored and balanced interactions. In Figure 16 and Table 2, the write-back policy row has the highest number of red cells with CT (performance cluster) letters.

Table 2. Counts of First Iteration Pareto Points by Cache Property and Cluster

	Power Cluster (CP)	**Performance Cluster (CT)**	**Balance Cluster (CB)**	**Total**
I-Cache Size	20	3	8	31(42.5%)
D-Cache Size	1	7	16	24 (32.9%)
Write-Back Policy	1	9	5	15 (20.5%)
Line Length	0	2	0	2 (2.7%)
I-Cache String Buffers	0	1	0	1(1.4%)
I-Cache # of Victims	0	0	0	0 (0.00%)
Total	**22**	**22**	**29**	**73 (100%)**

(4) Smaller instruction cache size (smaller than 2K) is recommended for power-favored optimization since this property produces the lowest power consumption with a graceful degradation in performance. In Table 2, the instruction cache size row has the highest number of optimal configurations categorized in power cluster (CP). Also, in Figure 16, rows of I-Cache size from 64 to 2K bytes cover the majority of the optimal cache configurations. While the I-Cache with larger size than 2K does not produce many optimal cache configurations.

Similarly, in Figure 15(b), the five optimal configurations in the performance cluster can be collapsed into one optimal solution; the other four can be ignored because

they do not create any significant performance improvement. In Figure 15(c) inside the power cluster, the two cache configurations on the DFDIV plot graph form a straight vertical line on the power axis. There is one additional negligible Pareto point (the top one) that can also be removed from the optimal cache configurations. All negligible Pareto points in the benchmark programs are presented in Table 1.

Our analysis also reveals unexpected insights from these irregular Pareto shapes and the negligible cache configurations. One most obvious result is revealed by AES benchmark in Figure 15(b) inside the performance cluster. Four out of the five optimal cache configurations inside the cluster are related to data cache write-back policy with different number of victims (0, 2, 4 and 8). A victim is a cache line that is evicted from the cache. If no victims are saved, all evicted lines must be read from memory again, when they are needed. Conventional wisdom states that by saving the most recent lines data can be fetched much faster, thus improving performance [46]. However, for this particular AES benchmark, our result clearly shows that not all victim policies contribute to significant improvement in performance. This evidence indicates that the conventional wisdom is not always accurate.

Additionally, in Figure 15(c), the example is not so obvious but there is a small evidence of counterintuitive insight. In the power cluster, the two optimal cache configurations are related to instruction cache size—64B and 256B respectively. In this case, changing from one I-Cache size to another does not yield significantly different power consumption. This is somewhat unexpected and contradicts studies relating cache size and system power consumption. For example, a study in [45] states that “increasing

cache line sizes tends to consume more energy" and, for I-Cache size less than 4K, the smaller size tends to consume more power.

3) *There exists cache configurations with certain properties that do not produce any Pareto optimal points*

In our cache solution space, we found that cache configurations related to I-Cache victims (the last row in Figure 16 and Table 2) do not produce any Pareto points. Also, cache configurations with I-Cache string buffer and D-Cache/I-Cache line length do not produce a significant number of Pareto points (second and third from the last row in Figure 16 and Table 2). These cache properties are therefore not candidates for power-performance optimization and can be ignored completely. Our rankings in Table 2 and colored cells in Figure 16 can help us identify the "hot spots", the best candidates for power-performance optimization and the areas having less impact on the system's performance and power consumption. From our case study, the hot spots for cache configurations are the top three in Table 2, in which they produce the most red cells in Figure 16. The table also shows that up to 50% of the total effort put into the cache based optimization can be cut (last 3 out of six cache properties in the table).

4) *There exists at least one cache configuration that can fulfill each of the typical power-performance tradeoff requirements*

As already mentioned, our analysis shows that there exist optimal cache configurations that can fulfil the three types of system requirements for a typical power-performance tradeoff—power-favored, performance-favored and balanced. As shown in Figure 15 and summarized in Table 2, the Pareto points can be clustered into performance,

power and balance clusters using the K-Means algorithm, the same way as the typical Pareto Optimal curve and clusters in Figure 14. For power-favored, performance-favored and balanced requirements, our results recommend the cache configurations related to instruction cache size (I-Cache Size), data cache size (D-Cache Size) and Write-Back policy respectively.

4.5.4 Threats to Validity

There are several validity threats to the design of this study. For threats to internal validity, our study is limited to the configurable cache system of the FPGA embedded system, one aspect of the energy-aware system. There are also other areas at different system layers having impact on the system power consumption and execution times. The cache configuration results might be different if configured differently along with other system parameters. In extending this work, we should also include other areas or combinations of different areas and parameters to obtain a broader coverage on the power-performance tradeoffs. Also, the configurable cache systems are only based on the configurable setting and property values available in the Atlys FPGA. For other embedded systems, there might be different tunable cache configurations.

For each hardware construction, we only change a single cache parameter value at a time while leaving the others the default values. Therefore, the power consumption and execution values are only based on the default cache configuration as provided by the manufacturer. This does not cover all combinations of available cache configurations. For more complete results, more hardware platforms can be built with other combinations of the available cache setting values. Also, the power consumption data of the system are only

of the CPU and Memory. There are two other power monitor rails of Video/Audio and Ethernet ports that have been ignored.

During the data collection process, because we have 396 software projects to execute, we only execute each benchmark on each hardware platform three times. Also, each benchmark program is set to execute multiple iterations for each run to make sure each benchmark executes in about 10 seconds. Therefore, the measured numbers can contain some overhead from these loops that control the executions. If we increase the number of executions, the data would be more accurate. Also, the Pareto analysis result will be more accurate if conducted on a larger solution space. Our solution space may not be large enough. The front of our Pareto optimal values is therefore considered an approximation of the "true" front. Further runs of the system with more cache configurations may improve the front approximation. Our selection of using Pareto optimality and the post-Pareto methods for the analysis is for demonstration purposes only. There are many other methods that can also be used to solve the power-performance tradeoff problems.

For threats to external validity, we try to generalize the result of our study to all software application domains. We select the CHStone benchmark suite because it covers multiple domains of real-world software applications—media, cryptography, arithmetic and processor. However, the processor domain only contains one benchmark which might not have enough coverage. And, the generalizability of the result is only for these four software domains. There is also a risk that the result might not reflect all domains of

complex software applications, since all the benchmarks are small programs and specially designed for embedded systems.

4.6 Conclusion

As a case study, our research demonstrates a detailed power-performance optimization process that can be used in developing efficient energy-aware systems. Because the process is tedious, it is crucial that developers and researchers understand the manual process and are aware that unexpected insights discovery is important and not easy to do in real systems. In the process, we also gather some basic background of how we can use Pareto optimality in power-performance tradeoffs; how power-performance requirements and the optimal solutions can be categorized; and how the data are collected, analyzed and used. Along with this, we provide some useful test results of FPGA cache configurations and demonstrate that the Pareto optimal cache configurations do exist in the CHStone benchmarks.

Our results suggest that some optimal configurations might not be as expected when analyzing the live power consumption data. We observe that the optimal configurations are sparse in the cache design space, are inconsistent across the benchmark and counterintuitive in many cases, making power-performance optimization processes hard to implement without analysis from actual data. Our results also show that even something very low-level like the cache system (that might not be captured in a power model) can impact the power-performance analysis significantly and unpredictably. These results motivate the need for tools and methodologies that operate directly on data gathered from the systems themselves.

CHAPTER 5: GREEN DATA STRUCTURE DESIGN

From this chapter forward, it is our main study of the dissertation. In this chapter, we provide in-depth details about our green data structure design approach and some of the background knowledge of our second case study.

5.1 Interface-Based Data Structure

In object-oriented programming, one common way to reduce complexity and increase maintainability, reusability and flexibility of software systems is by using interface-based design [6]. Many modern programming languages, such as Java, C#, SmallTalk and C++, implement their dynamic data structure class libraries using interface-based design where there are multiple choices of data structures with different implementations [8]—e.g. array-based, linked-list-based, hash-based and tree-based [7]. Dynamic data structures are useful for managing internal data in algorithms of software application (creating, retrieving, updating and deleting data, for example) with the flexibility to grow or shrink memory requirements based on the number of elements in the structure. Different implementations of the data structures have been shown to have different impact on the performance and memory consumption of software applications [8, 9]. Each is designed differently and is intended for different workloads and usage. Thus, it is possible that by putting the right data structure with the right workload, performance and energy consumption of software applications can be improved.

We select the C5 generic collection (version 2.3) for this study because it contains interface-based dynamic data structures and is created using a "code-to-interface" implementation [9]. It is a comprehensive, open-source C# data structure library for the .NET framework and Mono[1], with solid documentation. C5 also provides a full complement of well-known abstractions such as lists, sets, bags, dictionaries, priority queues, queues and stacks. For the case study, we explore how the choice of data structure impacts the energy consumption of software applications and what are the energy saving opportunities among these data structures. The predictive model, our proposed architecture and the GreenC5 prototype for building adaptive green data structures in this case study are also based on the C5 collection.

5.2 Data Structure Features

In order to create the predictive model for use in the dynamic selection and switching processes, our study looks at data structure features that can impact performance and limit the selection choice. Based on a data structure book by King [7], there are several features that influence the performance of dynamic data structures: number of existing elements or size, types and frequency of data structure operations. There are also features that can limit the selection choice: interface and data structure properties such as bag and set semantics. For a program or algorithm that requires data structures with a sort method, the choice can only be made on some data structure classes that implement the interface with a sort method, limiting the number of applicable data structures. Moreover, if a

[1] Mono is a software platform designed to allow developers to easily create cross platform applications.

requirement is to allow or not allow duplication of data elements, the data structures with bag or set semantic properties can also limit the selection.

Table 3. The Selected Data Structure Features

Features that can impact the performance of data structures	**Features that can limit the choice of data structures**
Number of elements in the data structure at the time of an operation (N)	Interface
Frequency of data structure operations	Bag/set semantics

Table 3 shows the selected data structure features in this study. For the features that impact the performance, we select number of elements in the data structure at the time of an operation (labeled as N), and frequency of insertion, deletion, query and update operations. We focus on the four operations because they are fundamental to a data structure. Many other operations, such as the sort operation, can be made up from combining these common operations together [7]. For features that limit the selection choice, we select interface, bag and set properties of dynamic data structures.

We also select only those C5 dynamic data structures that implement the ICollection interface in the C5 interface hierarchy (Figure 17). ICollection implements the four common data structure operations. Under this interface, we select 9 (out of 12) dynamic data structures, as listed in Figure 18, grouped by interface, set and bag semantics. We call them data structure groups, and denote them by G. Based on the selected features, there are 6 data structure groups: ICollection, ICollectionBag, ICollectionSet, IList, IListBag, and IListSet. For example, in the ICollectionBag group, there are four dynamic data structures that implement the ICollection interface and have bag semantic property. Similarly, there are two and three selection choices in the IListBag and IListSet groups,

respectively. The groupings demonstrate how selection choices can be different and vary based on application requirements. Our predictive model for energy efficiency also makes predictive data structure selection within each of these data structure groups. The WrappedArray under the ICollection interface is not included in the study because it is not a dynamic data structure.

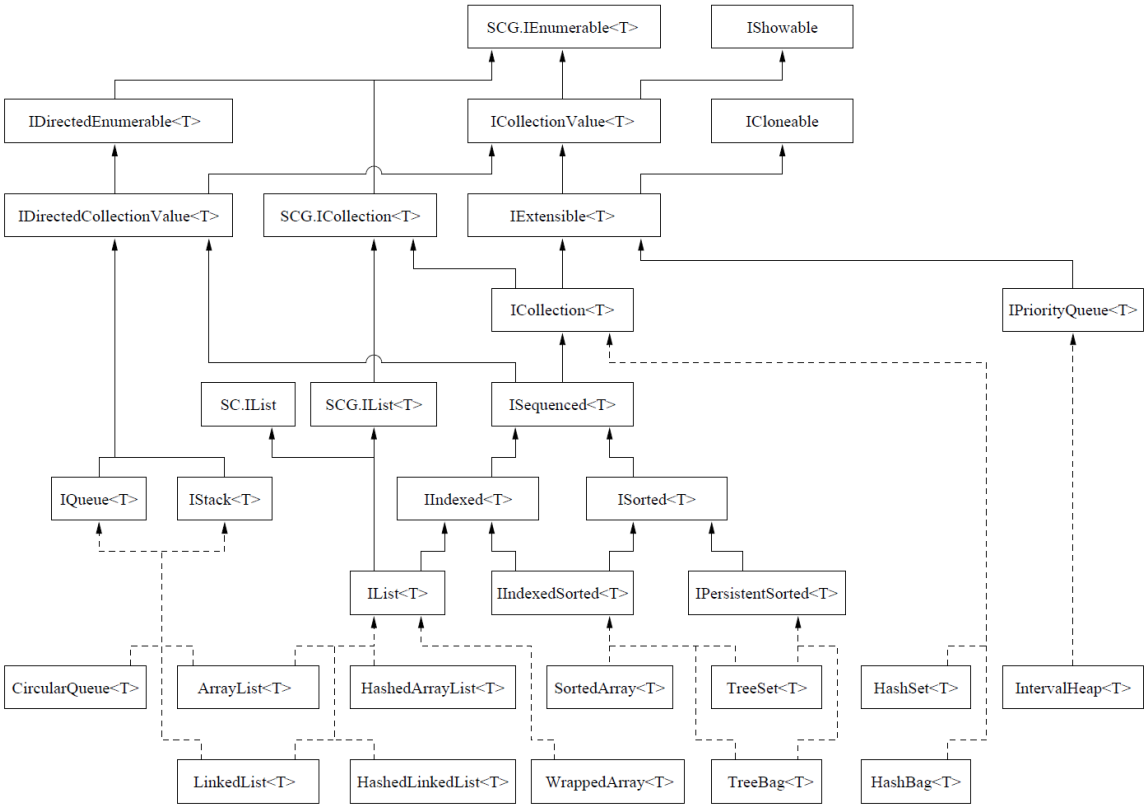


Figure 17. The C5 Collection Classes and Interfaces (from [9]). Solid Lines Indicate a Sub-Interface Relation, and Dashed Lines Indicate an Implementation Relation.

We also map the four selected data structure operations to the Create, Retrieve, Update and Delete operations (also known as CRUD operations), the four basic functions of persistent storage and database-driven applications [10]. We consider insertion and addition of elements as the same operation, analogous to the create operation. We use percentage numbers, instead of counts, as the frequency of CRUD operations. These

numbers are labeled as %C, %R, %U, and %D respectively. Since non-CRUD operations are ignored, these numbers always add up to 100. These numbers reflect the workload that a data structure is subjected to. Altogether, the group G, the size N, %C, %R, %U and %D are combined as input features to our model, and are the independent variables of our training and validation datasets.

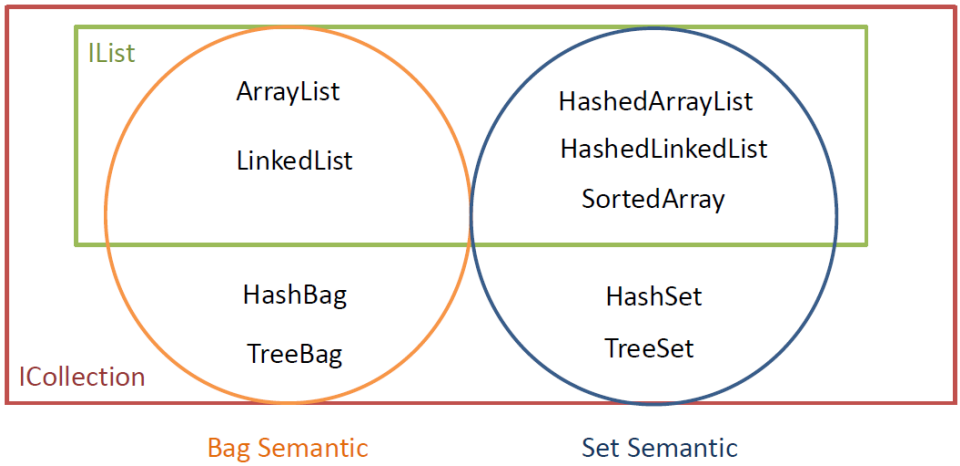


Figure 18. C5 Data Structure Groups by Interface and Set/Bag Semantics

5.3 Green Data Structure Architecture

The sought GreenC5 is an example of an adaptive green data structure. Figure 19 displays the high-level components of our proposed GreenC5. It is an enhanced version of the C5 Generic Collection that wraps the 9 data structures into one, and adds the Green component to make it smart and energy-aware. There are two main components in our proposed adaptive green data structure—the CRUD-based C5 Collection and the Green component. The first component is a wrapper/factory class of the 9 C5 data structures. It contains the public interface of the C5 data structures and has the ability to transform itself to different implementations at runtime, as directed by the Decision Maker of the Green component.

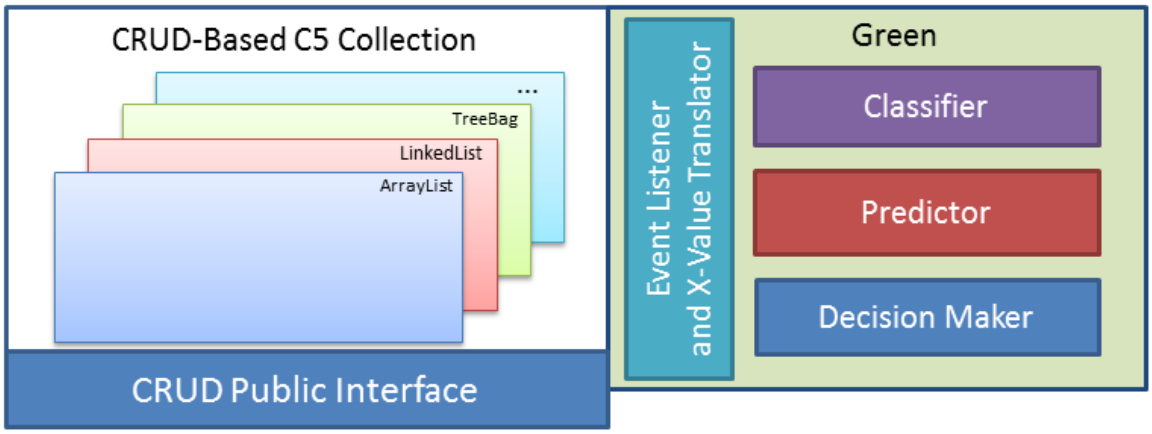


Figure 19. A Component Architecture of the GreenC5 Data Structure

The second component, the Green component, is composed of four main sub-components—Event Listener and X-Value Translator, Classifier, Predictor, and Decision Maker. These sub-components add the ability for the C5 Collection to learn workloads, classify, predict energy efficient C5 data structures, and make decisions based on the current workload, environment and requirements. The Event Listener and X-Value Translator component acts as a utility component for observing activities, operation execution and states of the CRUD-based C5 Collection component, and translating them into meaningful feature values for the Classifier. The Classifier acts as a virtual energy measurement tool inside the green object. It guesses the most-likely energy efficient C5 data structure for an observed sequence of operations, based on prior knowledge gained during training. Therefore, the output from the Classifier component is a sequence of data structures per instance of the data structure being executed in a program. This sequence is input to the Predictor component to learn (in real time) and predict the next data structure most likely to appear in the sequence. The Decision Maker then uses this prediction to analyze, decide and instruct the CRUD-based Collection when to switch to the new data

structure implementation. The Decision Maker component will be explained in more detail in Chapter 7.

Figure 20 displays a complete process of the GreenC5 data structure, depicting how the green data structure is trained and how it operates at runtime. The complete workflow in the figure can be summarized into the following sections and sub-sections:

1. **A Priori Energy Model Generation**—a one-time process for producing knowledge for the Classifier.
    a) **Energy Profiling**—for collecting energy data and creating a training dataset for the Classifier and all ground truths for model validations in the experiment.
    b) **Offline Supervised Learning**—for training and validating the Classifier.
2. **Per-Instance Green Data Structure**—for tracking how each instance of an adaptive green data structure works at runtime.

The final output from the left half of Figure 20 is a priori knowledge to be embedded in the Classifier of the GreenC5. We use an Artificial Neural Network (ANN) as the Classifier. Therefore, this knowledge contains weights and biases for the ANN edges, along with other values for data normalization, encoding and decoding purposes. In the right half, it is a per-instance GreenC5 data structure. As proposed in our previously paper [27], the CRUD-based C5 Collection and Green components no longer need to run in separate threads. In this implementation, it is simpler to have them run sequentially in the same

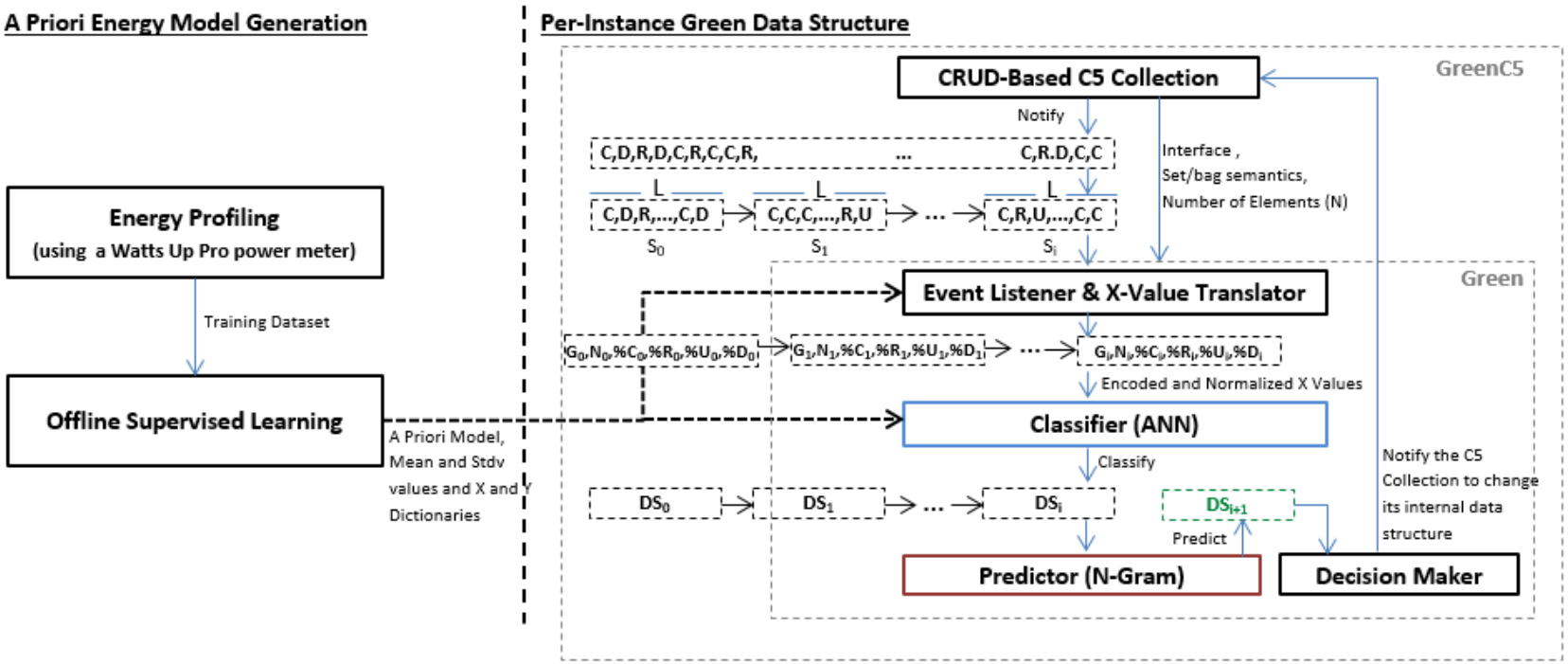


Figure 20. Adaptive GreenC5 Data Structure Process

thread. This is also to avoid adding complex code to handle data structure concurrent issue when they run asynchronously in separate threads. Most importantly, the performance penalty and overhead from running the code sequentially in the same thread is much lower than anticipated.

When a CRUD operation is performed on a GreenC5 data structure instance, the Green component continuously tracks and counts the operations, and observes other data structure states such as the interface and set/bag semantics (collectively the data structure group G), and the current number of elements in the data structure (N). The Green component sees a sequence of CRUD operations as the data structure's workload. During execution, an instance of the data structure in the program can be alive for a period of time and sometimes indefinitely if a program is always on and running. As a result, the Green component can observe a long sequence of data structure operations. In the figure, this long sequence is divided into subsequences, S_0, S_1, S_2, ..., each of length L. Together with the other observed information (G and N), each subsequence S is then input to the X-Value Translator to be translated into a meaningful feature vector for the Classifier. This feature vector is in the <G,N,%C,%R,%U,%D> format. The ANN-based Classifier maps this feature vector to a data structure DS, and outputs it. Recall that DS is one of the 9 data structures depicted in Figure 18–ArrayList, LinkedList, HashBag, TreeBag, HashedArrayList, HashedLinkedList, SortedArray, HashSet, and TreeSet. Hence, for every subsequence S_i, the Classifier outputs a data structure DS_i, effectively producing a sequence DS_0, DS_1, DS_2,..., of data structures. This sequence is fed to the Predictor in real time, where it is used for online learning and prediction of the next data structure that is

likely to appear in the sequence. The predicted data structure is input to the Decision Maker to be used for decision making to instruct the CRUD-based C5 Collection to switch if needed.

Note that we do not directly use the features (G, N and percentage of CRUD operations) to perform prediction of the data structure to use for the next L operations. Instead, we make a best guess for the data structure that would have been the most energy-efficient, and use the guesses as input in the prediction. This method allows us to discard feature vectors once they are processed, and also reduces the dimensionality of the input space for the prediction. Further, training a model that uses a history of feature vectors for prediction is not straightforward, and can easily become prone to issues related to biased sampling in the training set.

5.4 Energy Profiling

The energy profiling step is crucial to produce a dataset for training the Classifier and create ground truths for model validations. To get the most accurate model for the Classifier, in this experiment, we avoid using any model-based power meter for the energy profiling. As described in our power-performance tradeoff paper [3], model-based meters might not represent the full complexity of the system being analyzed and might introduce errors. Instead, all datasets in this experiment are created and validated using a Watts Up? Pro [16], a plug load power meter. The power meter can measure power consumption data at about 1 sample/second. It is a cost effective tool that can produce accurate power consumption data of computer systems. For this study, we have developed an energy profiler to read power data by sending a command to the device via a USB port. As part of

the GreenC5 development and overall evaluation process, we conduct the profiling on a HP PC, with Intel Core i7 and 64-bit Windows 8.1 Pro (labeled as COMP1 in the experiment). The active power consumption of the base PC ranges from about 35 to 100 watts.

```
L := 10000
start WattsUpPro asynchronously
for each N do
  for each C5DataStructure do
    for each Workload := {%C, %R, %U, %D}  do
      while PowerReadCount < = 5 do
        create an instance of C5DataStructure
        fill instance with N elements
        start Timer and capturing of power data samples
        perform C operation (%C×L)  times
        perform R operation (%R×L)  times
        perform U operation (%U×L)  times
        perform D operation (%D×L)  times
        stop Timer
       end while
       save average power/execution time data to a file
    end for
  end for
end for
stop WattsUpPro
```

Figure 21. Energy Profiling Algorithm

In the energy profiling process, energy consumption data are collected for creating the datasets of the Classifier. The exact process is detailed in energy profiling algorithm in Figure 21. For each of the 9 C5 data structures, the instance is first filled with N elements. Then the operations corresponding to each workload (%C, %R, %U and %D) are performed on the data structure instance. The data structure is profiled for energy consumption while performing the CRUD operations. During the energy profiling operations, a *WattsUpPro* instance runs asynchronously and a power reading event is raised

every second. The loops in the pseudo-code are controlled by the *WattsUpPro* instance. The inner loop breaks after the power read count reaches 5, i.e. five power values are collected. We ignore the first read sample to reduce noise due to potential delays of the USB port. The average power consumption data and execution times are measured and recorded to a file. The data are then analyzed to determine the most energy-efficient data structure (called the Y value) for each feature value set explored in the profiling. The Y values, together with the X values (independent variables), become a ground truth dataset.

There are a total of 21 datasets created by this process—one training dataset, one validation dataset, and 19 program validation datasets. The training and validation datasets contain 37,098 and 7,392 observations, respectively. The training and validation datasets uniformly explore the space of possible X values. The 19 program validation datasets are from 10 simulated programs and 9 real-world programs. The 10 simulated programs execute a random CRUD sequence of 400K size. Each CRUD sequence is equally divided into 40 10K-size subsequences. The 10-K size (also in Figure 21, the initial value of L) is selected because it is an appropriate number for our energy profiler to capture power consumption data. Note that the CRUD percentage numbers of these programs may not exist in the training dataset. The number of elements N of each subsequence is calculated based on the prior CRUD operations. Each simulated-program validation dataset contains 40 observations.

The program validation datasets of the 9 real-world programs are created from the actual CRUD operations generated by 3 real-world, open-source C# programs—A* Path Finder [17], Huffman Encoder [18] and Genetic Algorithm [19]. Each real-world program

contains at least one .NET data structure. In generating the CRUD sequences of the real-world programs, we replace the original data structures with an enhanced one that can trace and map the add, insert, access, delete and update operations to CRUD operations. The length of the generated CRUD sequences ranges from 120,000 to 1.52 million. Like the simulated programs, each such CRUD sequence is divided into 10K-size subsequences and translated into program validation datasets. The last subsequence with length less than 10K operations is ignored.

In the energy profiling process, the CRUD operations are mapped to add, find update and remove operations in the C5 data structures. In this experiment, we assume that all data structure elements are unindexed. Therefore, they are accessed by using element objects instead of indexes or keys. Also, to get the upper bound energy consumption data, the operations here are controlled so that the added element is always added to the last index. Similarly, the find and update operations always search for the last element in the dynamic data structure. However, for the remove operation, the first element is always removed. For query-related operations, all positive queries are to be performed, meaning that the accessed data element always exist in the data structure. According to King [7] and Kokholm and Sestoft [9], these operations normally result in the worst performance for dynamic data structures. It normally takes more time to search for last element than the first one in a data structure. Also, it normally takes more time to remove the first element than the last one. When the first item is removed, all other remaining items in most data structures are automatically moved up (the data structure is dynamically shrunk in size).

5.5 Energy Saving Opportunity

As a feasibility study, the energy data collected for creating the training dataset are further analyzed to see how the Y values, the most energy-efficient C5 data structures, are distributed in the training dataset and how much energy saving opportunity there is among the C5 data structures. The pie chart in Figure 22 is the analysis result displaying the distribution of the most energy-efficient C5 data structures in the 37,098 observations of the training dataset. As we can see, HashSet is the most preferable energy-efficient C5 data structure, and covers 31.44% of the training dataset. Overall, all 9 data structures are well represented in the training dataset.

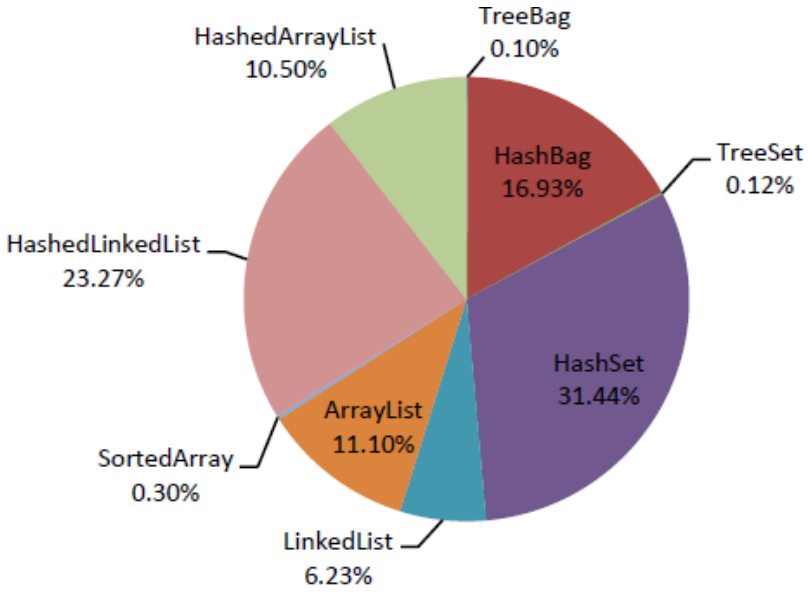


Figure 22. Distribution of Most Energy-Efficient C5 Data Structures in the Training Dataset

For more detail, Figure 23 displays an energy efficiency ranking table of the data structures by group. The higher percentage values indicate the more preferable energy efficient data structure in each group. For example, in row 5, the IListBag group has two choices of applicable data structures: LinkedList and ArrayList. ArrayList is more

preferable and should more likely be selected since it covers 63% of the workloads, while LinkedList only covers 37%. The last row of the ranking table is what is represented in the pie chart in Figure 22. This table can be used as a guide to select an energy-efficient data structure within a given data structure group. One main capability of the adaptive data structure that we seek is the automatic selection of the most energy-efficient C5 data structures without using this ranking table.

	TreeBag	HashBag	TreeSet	HashSet	LinkedList	ArrayList	SortedArray	HashedLinkedList	HashedArrayList
ICollection (6,183 obs.)	0.18%	3.44%	0.21%	92.80%	0.13%	0.86%	0.13%	0.89%	1.36%
ICollectionBag (6,183 obs.)	0.44%	98.14%			0.15%	1.28%			
ICollectionSet (6,183 obs.)			0.53%	95.83%			0.42%	1.20%	2.02%
IList (6,183 obs.)					0.13%	1.46%	0.19%	68.64%	29.58%
IListBag (6,183 obs.)					37.00%	63.00%			
IListSet (6,183 obs.)							1.04%	68.91%	30.05%
Total (37,098 obs.)	**0.10%**	**16.93%**	**0.12%**	**31.44%**	**6.23%**	**11.10%**	**0.30%**	**23.27%**	**10.50%**

Figure 23. Distribution and Ranking Table of Most Energy-Efficient C5 Data Structures by Data Structure Group

The potential energy savings are calculated using actual energy consumption data (in Joules) of each workload (observation). These numbers are of the C5 data structures that can be used to estimate and compare with the potential energy savings of our GreenC5. For each observation in the training dataset, we calculate the energy difference between the most energy-efficient and the least energy-efficient data structures, ignoring the ones in between:

Potential energy savings =

energy consumption of the worst data structure choice – energy consumption of the best data structure choice in each data structure group

The percentage difference tells us how much energy can be saved by selecting the best data structure, compared to the worst one. This analysis (based on the data points in the training dataset) shows that, overall, the median of potential energy saving about 94.24%. By data structure group, in Figure 24, the IListBag group produces the smallest energy saving opportunity of about 16.97%. This implies that if our requirement is to select only C5 data structures in the IListBag group (only two choices, ArrayList and LinkedList), the potential energy saving by selecting the right data structure is estimated at about 16.97%. In contrast, the largest potential energy saving is revealed by data structures in the ICollection group, at about 97.50%.

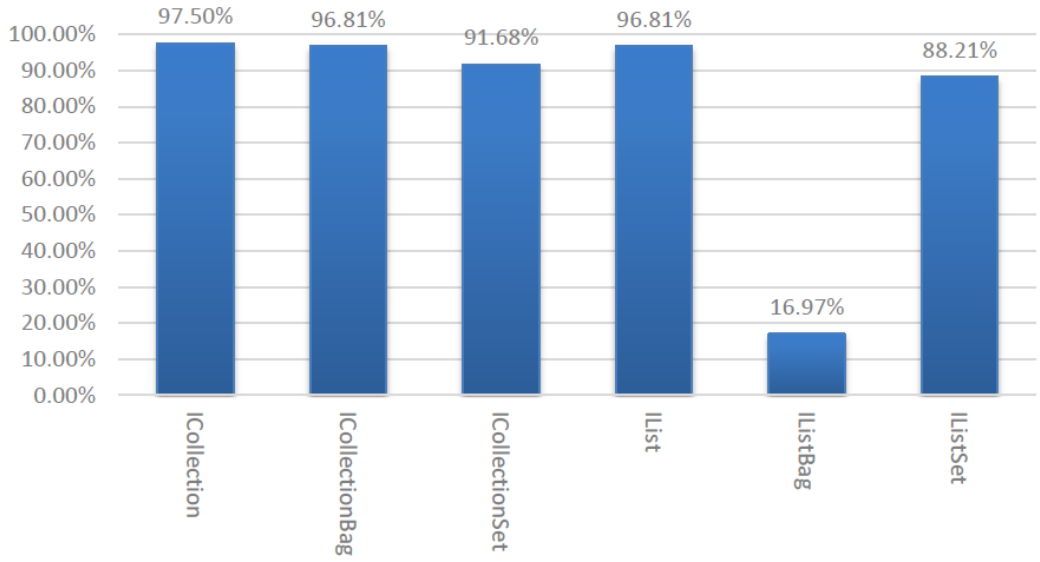


Figure 24. Potential Energy Savings of C5 Data Structures by Data Structure Group

5.6 Related Work

There are several studies related to our research. A closely related study is the Smart Data Structures project conducted by Eastep et al. [11]. Their research is aimed at creating a new class of parallel data structures that leverage online machine learning to adapt automatically. However, the approach has many differences with our study. First, the

objective there is to create a new set of data structures for parallel computing, while we attempt to apply adaptation techniques to select and transform the right data structure for the right workload from existing object-oriented data structures, mainly for energy efficiency. As such, the smart data structures [11] concentrate on using online learning and self-adaptation techniques to improve speed in order to gain the energy savings. The main technique is self-tuning of internal algorithms by changing parameters to get good performance. Instead, the knowledge for adaptation and online/offline learning of our models is based on actual energy consumption data of computer systems and done using statistical machine learning techniques.

In another related study, Daylight et al. introduce systematic, high-level, data structure transformations in the context of memory-efficient and low-power, embedded software design for dynamic multimedia applications [12]. Instead of using predictive models, the decision-making for the transformation is performed based on a Pareto tradeoff analysis between the data accesses and memory footprints. The speed and power consumption improvements are the gains from the decision results. The adaptation approach and methodology rearranges internal data structures for better memory footprint and data accesses. The power consumption data for the analysis are derived from a memory chip model, while our models are based on the energy consumption data measured by a power meter.

Our training and validation datasets are created from a custom energy profiler using a power meter to measure power consumption at the system level. However, modern processors have started embedding power measurement components in their design (e.g.

Intel Power Gadget [23]) that can potentially be used to estimate processor power consumption of software applications at a finer granularity. At the end of this dissertation, we also include an initial exploration of this avenue, primarily to search for an alternative, more practical power meter that can provide real-time energy information to the adaptive green data structure for online learning.

There are also other studies related to software adaptations for performance and/or energy efficiency. For example, the study by Flinn and Satyanarayanan [13] extends the Odyssey platform to guide mobile applications to adapt for battery life. By monitoring energy supply and demand, the platform is able to select the correct tradeoff between energy conservation and application quality. Also, research on dynamic adaptive data structures for monitoring data streams focuses on changing a specific data structure representation for accuracy, speed of response and memory requirements [14]. And lastly, the study by Ansel [15] presents a method for auto-tuning programs with algorithmic choice. The main differences with our research are that these research projects either use different adaptation or learning techniques, or are not data structure related.

CHAPTER 6: PREDICTING DATA STRUCTURES FOR ENERGY EFFICIENT COMPUTING

This chapter discusses in more detail our implementation of the two predictive components of the adaptive green data structure, the Classifier and the Predictor, along with some prediction accuracy and validation results. The implementations represent possible machine learning based solutions to the two problems highlighted in our adaptive green data structures approach—classification and sequence prediction problems. We choose to solve the problems and implement the Classifier and Predictor with artificial neural network (ANN) and n-gram, respectively.

6.1 Classifier

The problem of identifying the data structure to which a new feature vector maps to is a classification problem. We choose to solve the problem by implementing the Classifier component with an ANN. Our C# implementation of the network is based on a book by James D. McCaffrey [20]. As shown in Figure 25, our hand-tuned ANN consists of 10 input, 15 hidden, and 9 output nodes. Among the 10 input nodes, there are five nodes for the data structure group, G; one node for N; and the other four for %C, %R, %U and %D. The feature G is encoded into 5 input nodes using the 1-of-(c-1) effects encoding method, where c is the number of data structure groups. The 9 output nodes are derived from encoding the 9 C5 data structures using the dummy encoding method. The 15 hidden

nodes are the result of hand tuning that produced the highest accuracy results. The activation functions in this implementation include the hyperbolic tangent function on the input-to-hidden nodes and the SoftMax function for hidden-to-output nodes.

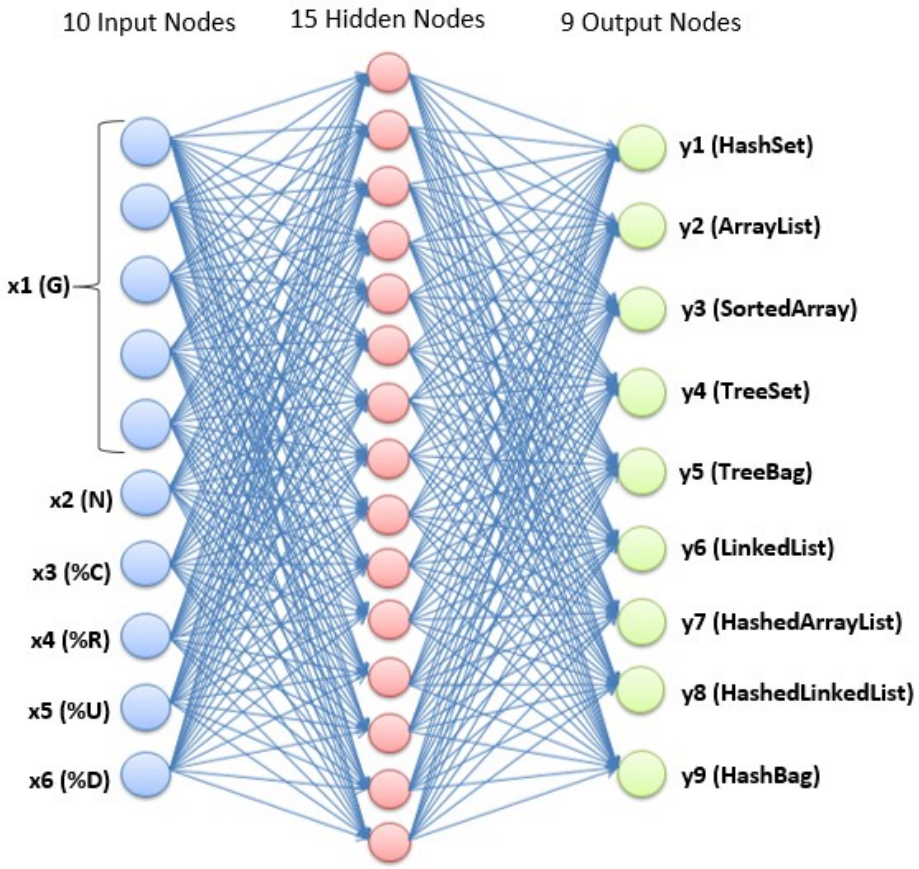


Figure 25. Our Hand-Tuned Artificial Neural Network Classifier

Offline supervised learning is used for training the Classifier. The incremental backpropagation method is the training algorithm in our implementation. The result from the training is an a priori energy model that consists of 309 weights and biases, the knowledge for the classification model (included in one of the appendices). The numbers are hardcoded in the GreenC5 data structure to be used at runtime, along with means, standard deviation values, and X and Y dictionaries—the means and standard deviation

values are used for X-value data normalization, and the X and Y dictionaries are used for translating encoded numeric values to/from the data structure group and C5 data structures respectively. The other default parameter values are also derived from the hand-tuning process—maximum epochs = 3000, learning rate = 0.02, momentum = 0.01 and stop error = 0.04. The classification method used in the implementation is the multi-layer feed-forward method. More detail and definitions of the terminology can be found in the James' book [20].

6.2 Predictor

The purpose of the Predictor is to add the ability to predict the next most-likely energy efficient data structure in the already seen data structure sequence produced by the Classifier. This is a sequence prediction problem. We solve the problem with a well-known probabilistic language model, the n-gram model [22]. This model is used for predicting the next item in a sequence in the form of a (n-1)-order Markov process. Our C# implementation of the n-gram Predictor is based on the string matching pseudo code found in the book by Ian Millington [22].

Our n-gram Predictor component uses incremental online learning to infer the distribution of data structures conditioned on observed data structure sequences, and make online predictions based on this distribution. The n-gram sequences used for learning come from the Classifier's output. The prediction is continuously made once an n-gram is registered. The output will be "Unknown" if there is not enough knowledge for the prediction, and is considered a misprediction. For example, Figure 26 shows the process of the incremental online learning and prediction of a trigram-based predictor. There are two

main methods of the Predictor component that perform the two functions, *registerSequence* and *predictNext*. Sequences of C5 data structures produced by the Classifier component are used to register to the Predictor for incremental online learning—every last seen *n* data structure sequence is input to the *registerSequence* method.

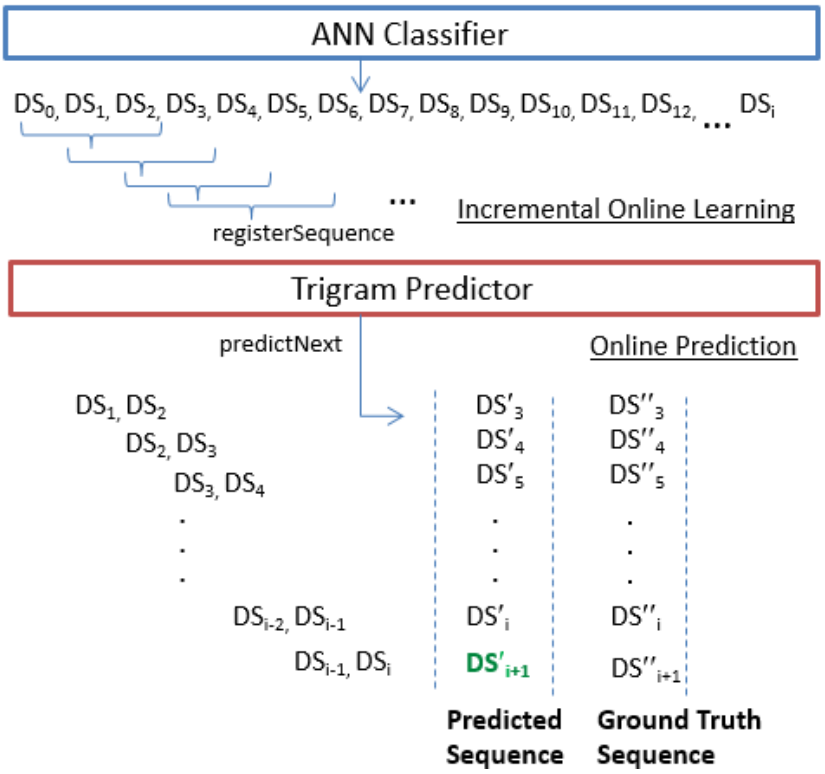


Figure 26. Incremental Online Learning and Prediction Process of a Trigram-Based Predictor

From the figure, every last three data structures, $\{DS_0, DS_1, DS_2\}$, $\{DS_1, DS_2, DS_3\}$, $\{DS_2, DS_3, DS_4\}$ and so on, are incrementally registered to the trigram predictor. After each registration, a prediction is made based on the last *n-1* registered items (last two items for a trigram or last one item for a bigram). In this example, the predictions are made based on items $\{DS_1, DS_2\}$, $\{DS_2, DS_3\}$, $\{DS_3, DS_4\}$ and so on, respectively. For the incremental online learning, the more data structure sequences are registered to the trigram, the more

knowledge and the better prediction of the Predictor. The final outcome of each program execution is a sequence of predicted most energy-efficient data structures, to be compared with the ground truths. We validate and test the accuracy of the model with the 19 validation programs.

6.3 Evaluation Results

6.3.1 Classification Results

To reduce bias and over fitting, we use a 10-fold cross validation method [21] to train the Classifier. The original training dataset is randomly partitioned into ten equal size subsamples. A single subsample, 10% of the training dataset, is retained as the test data, and the remaining 90% is used as the training data. This process is repeated 10 times, with each of the 10 subsamples used exactly once as the test data. The average accuracy result from all 10 subsamples is observed to be 95.80%. The most accurate model for classification is selected as the final a priori model to be used in the Classifier. This model is also tested with other remaining unseen datasets, the 20 remaining datasets (one validation dataset and 19 program validation datasets). We also test the accuracy with the training dataset to see how the Classifier performs on the already seen data. The accuracy result on the training dataset is 96.01%. The accuracy on the validation dataset is 82.40%, and averages 76.52% on the 19 programs (ranges from 59.71% to 99.25% accuracy). The numbers show that the neural network Classifier is adequately accurate.

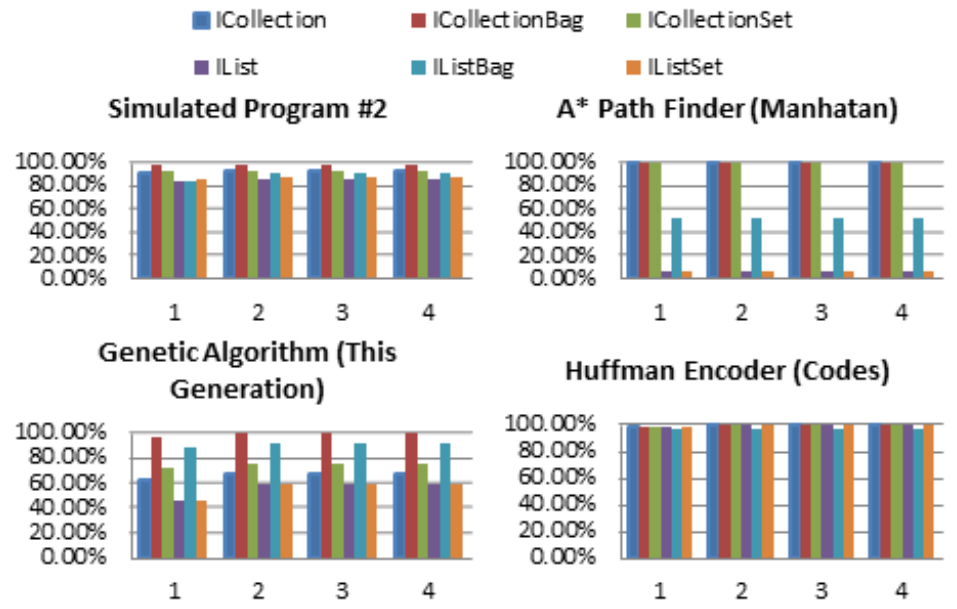
6.3.2 Prediction Result

To test the n-gram Predictor, both the Classifier and Predictor components are used. CRUD operation sequences of the 19 programs are input to the Classifier to produce data

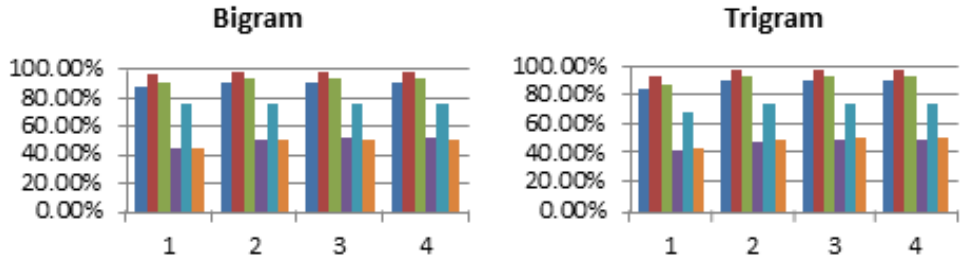
structure sequences for the Predictor to learn and predict. Each program is set to run in loops for multiple iterations simulating that real-world programs/algorithms can be executed repeatedly multiple times. The graphs in Figure 27(a) display prediction accuracy results of a trigram predictor from running 4 of the 19 simulated and real-word programs. The y-axis of the graphs represents the accuracy in percentage and the x-axis shows the first four iterations of the looped execution of a program. The accuracy numbers indicate how many data structures in each predicted sequence match with the corresponding ground truth. The graphs display the accuracy results by data structure group as indicated by the graph legend at the top of the figure. As we can see, the predictions on C5 data structures in ICollection, ICollectionBag and ICollectionSet are more accurate than that of ones in IList, IListBag and IListSet. However, there are some programs, such as the simulated program #2 and Huffman Encoder, for which the trigram Predictor produces very accurate predictions for all groups.

We also notice some low accuracy in the IList and IListSet groups when executing the A* Path Finder programs. After further investigation, we found that the inaccuracy is due to the limited size of the training dataset, and the uniform coverage therein. As a result, the workloads represented in the training dataset do not adequately sample those produced by these particular programs. We validate this by also training the Classifier with two of the five A* Path Finder programs, and then testing the Predictor with the other three remaining programs. The accuracy of the test programs in these data structure groups improves significantly, to more than 90%. Therefore, to improve the accuracy of the Predictor, it is crucial that the Classifier is trained with either more granular data points, or

with a data set that contains data points that are representative of the programs expected to execute in the system.



(a)



(b)

Figure 27. Prediction Accuracy Results: (a) Accuracy Results by Program using a Trigram Predictor and (b) Averaged Accuracy Results of All 19 Programs using Bigram and Trigram Predictors

We also conduct some misprediction analysis. One interesting result revealed in the mispredictions of a bigram Predictor is that, even if the prediction is incorrect, the energy savings from the mispredicted data structure is still more than 56%, compared to the worst data structure choice. Therefore, absolute accuracy is not always necessary to gain

advantages from the approach, especially when data structure choices are made in an uninformed manner.

Overall prediction accuracy results are displayed in the graphs in Figure 27(b). The graphs display average prediction accuracy of the bigram and trigram predictors when executing all 19 programs. For each data structure group, while incremental online learning is in progress, the accuracies of the two predictors are increasing in every iteration of program execution, and start to converge at iteration 4. The values stay unchanged after iteration 4, implying a steady state in the underlying model. Overall, the prediction for data structures in the ICollectionBag group gives the highest accuracy, converging at about 98%. On the other hand, ones in the IList and IListSet groups give the lowest prediction accuracy, primarily due to the poor performance in the A* program. For the top 4 interface groups (ICollection, ICollectionBag, ICollectionSet and IListBag), the average accuracy converges above 70%. Since the performance is not significantly different, a bigram Predictor should be sufficient. A bigram Predictor will predict the next energy efficient data structure by only looking at the current data structure in the sequence. According to Mandery [127], one of the limitations of n-grams is that as number of n grows, the memory requirements grow rapidly due to dimensionality. A lower n number is better and makes the Predictor component lightweight. With this reason, along with the adequate prediction accuracy, a bigram-based predictor is used in the implementation of our GreenC5 data structure.

CHAPTER 7: GREEN DATA STRUCTURE IMPLEMENTATION

This chapter discusses in more detail the implementation of our adaptive green data structure prototype, the GreenC5. The implementation represents a possible machine learning based data structure of an adaptive green data structure. In addition to the Classifier and Predictor, this chapter focuses more on the remaining implementations of the Green component, the Decision Maker and the data structure transformation and switching process.

7.1 GreenC5 Architecture

The GreenC5 is an enhanced version of the CRUD-based C5 Collection. It is implemented in C# and intended to be used in .NET programming. The main difference between this data structure and the original C5 data structures is its internal mechanism that makes the data structure smart, adaptive and energy-aware. The following section describes some features of our GreenC5 prototype class library.

7.2 Main Features

Main features of the GreenC5 data structure are highlighted below:

1) **Smart, adaptive and energy-aware**: the GreenC5 data structure can be trained to classify, predict and intelligently adapt for energy efficiency at runtime.

2) **Easy-to-use**: only one GreenC5 type is needed for each instantiation in a program. When fully functional, the usage is similar to the C5 data structure with few extra

installations and configurations. GreenC5 can potentially be used in multiple platforms with similar performance results without the need for calibrations.

3) **Lightweight**: the green data structure has less than 3% overhead when compared to the original C5 data structures.

7.3 Class Diagrams

Appendix A and B are the class diagrams of the CRUD-based C5 Collection and GreenC5 data structure, respectively. The diagrams detail out all properties, methods and events of the green data structure classes. In Appendix A, the CRUD-based C5 Collection class is a factory class of the 9 C5 generic data structures. The class implements the ICrudable interface so that it contains only 4 CRUD public methods of C5 data structures, while other operations are ignored and inaccessible. Using a Factory design pattern [28], the class has the ability to manufacture different data structures at runtime. The Factory pattern deals with the instantiation of objects without exposing the instantiation logic. A factory is actually a creator of objects that have a common interface. The creation of each object is done by calling the factory method rather than by calling a constructor.

The *InternalC5DataStructure* property is the actual internal data structure that holds and stores the data and is declared as *ICollection<T>* type of the C5 Collection libraries. The *CreateInstanceOfInternalC5DataStructure* method of the C5CollectionFactory class is called when a new instance of a data structure is to be created and it returns the data structure of choice to the *InternalC5DataStructure* property. The data structure transformation process takes place in this class. For every transformation and switching process, the *Copy-To* methods are called to copy the existing data of the current

data structure to a new data structure. The *TransformCompleted* event is then raised when each transformation is completed. Because the 9 data structures implement the same ICollection interface, Factory design pattern enables the class to have the ability to switch to different data structures at runtime.

Appendix B depicts our GreenC5 class library. The diagram contains three main component classes (GreenC5, Green and DecisionMaker) and three utility classes (CrudCounter, FeatureRegisteredEventArgs and TransformNotifiedEventArgs). The CrudCounter class is for counting CRUD operations. It also contains an event to be raised when the combined CRUD operation counts reach some count threshold. In this prototype, the CRUD count threshold (*CrudLengthThreshold* property) is set to a default of 10,000, the same as the length L of our CRUD subsequences in Figure 21. In this case, for every 10,000 CRUD operations, a vector set of data structure features is to be registered to the Classifier. For the other two EventArgs type classes, the FeatureRegisteredEventArgs is to hold registered feature and state information when the feature register event is raised; and TransformNotifiedEventArgs is to hold the name of a C5 data structure to be transformed to, when it notifies the GreenC5 object. The rest are some event handlers and enumerations used in the implementation. As proposed in Figure 20, the Event Listener and X-Value Translator component is not explicitly implemented here but its capabilities are actually embedded in the GreenC5, Green and CrudCounter classes.

There are four enumerations that explain some capabilities of our GreenC5 prototype—DataStructureMode, DataStructureGroup, TransformationMode and ActiveStatus. For DataStructureMode, it is an enumerated type that contains three

application run modes of the GreenC5—Static, Silent and Dynamic. When the GreenC5 is set to run in a Static mode, each instance has a static internal C5 data structure and its Green component is disabled. There is no internal mechanism running so it is considered least overhead. For a Silent mode, each Green5 instance has its Green component enabled and the internal mechanisms such as the classification, prediction, decision making, event and notification processes running normally. However, the internal data structure stays unchanged and the actual data copy and transformation processes never take place even though it is notified by the Green component. In contrary, the Dynamic mode enables the GreenC5 to run in fully operational mode where all the internal mechanisms are running normally, including the data copy and the transformation of its internal data structure. When running in this mode, the transformation and switching to different data structures can take place anytime dynamically and automatically. Therefore, the Dynamic mode has highest overhead. It is also the default mode of the GreenC5.

The DataStructureGroup enumeration represents available choices of feature G (data structure group). Users can change the values at the program level from the available choices. In the implementation, ICollection is the default value and it is set in *InterfaceAndSetBagProperty* property of the GreenC5 class. Every time the property value is changed, the values of the *CurrentDataStructure* and *InternalC5DataStructure* properties are automatically changed to a proper C5 data structure. In this case, they are automatically set to a HashSet by default (most preferable C5 data structure in ICollection group, see Figure 23). The TransformationMode enumeration contains two modes of data transformation—Immediate or WhenIdle. However, in this version, the WhenIdle mode is

not implemented. GreenC5 defaults to Immediate mode, meaning that GreenC5 immediately transforms when notified by the Decision Maker. As the result, GreenC5 is assumed to always be in Active status in this implementation.

GreenC5 is a generic data structure class that is inherited from CRUDBasedCollection generic class and implements the INotifyPropertyChanged interface. Therefore, the available public methods of data structure operations are only those of the four CRUD operations. Also, the INotifyPropertyChanged interface allows PropertyChanged events to be raised when public property values have been changed. For example, when the *RunMode* property is changed to Static or another mode, the event is raised so that the Green component is enabled or disabled automatically, and some other default values are also changed to proper values.

GreenC5 class is considered a dynamic data structure because it has a C5 data structure as the internal data storage. The class consists of two main components and are declared as private fields—the CrudCounter and Green components. However, the two components are enabled/instantiated only when the run mode is set to Silent or Dynamic. The CrudCounter tracks and counts numbers of CRUD methods being called and notifies the Green component to register a feature vector value for every 10,000 CRUD operations it observes. The Green component is the brain of the GreenC5. It is composed of an ANN-based Classifier, a bigram-based Predictor and a Decision Maker sub-component. The functionalities and implementation detail of the Classifier and the Predictor are already explained in the previous chapters. This section will explain more on the processes of the latter two sub-components—decision making and data structure transformation processes.

7.4 Data Structure Transformation and Adaptation

Self-adaptation in our GreenC5 is the process of transforming its internal data structure to different C5 data structures. The runtime process takes place in the CRUD-based C5 Collection instance, and only when it is notified by the Green component or when the value of the *InterfaceAndSetBagProperty* property of the GreenC5 object is changed. This process allows GreenC5 to adapt itself to the workload, environment and requirements, for energy efficiency. However, the overhead of each transformation process is considered high since it involves instantiating a new C5 data structure, copying over existing data to the new one and the disposal of the old one. Therefore, it is important that GreenC5 handles the transformation process properly in order to maximize the overall energy saving and avoid unnecessary overheads.

7.5 Decision Maker

The component that controls the transformation process and decides when to transform, and to which data structure, is the Decision Maker component. This component has a crucial task because a wrong decision can sometimes cause the green data structure to perform worse than the original C5 data structure choices, or the actual energy savings is not at the maximum level possible. Also, to avoid high overhead, its internal logic and codes should also be as simple as possible. In this implementation, our goal is to make the component lightweight. Therefore, having a simple decision making logic with adequate accuracy is the key of the implementation.

Our methodology for the decision making is basically to answer when it is worth to transform, and how often, so that the overall energy consumption can be minimized.

Each transformation can take place any point in time and as many times during the CRUD operations. The goal is to maximize the number of overall energy savings. Therefore, the transformations should take place only when it is necessary so that overall energy savings is at the maximum level possible. The number of transformations should be at some optimal point because the higher number of transformations the higher overhead and the lower number of transformations the higher chance of the energy saving opportunity to be missed.

One way to answer the decision making question is by looking back at how many CRUD operations have been performed since the start of the program execution or since the last data structure transformation. This is the same as counting how many feature vectors are registered to the Classifier; how many data structures have been registered to the Predictor; and how many prediction has been made. In this case, each count of the events means that 10,000 CRUD operations have been performed and one feature registration and one online prediction have been made. We use the sequence registration count (labeled as SC) as the first decision making criteria. For example, the criteria can be SC >= *k1*, where *k1* is a constant integer representing how far to look back or how many data structure has been registered to the Predictor or how many times the predictions has been made. By assigning the criteria with some *k1* values (inside the Decision Maker in Figure 28), the SC criteria can control the GreenC5 when to transform and how many times during each program execution. For example, for SC >= 5 criteria, the transformation can take place only when the predictions have been made at least five times or the CRUD operations have been performed at least 50,000 times. So, for a program with 400×10^3 operations, the maximum number of transformations that can take place is 8.

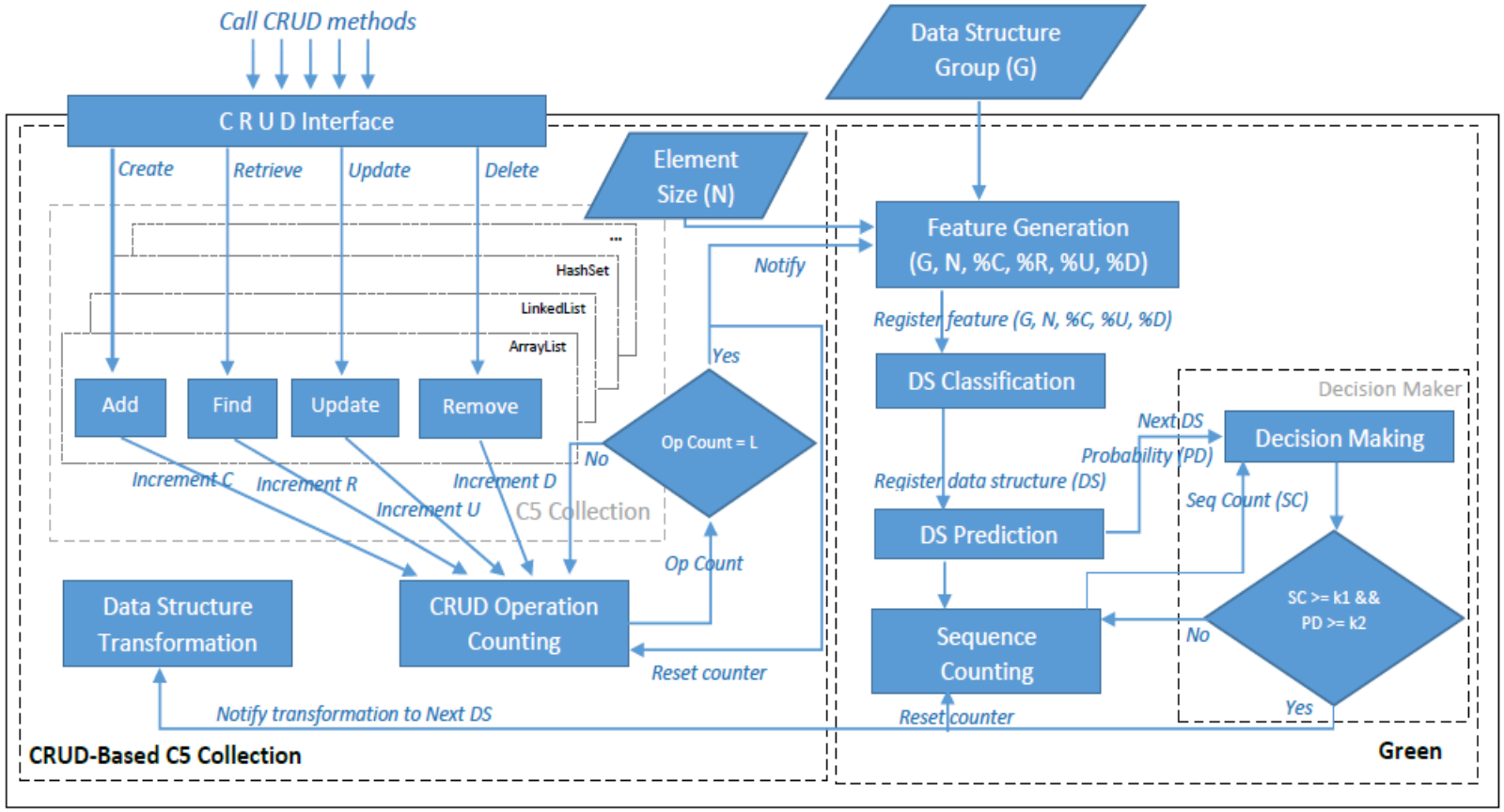


Figure 28. Flow Chart Diagram of the GreenC5

In this implementation, our first default decision making criteria is SC >= 5. The value of *k1* = 5 is derived from a hand-tuned process that, when combined with the other criteria (to be mentioned next), gives highest overall energy savings. From what we discovered, the lower numbers can sometimes result in higher number of transformations causing higher overheads and lower energy savings. The higher numbers can sometimes result in lower energy savings. If the number is too high, the transformation sometimes never take place.

Another decision making criteria is for deciding whether it is feasible or worth to switch to a different data structure. This criteria gives us some level of confidence for each decision. We look at the percentage probability value produced by the Predictor during each prediction (labeled as PD). In this implementation, the Predictor is bigram-based. The Predictor predicts the next energy-efficient data structure (DS_{i+1}) from the current data structure (DS_i). When the Predictor predicts for DS_{i+1}, the bigram-based predictor also returns the probability value associating to the predicted C5 data structure. This probability value is used in the second decision making criteria; PD >= *k2*, where *k2* is the percentage probability threshold value produced by the Predictor component when performing a prediction. For example, for PD >= 50%, the transformation can take place only when the bigram predictor produces the probability value (PD) or relative frequency value of 50% or above. Base on Jurafsky and Martin [34], the probability value created by an n-gram is called “relative frequency”. One method to estimate the probabilities is by getting counts from a data structure history, and then normalizing the counts so that they lie between 0 and 1. Our method for producing the PD values is based on the n-gram algorithm by

Millington [22]. The actual PD values of our implementation are expected to fall between *1/n* to *1*, where n is the number of the C5 data structures in each of the structure group G. However, for simplicity, our PD values are in percentage instead. In our n-gram algorithm, to compute a particular bigram probability of a DS_{i+1} given a previous data structure DS_i, we compute the count of the bigram $C(DS_i,DS_{i+1})$ and normalize by the sum of all the bigrams that share the same DS_i:

$$P(DS_{i+1} | DS_i) = \frac{C(DS_i,DS_{i+1})}{\sum_{DS} C(DS_i,DS)} \quad (3)$$

Intuitively, the higher *k2* value of PD criteria should also give us higher level of confidence for decision making. It also tells us that it is worth to transform to DS_{i+1} because the data structure has been seen most of the times in the history compared to other choices. Therefore, for higher *k2* value, it is more feasible to transform and there is higher possibility to gain more energy savings. The second decision making criteria is default to PD >= 60%. The default *k2* value of 60% is selected mainly because the smallest number of C5 data structure choices in IListBag group is two. Therefore, the probability value should be greater than 50% and should not be too high or too low. Based on some trial and error experiments, 60% is a good and reasonable number to start with. The lower the value, the higher the number of transformations that can take place.

Together, the default two decision making criteria of our GreenC5 data structure are SC>=5 and PD >= 60%. Both criteria controls how often and when the transformations can take place disregarding how many features are registered and how many data structures are predicted. Even though, these decision making criteria values (*k1*, *k2*) might not be the optimal ones, they represent a good starting point for further exploration. In this

dissertation, a full exploration of the decision criteria values is conducted as one of our additional case studies presented in chapter 8. Figure 28 displays the flow chart diagram that shows some logic, data and process flows of the GreenC5 data structure. The dashed-line boxes underneath the flow chart diagrams indicate the areas inside GreenC5 components where each process takes place.

7.6 Usage of the GreenC5 Data Structure

Ultimately, the GreenC5 data structure should preserve all properties and capabilities of the original C5 data structures including the usage. It should be easy-to-use and require minimum learning curve, configurations or extra installations. The followings are some examples of the usage of our GreenC5. There are at least three important class libraries to be installed in the target machine and included on the top of each program/project:

```
using C5;
using CrudBasedCollection;
using GreenCrudBasedCollection;
```

To create an instance of a GreenC5 data structure, programmers can instantiate it the same way as instantiating any of the C5 generic data structures. The below lines of code is an example of creating a new GreenC5 instance object to store string objects. By default, the data structure will run in a Dynamic mode unless specified by the programs. In Dynamic mode, the data structure automatically and dynamically transforms itself to different C5 data structures in the ICollection group for better energy efficiency. The code example demonstrates how to add 10,000 string objects to a GreenC5 instance and then update, retrieve, print and delete the strings. As already tested, the code section results in less energy consumption than using a LinkedList (the least preferable C5 data structure in

ICollection group). This example forces the GreenC5 instance to start as a LinkedList. However, by the end of the code execution, the internal data structure of the GreenC5 has automatically and correctly been transformed to a HashSet and consume less energy than the LinkedList and other C5 data structures.

```
GreenC5<string> ds = new GreenC5<string>();
ds.CurrentDataStructure = C5DataStructure.LinkedList;
//force it to start as a LinkedList
ds.CreateNewInstanceOfInternalC5DataStructure(C5DataStructure.LinkedList);

for (int i = 0; i < 10000; i++)//Create
{
  ds.Create("Hello" + i);
}
for (int i = 0; i < 10000; i++)//Update
{
  ds.Update("Hello" + i);
}
for (int i = 0; i < 10000; i++)//Retrieve
{
  ds.Retrieve("Hello" + i);
}
for (int i = 0; i < 10000; i++)//Print
{
  Console.WriteLine(ds.Retrieve("Hello" + i));
}
for (int i = 0; i < 10000; i++)//Delete
{
  ds.Delete("Hello" + i);
}
```

7.7 Code Release

The GreenC5 project is set to be an open-source project. Therefore, some fundamental projects and the source code have been released to the public via a GitHub website [141]. The main goal is for research purposes and to provide some power metering tools for green software development projects and for future development of the GreenC5. All released projects are in C# language and combined in a single project called DUGreen project. The released projects include:

1. **Watts Up? Framework:** this framework contains APIs for power measurement using a Watts Up? Pro meter.
2. **Intel Power Gadget Framework:** this framework contains APIs for power measurement using the Intel Power Gadget with other required libraries.
3. **CRUD-Based Collection Energy Profiling:** this project contains a program for creating training and program validation datasets using a Watts Up? Pro power meter.
4. **Machine Learning Framework:** this project contains two implementations of an Artificial Neural Network and N-Gram.
5. **CRUD-Based C5 Collection:** this project contains a wrapper/factory class of the C5 data structures.
6. **Green CRUD-Based C5 Collection:** this project contains the GreenC5 data structure class, Green component and other utility classes.
7. **GreenC5 Simulator:** this project is the implementation of our GreenC5 simulator to allow users to interact with the GreenC5 in different use-case scenarios and settings. The project also contains code examples of how the GreenC5 is used and the implementations of Watts Up? and Intel Power Gadget power profilers.

CHAPTER 8: GREEN DATA STRUCTURE EVALUATIONS

This chapter explains the evaluation of the GreenC5 data structure along with the results and analysis. It also includes several additional case studies to further evaluate the green data structure.

8.1 Experimental Setup

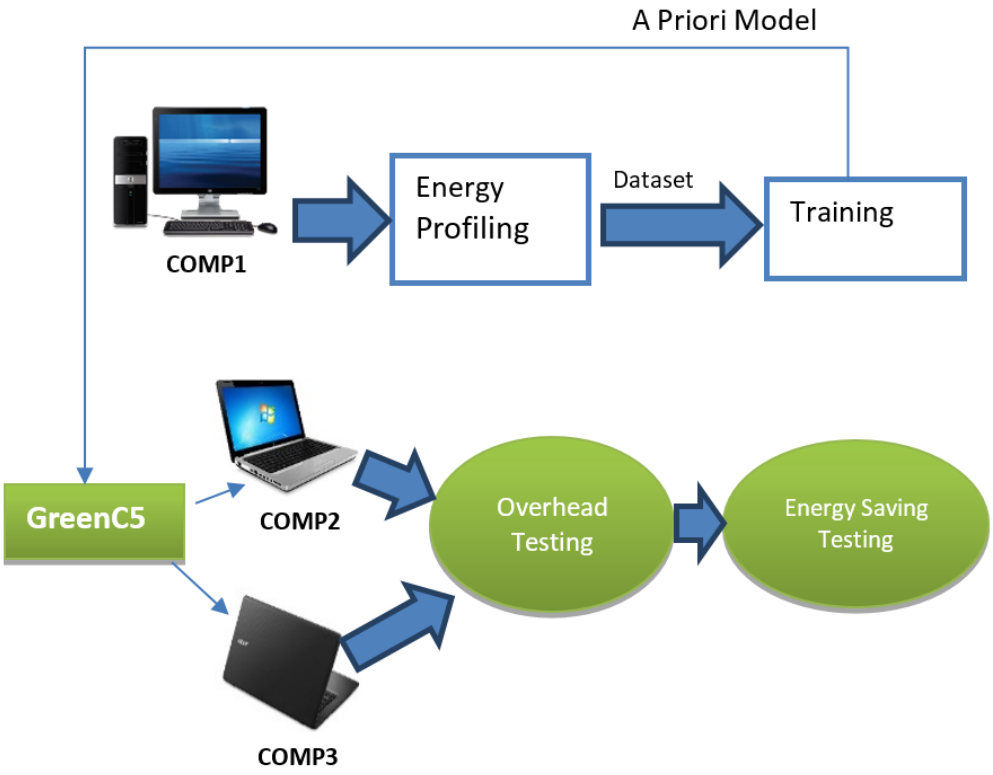


Figure 29. GreenC5 Evaluation Process

Figure 29 shows the overall process to evaluate the performance and overhead and to calculate the estimated and actual potential energy savings of GreenC5. There are three computers used in this study, labeled as COMP1-3 with different specifications, based

power consumption and versions of Windows operating system. List of the computers is displayed in Table 4. COMP1 is used to perform the energy profiling, training and creating the a priori energy model, as already explained in previous chapters. The other two machines are used to test GreenC5 data structure embedded with the same a priori model created by COMP1.

Table 4. List of Test Computers with Specifications and Based Power Consumption

Computers	**Specifications**	**Base Power Consumption (Watts)**
COMP1	HP Envy PC, model h8-1520t with Intel Core i7 CPU@3.4 GHz 10GB RAM and Windows 8.1 Pro	35
COMP2	HP Probook 4720s with Intel Core i7, CPU@2.67 GHz, 8GB RAM and Windows 7 Pro	30
COMP3	Acer Aspire R14 with 6th-Generation Intel Core i5 CP-6200U, 8GB DDR3 RAM, 256GB SSD Drive and Windows 10 Home edition	9

The purpose of the experiment is to investigate whether our a priori model created during the energy profiling and offline learning processes in the first machine can be used in other machines; and whether they produce similar results. We want to see if the model can potentially be universal where only one single model is needed and can be used in many other machines. The purpose is to avoid a repeated machine calibration process when using GreenC5. In real world, the number of machines in the experiment can be increased for better results. However, in our study, the number is limited to 3 machines. Also, to improve accuracy in this evaluation process, the total number of observations of the training dataset is expanded to 38,826. This dataset includes observations of the original training dataset and the ones from two of the five A* Path Finder programs. Therefore, the

remaining 17 of the 19 validation programs are used to evaluate GreenC5. The following evaluation results are based on the 17 validation programs.

8.2 GreenC5 Evaluation Results

According to the experimental setup in Figure 29, there are two tests needed to validate the GreenC5 prototype—overhead and energy saving tests. This section explains the preliminary results showing that GreenC5 can help minimize the energy consumption of the base systems.

8.2.1 Overhead Testing Results

When enhancing application objects and components, more code is normally added to the programs. More complex code normally means higher overhead that can sometimes undesirable. This experiment is to see how much overhead in term of energy consumption is added to GreenC5 when compared with the original C5 data structures. The method is straight forward. We run 17 validation programs on each of the two test computers (COMP2 and COMP3), by inputting CRUD sequences of the validation programs to each of the 9 C5 data structures. The energy consumptions from running each of the programs are captured and saved by the energy profiler for analysis.

The same process is also done on the GreenC5 data structure in a Silent mode. In this mode, the GreenC5's internal data structure is set statically to each of the C5 data structures. The Silent mode activates all internal mechanisms and enables the Green component, but no actual transformation takes place. When a GreenC5 is running in a Silent Mode, it acts like a C5 data structure running with an active Green component attached to it. By comparing the energy consumption of the original C5 data structures

with the GreenC5 running in a Silent mode, the energy differences can tell us how much additional energy consumption caused by the Green component. The following overhead results are based on data collected from executing each validation program for two runs on each computer. Both computers are executed in a controlled environment, where some system processes of Windows operating systems such as Wifi and Internet connection are turned off to minimize noises. Note that we do not include the adaptation and transformation processes in this overhead testing because the cost is high and vary depending on the types of C5 data structures. Instead, these processes are managed by the Decision Maker component to maximize system's energy efficiency.

The overhead results are quite surprising. We expect to see high overhead in term of energy consumption since the added code of the green component is rather complex. Instead, the average energy differences are quite small. For most programs, GreenC5 consumes more energy than the original C5 data structures. Only few experiments show that GreenC5 consumes less energy than the original ones. The percentage energy differences range from small decimal numbers for most program executions and up to the twenties for few program executions. This is due to the fluctuations of the system power consumption. However, on average, the energy differences or overhead of GreenC5 for all programs in both computers are less than 3% (2.62% and 2.14% for COMP2 and COMP3 respectively). Since, the numbers are small, we consider the GreenC5 data structure as lightweight.

8.2.2 Potential Energy Saving Results

Next test is to compare the energy consumption of programs running the least preferable C5 data structures of each data structure group with ones running the GreenC5 data structure in Dynamic mode. Our goal is to see how well the green data structure can adapt for energy efficiency and what are the actual potential energy savings from dynamically transforming its internal data structure to different C5 data structures at runtime.

To achieve the goal, the energy consumption data of the C5 data structures are used as the base to compare with that of GreenC5 running in Dynamic mode. For each of the program, we seek the best and worst C5 data structures in each of the data structure groups when running each of the validation programs (Min and Max lines in Figure 30). The energy consumptions of the two data structures are set as the lower and upper bounds of each group. Next, we execute the programs with the GreenC5 data structure in Dynamic mode with different values of *InterfaceAndSetBagProperty* property (data structure group G), simulating different interface and set/bag semantic requirements. We start each execution by setting the initial internal data structure of the GreenC5 to the worst choice in each group, found in Figure 23. For example, the GreenC5 data structure is set to start with a LinkedList for ICollection, ICollectionBag, IList and IListBag groups. The CRUD workload of each program is then input to the green data structure. If the GreenC5 adapts correctly, the energy consumption of the GreenC5 is expected to fall in between the Min and Max points in each data structure group and by the end of each execution, the internal data structure of the GreenC5 should be transformed to a better energy-efficient C5 data

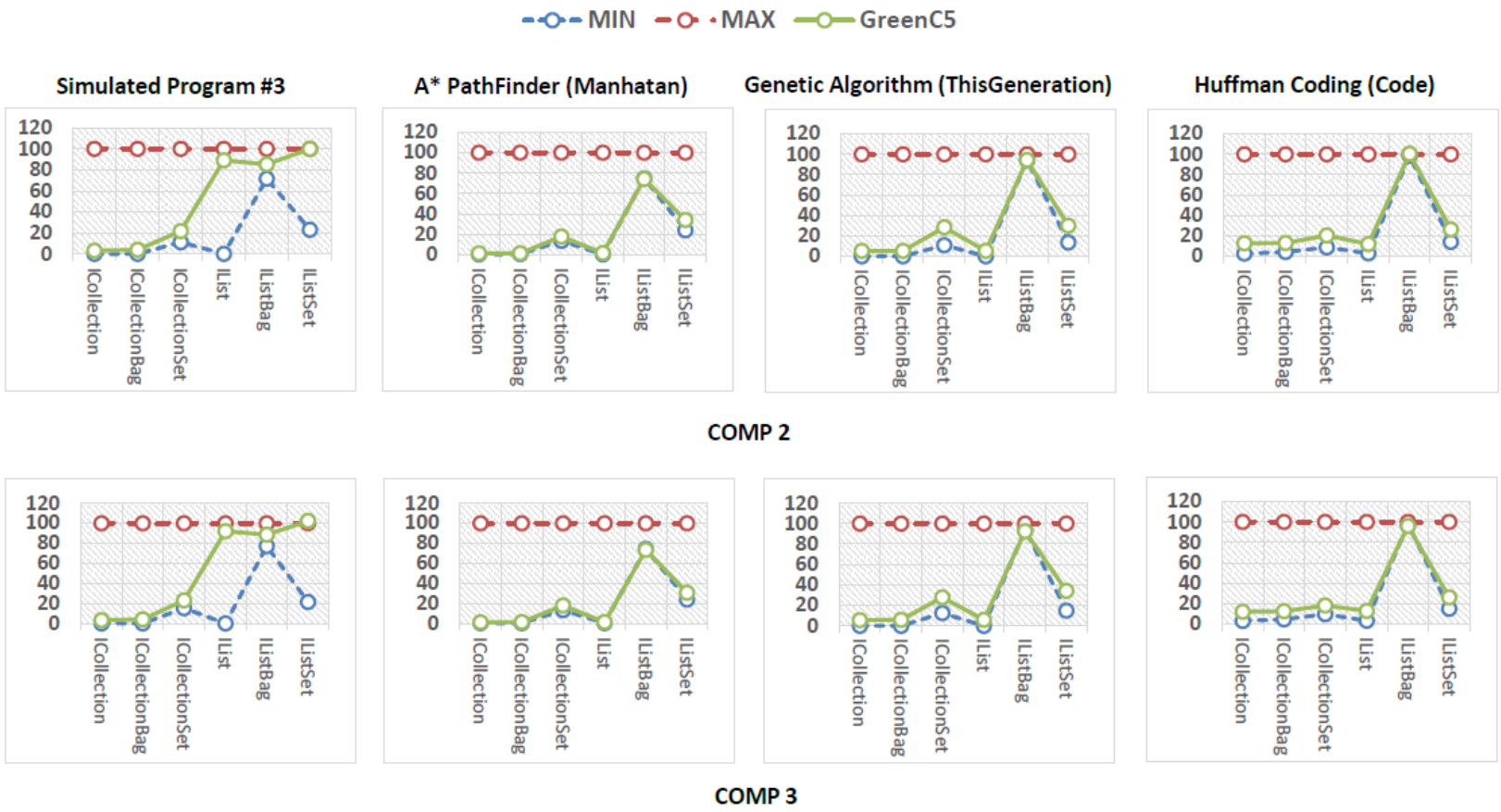


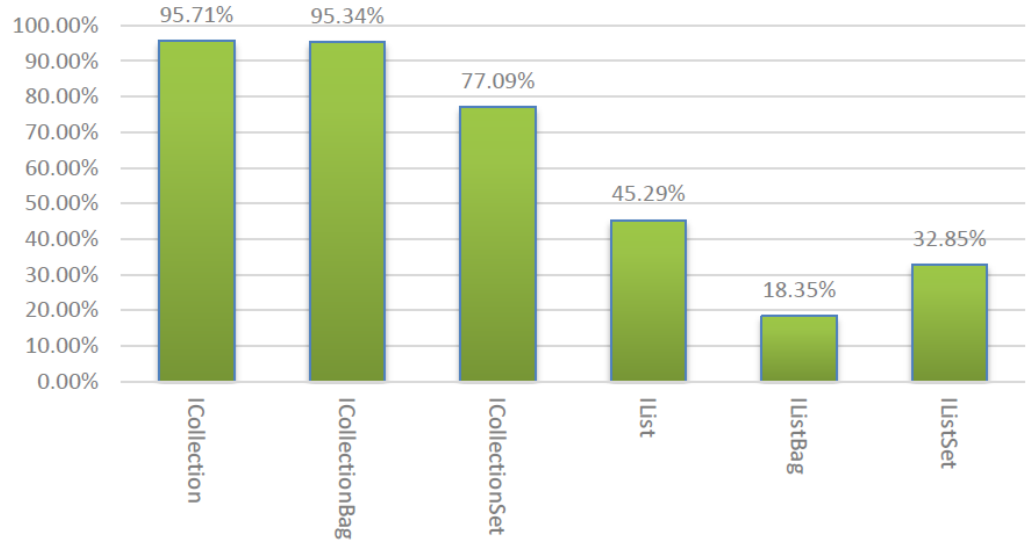
Figure 30. GreenC5's Potential Energy Savings by Data Structure Group by Program

structure in each group. Figure 30 shows some experimental results of the potential energy savings of GreenC5 by data structure group per program conducted in COMP2 and COMP3.

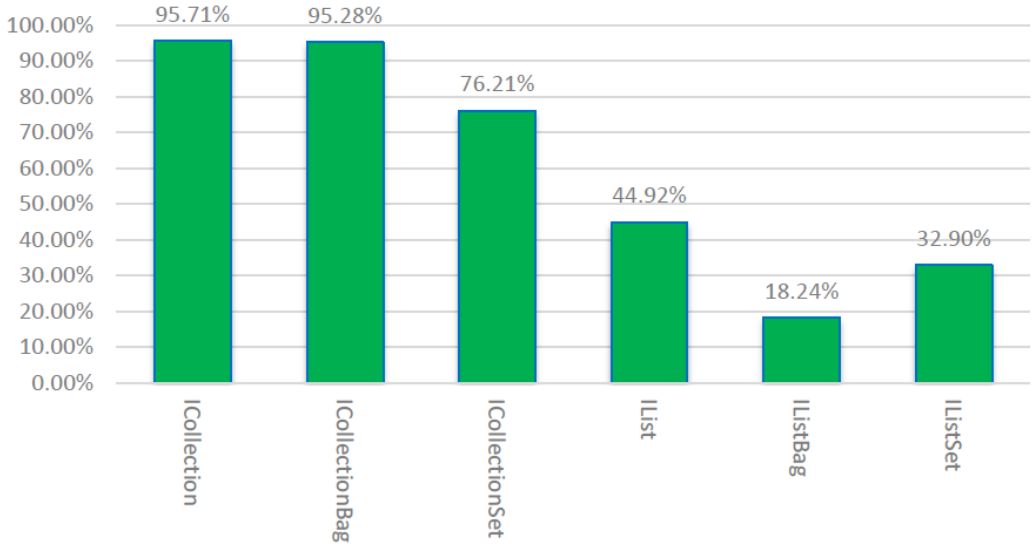
In Figure 30, the energy consumption data of both computers are normalized so that the Max points or upper bound are always at 100 (Y axis). The dots on Min and Max dash lines represent the maximum and minimum energy consumption points of C5 data structures by data structure group per program, respectively. The dots on GreenC5 lines represent the energy consumption of Dynamic-mode GreenC5. The closer to the bottom lines the better energy efficiency and the higher potential energy saving. The energy differences between the Min and Max points are called expected energy savings. While, the actual energy savings of the GreenC5 are the energy differences between the GreenC5 and the Max points. As you can see, the GreenC5 points lie between the top and bottom lines as expected. Only few that lies above the top or below the bottom lines (in the IList group of Simulated Program #3, for example).

Surprisingly, GreenC5 performs quite well as you can see that most of its energy consumptions are close to or almost lie on top of the Min lines and mostly in between the Min and Max lines. This means that our Predictor can predict accurately and the Decision Maker can notify GreenC5 to transform and adapt correctly with minimum overhead. Moreover, the results from both computers are almost identical even though their base power consumption and system specifications are significantly different. This is another unexpected result. This means that our a priori model can potentially be a universal model that can be used in multiple computer platforms. As a result, only one model is needed for

GreenC5 and there is no need for system calibration, making GreenC5 even more user friendly.



COMP2



COMP3

Figure 31. Average Potential Energy Savings of GreenC5 by Data Structure Group

For the overall picture, Figure 31 shows the graphs of average actual potential energy savings of GreenC5 in the 17 validation programs on COMP2 and COMP3, respectively. The numbers indicate the percentage energy savings, representing how much energy can actually be saved from using GreenC5, by data structure group. Again the two graphs are almost identical and are in the ranges as estimated during the energy profiling process (Figure 24). GreenC5 performs very well in the ICollection and ICollectionBag groups—more than 95% potential energy savings when compare to the worst C5 data structure choices. IListBag group produces the least potential energy savings (little above 18%). For all data structure groups, the median potential energy savings are 61.19% and 60.56% for COMP2 and COMP3 respectively. The numbers show that GreenC5 can adapt for energy efficiency and can potentially help the computer systems consume less energy.

8.3 Threats to Validity

There are many other aspects that need to be considered in order to develop a fully functional adaptive green data structure. This section explains several validity threats to the design of our study and the implementation of the GreenC5 prototype. For the creation of a priori knowledge of our green data structure, the model is based on C5 dynamic data structures that implement the ICollection interface, and is based only on the selected data structure features. Because the training dataset does not cover all possible workloads, our predictive model has some limitations. First, the predictive model is limited to common operations that map to the CRUD operations. Other operations are ignored. Second, the feature N is also limited to 50K elements and the length of CRUD subsequences is fixed at 10K. The workloads, %C, %R, %U and %D, are also limited at some level of granularity.

We do not consider whether the CRUD workloads in the training dataset are feasible. For example, we include a workload with 90% of delete operations and 10% of add operations; such a scenario will never realize. Also, our models (ANN and bigram) are selected mainly because they can be used to solve energy-efficient data structure classification and sequence prediction problems and the source codes and algorithms for developing the models are available. Some of them are hand-tuned without full exploration of all possible solutions. They are not claimed to be the best solutions for these problems. Other machine learning methods such as Recurrent Neural Network (RNN), SVM, Logistic Regression and HMM are among the possible candidates for solving the energy-efficient data structure classification problems.

The energy profiling is a key process in creating the a priori knowledge for our predictive model. The energy data collection process is done with a power meter that can read power consumption at 1 sample per second. The result can be more accurate if we can measure power consumption data at a finer grain and on different parts of the system, such as CPU, memory, display, etc. Our power measurement is at the system level, and may have overhead and noise. The energy collection process is designed to collect energy data on a fixed setting as described in the algorithm presented in Figure 21. We claim that the priori knowledge is potentially universal. Our claim is based on only two Windows machines and mainly because the results from both computers are almost identical. More tests can be done on more programs and computers to make the claim more legitimate.

Lastly, the version of our GreenC5 data structure is mainly for research purpose and is implemented as a prototype of an adaptive green data structure only. It is not a fully

functional green data structure that can replace the C5 data structures in existing programs. There is much more work to be done to make it fully functional. The internal mechanism and logic of the Green component, the decision maker and transformation process are as described in Figure 28. The energy saving results are based on the fixed decision making criteria values (SC >=5 and PD >=60%). Even though, an exploration for better decision making criteria values is conducted in one of the case studies, the better optimal criteria values produced from the experiment are not used in any of the mentioned experiments. Also, the potential energy savings are evaluated from the 17 program workloads created from 3 real-world and 8 simulated programs. More workloads from more programs are also needed and the testing should be conducted more extensively on more platforms for better results.

8.4 Additional Analysis and GreenC5 Simulator Implementation

The purpose of the additional analysis is to further evaluate the GreenC5 prototype in different use-case scenarios and to answer some of the additional research questions that are considered threats to the validity of our experiment. The following questions are to be answered in the case studies:

1) What are the optimal values for the decision making criteria in the decision maker (values of k_1 and k_2)?
2) Are there any other alternative power measurement tools that can be used in GreenC5, possibly for reinforcement and online learning in the future work?
3) What is the energy efficiency improvement in other use-case scenarios? For example, how the GreenC5 affects the system energy consumption when

multiple instances are being executed sequentially in a single thread and asynchronously in separate threads/programs?

8.4.1 Additional Analysis #1: Alternative Power Profiling Tool and GreenC5 Simulator Implementation

To answer some of the additional research questions related to our GreenC5 prototype, we develop a GreenC5 and demonstrate how GreenC5 can be used in different use-case scenarios and settings, and how it can help software applications to save energy. The simulator also integrates both Watts Up? and Intel Power Gadget power monitoring tools for comparison. Intel Power Gadget is selected as an alternative power profiling tool because it can potentially be used for online learning of GreenC5 in our future research projects. This section explains features and the implementation of the GreenC5 simulator and the integration of the Intel Power Gadget with the simulator.

8.4.1.1 GreenC5 Simulator Implementation

The main goal of the GreenC5 simulator is to allow users to interact with the GreenC5 data structure. The simulator is implemented in C# and is intended to run on Windows machines with 2nd or later generation of Intel Core processors. It utilizes the C5 collection and the GreenC5, Watts Up? and Intel Power Gadget class libraries. Using the graphical user interface of the simulator, users can add GreenC5 instances to application threads, change the settings, execute the program in different use-case scenarios and see how the program is performing and how much energy is being consumed, interactively. Users can set GreenC5 to run in static mode, silent or dynamic mode. Users can also select a program from the 19 validation programs and set the decision making criteria values for

the GreenC5 instances. The real-time power consumption data from both Watts Up? and Intel Power Gadget power are displayed side-by-side for comparison. The simulator is designed to be used for one of experiments in the additional analysis. The results will be explained in the following sections.

Figure 32 is a screenshot of our GreenC5 simulator. The screenshot contains three main areas of the simulator application. In the first area, the top part of the window, are the live power consumption graphs (in watts) of the two power monitors. The second area, the bottom left part, is the simulated application area; users can add application threads and GreenC5 instances to each thread and set the GreenC5 setting properties, simulating different application use-case scenarios. The last area, the bottom right, is the application information and setting area. The area contains settings of the simulated application and the decision making criteria of the added GreenC5 instances, and the user controls for controlling the simulator. It also displays live power number readings, live energy consumption (in joules and watt hours) and productivity (number of executions per joule) of the application simulation. At the top of this area, a live number of normalized correlation of the two graphs is also displayed. The number indicates how much the two graphs are correlated. This number can also tell us whether Intel Power Gadget tool can be used as an alternative power monitoring tool for the future GreenC5 projects.

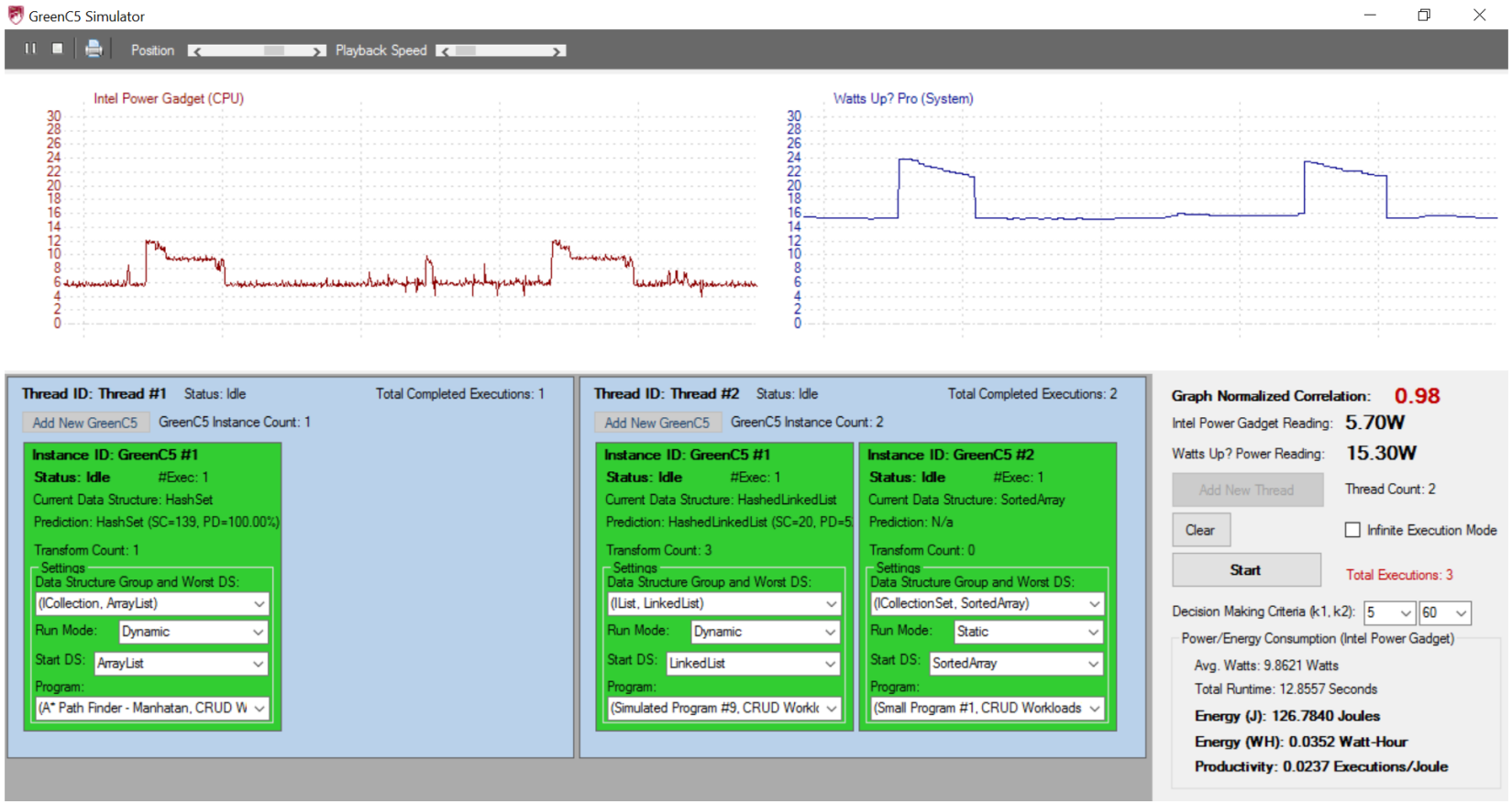


Figure 32. A Screenshot of the GreenC5 Simulator Application

One way to determine whether two power monitoring tools are similar, can provide similar results and can be substituted for each other in GreenC5 and green software development projects is to measure the graph correlation of their power readings. One limitation of the Intel Power Gadget is that the power consumption data it provides is for the CPU only, while Watts Up's power readings are for the computer system (CPU, GPU, Wi-Fi, display and others). As seen in Figure 32, the power reading samples of the two graphs are different so they need to be normalized before computing the graph correlation. In the implementation, the power sample rate of the Intel Power Gadget is set to 100 milliseconds/sample. However, the maximum power sample rate of the Watts Up? is at 1 sample/second. So, when plotting graphs and calculating the graph correlation, we adjust the power sample rate of the Watts Up? graph to have the same rate of 100 milliseconds/sample as of the Intel Power Gadget, to make it easy for comparison. The additional power reading samples are duplicated from numbers of the most recent readings in each time interval. We use a method used in signal processing called normalized correlation of discrete signals [104], as a measure of similarity of the two signals. The calculation is based on the following formula:

$$Norm\ Corr_{X,Y} = \frac{\sum_{n=0}^{N-1} X[n]Y[n]}{\sqrt{\sum_{n=0}^{N-1} X^2[n] \sum_{n=0}^{N-1} Y^2[n]}} \quad (4)$$

where, *X* and *Y* are the power datasets of Intel Power Gadget and Watts UP? being used to plot the graphs, respectively. *N* is the number of samples of *X* and *Y* datasets. *X[n]* and *Y[n]* is a power data sample in each of *X* and *Y* datasets. The top part of the formula, the numerator term, is for calculating the correlation of the two graphs. To get the normalized

correlation number, the top term is normalized by the denominator term. The denominator term normalizes and scales the weight of power data of both graphs to be at the same weight. The normalized correlation value is expected to fall between -1 and 1. A value closer to 1 indicates higher positive correlation of the two graphs.

8.4.1.2 Intel Power Gadget Evaluation Method

As described in Figure 33, we conduct experiments using our GreenC5 simulator running on COMP3 (with 6th-Gen Intel Core i5). Each of the 17 validation programs is executed with the same set of three use case scenarios. The graph correlation values of each execution will be recorded for the analysis.

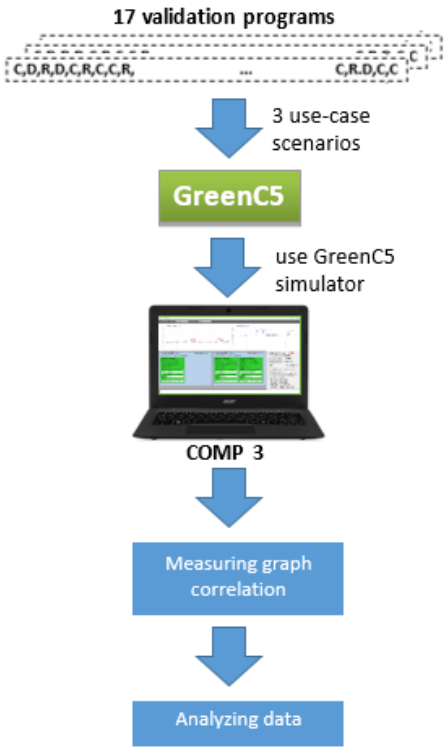


Figure 33. Intel Power Gadget Evaluation Process

8.4.1.3 Experimental Results

The total of 51 use-case scenarios are manually executed using the GreenC5 simulator. The average normalized correlation value of the two graphs is above 0.98. This value is very close to 1. The number indicates a strong positive correlation, meaning that the two graphs or power sample readings have a strong positive linear relationship. Even though the graphs have different weight of power readings (CPU vs. system), the two graphs seem to behave almost exactly the same and produce the same pattern of power readings when executing the same programs. As a result, we can conclude that Intel Power Gadget can be used as an alternative power measurement tool for future GreenC5 projects and any CPU-intensive program development.

8.4.2 Additional Analysis #2: A Performance Evaluation of Multiple Instances in Multiple Programs

The purpose of this additional case study is to see how GreenC5 performs and affects the system energy consumption in other use-case scenarios where multiple instances of the GreenC5 are executed sequentially in a single thread and asynchronously in separate threads/programs. We also want to compare its performance with the most and least preferable C5 data structures in each data structure group.

8.4.2.1 Experimental Setup

We create a project to simulate the use of C5 and GreenC5 data structures in different use-case scenarios on COMP3. Each use case and instance of the data structures are set to run either sequentially in a single thread or asynchronously in multiple threads. The energy consumption of each execution is captured by the Intel Power Gadget profiler

for analysis. We select 5 CRUD workloads from the 17 programs for the simulation, to be input to each of the data structure instances—2 simulated programs, 1 A* Path Finder, 1 Huffman Encoding and 1 Genetic Algorithm programs. Figure 34 depicts the evaluation process and steps of the experiment:

1. 5 use-case scenarios are created for the experiment—labeled as *UseCase#1*, *UseCase#2*, *UseCase#3*, *UseCase#4*, *UseCase#5*, where 1, 2, 3, 4, 5 instances of the C5 and GreenC5 data structures are initiated respectively.
2. Each use case is set to run on the least preferable C5 data structure instances, most preferable C5 data structures and dynamic-mode GreenC5 instances. For each GreenC5 instance, its data structure group property is set to each of different data structure groups. The initial C5 data structure property is set to start as the least preferable C5 data structure in each group and the default decision making criteria is set to SC>=5 and PD >=60%.
3. The use cases are set to run sequentially in a single thread application and asynchronously in different thread application. In the experiment, we utilize the .NET's Task Factory class library for parallel executions of the C5 and GreenC5 data structures.
4. The energy consumption of each execution is recorded using Intel Power Gadget for analysis. The total number of 2 runs are performed in this experiment.
5. The data analysis, graph plotting and energy savings are calculated against the least and most preferable C5 data structures in each data structure group.

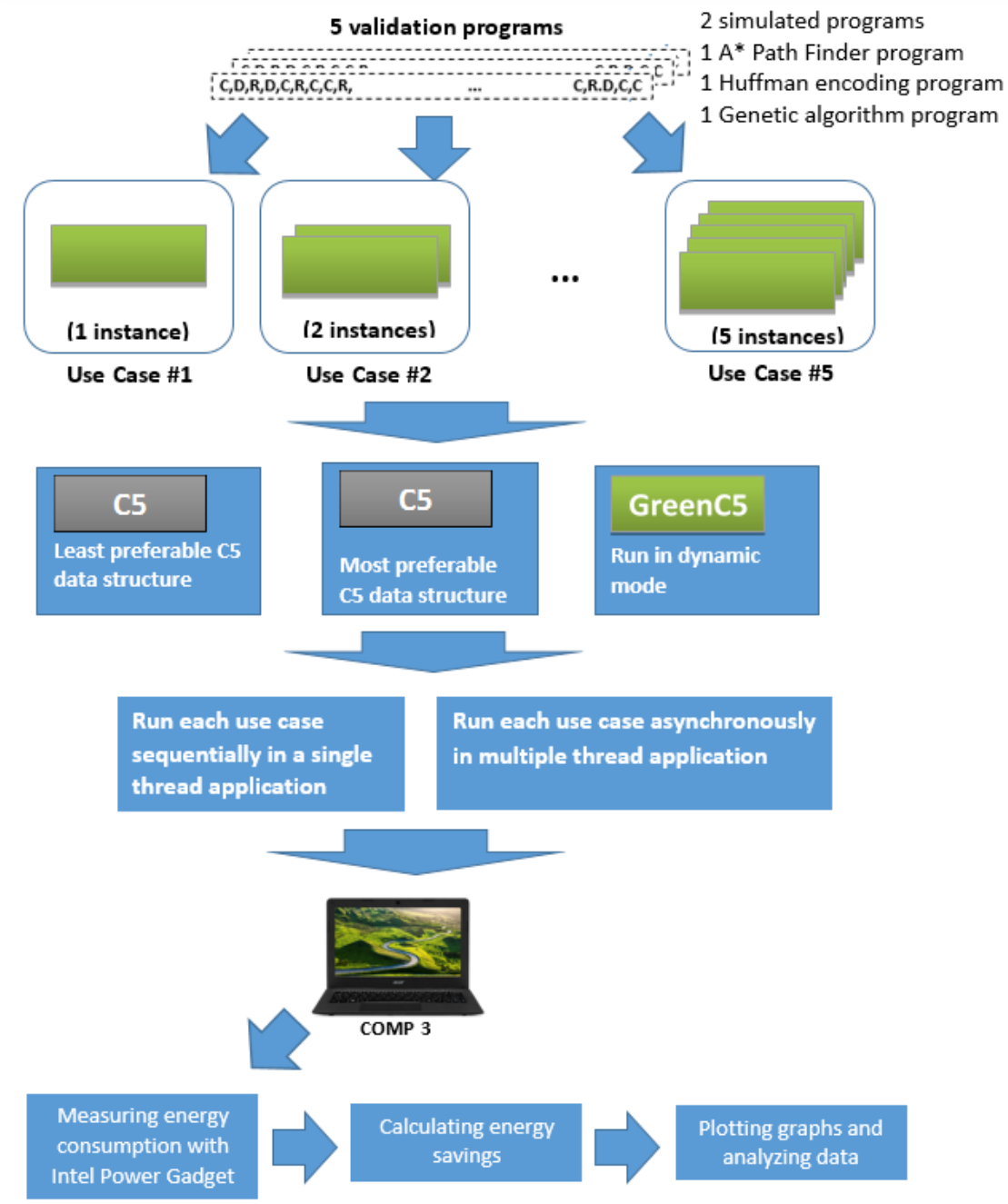


Figure 34. The Evaluation Process of GreenC5 in Different Use-Case Scenarios (Multiple Instances and Multiple Threads)

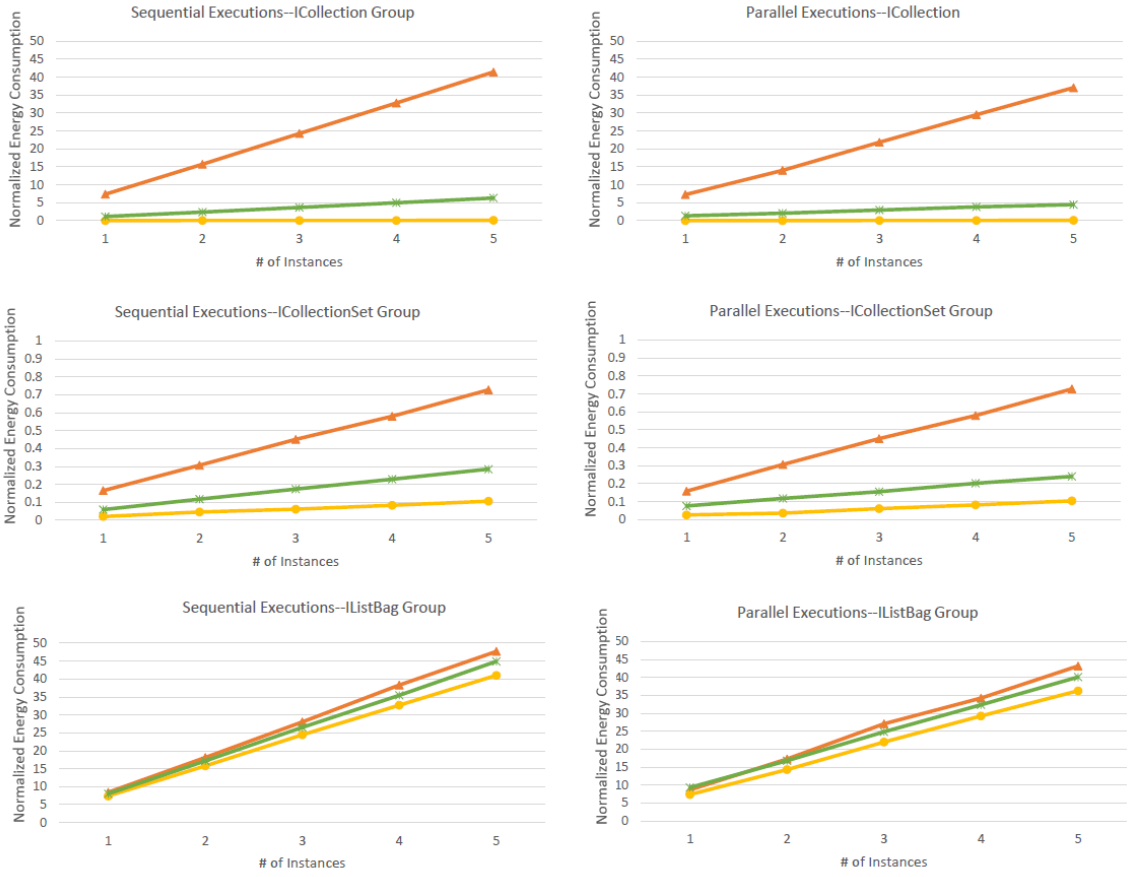
8.4.2.2 Experimental Result

The energy consumption data of the program executions from the two runs are averaged and normalized so that the numbers range from 0 to 100. Also, for better

comparison, the energy consumption data of the most and least preferable C5 data structures are plotted together in the same graphs with GreenC5. The energy data are averaged and grouped by program and data structure group and displayed by sequential and parallel executions, side-by-side for comparison. Figure 35 shows some of the energy consumption graphs. The left and right-side graphs display the energy data of the sequential and parallel executions respectively. Figure 35(a) displays the graphs of the overall executions of all 5 programs. Figure 35(b) shows three of the six energy consumption graphs by data structure group—from the top down, ICollection, ICollectionSet and IListBag respectively. The top legends in the figure, from left to right, represent energy consumption of the least, the most preferable C5 data structures and the GreenC5 data structure, respectively. In each graph, the y-axis is the average normalized energy consumption (in joules) of program executions. The x-axis represents program executions of different use cases—in which 1, 2, 3, 4 and 5 instances of the data structure are executed.

Overall, the energy consumption of the C5 and GreenC5 data structures are increasing when the number of instances increases. The trend is similar in both sequential and parallel executions. This shows that the GreenC5 seems to funtion well similar to the original C5 data structures when multiple instances are executed together. However, parallel executions tend to use less energy than sequential executions. This can be seen in the slopes of the right-side graphs in Figure 35 that are lower than that of the left-side graphs. To support this, Table 5 shows the average energy savings of parallel executions of the GreenC5 data structures versus sequential executions per number of instances. Executing one instance of the data structure in a separate thread is not normally done and

(a)



(b)

Figure 35. Energy Consumptions of Sequential and Parallel Executions of GreenC5 and C5 Data Structures, (a) All Data Structure Groups and (b) ICollection, ICollectionSet and IListBag Groups.

it does not gain any energy savings, so it is not included in the table. The percentage numbers show the extra energy savings or energy efficiency improvement gained by parallel executions against sequential executions. For multiple instances, when the number of instances increase, the energy efficiency improvement tends to also increase. From the data, at five instances, the energy efficiency improvement of parallel executions can reach almost 11%. Therefore, we recommend parallel executions over sequential executions of the data structures whenever possible for maximum energy savings. Moreover, even though the net energy consumption seems to be lower for parallel executions, but if we look at the execution time and power consumption data, parallel executions normally take less time to complete but consume more power to run than sequential executions.

Table 5. The Energy Efficiency Improvement of Parallel Executions vs. Sequential Executions of GreenC5 by Number of Instances

Number of Instances	**Energy Efficiency Improvement**
2	6.35%
3	7.18%
4	9.26%
5	10.70%

Additionally, from the graphs in Figure 35, the energy consumptions of the GreenC5 data structure, as expected, falls between the energy consumptions of the least and most preferable C5 data structures in each data structure group. The overall potential energy savings of the GreenC5 for all numbers of instances (the energy differences between the worst or least preferable C5 data structures and the GreenC5) about 47% and 45%, on average, for sequential and parallel executions respectively (see Figure 35a and the last

row of Table 6). The graphs also show that, for all number of instances, potential energy savings in ICollection, ICollectionBag and ICollectionSet are higher than that of the IList, IListBag and IListSet groups. These can be seen by the wider gaps between the least preferable lines and the GreenC5 lines in the graphs and the higher numbers shown in Table 6. Also, for multiple-stance executions, the data structures in ICollection and ICollectionBag provide the highest numbers of potential energy savings while ones in IListBag provide the lowest potential energy savings. Also, from the data, it seems that GreenC5 is also scalable just like the original C5 collection. GreenC5 seems to continue to function and perform well when the number of instances and threads increase. However, to make the assumption more valid, more evaluations should be conducted with higher numbers of instances and application threads. In addition, the results produced by Intel Power Gadget data seem to be correlated with ones produced by Watts Up? power meters in the previous experiments. This supplemental result also confirms that the Intel Power Gadget tool is a good alternative energy profiler tool for our future GreenC5 projects.

Table 6. Average Potential Energy Savings of Parallel and Sequential Executions of GreenC5 for All Numbers of Instances by Data Structure Group

Data Structure Group	**Sequential**	**Parallel**
ICollection	84.51%	85.54%
ICollectionBag	84.30%	85.91%
ICollectionSet	61.66%	62.33%
IList	22.68%	16.45%
IListBag	5.78%	3.17%
IListSet	24.91%	12.63%
Average	**47.31%**	**44.43%**

8.4.3 Additional Analysis #3: An Initial Exploration of Decision Making Criteria

The key functionality of the decision maker is to control the number of the transformations and timing of each transformation during each program execution so that the overall energy saving is at the maximum level. If there is no decision making mechanism to control the transformations, the number of transformations can be as high as the number of predictions made by the predictor component. If the transformations take place too many times, the overhead and energy cost of the transformations can be too high, making the GreenC5 perform worse than the original C5 data structure and the energy saving opportunity windows to be missed. The decision maker's main role is therefore to control the transformations to take place only when necessary in order to maximize energy savings. The main mechanism is the controlling of *k1* and *k2* values in the SC>=*k1* and PD>=*k2* decision making criteria.

The purpose of this additional analysis study is to systematically explore for the optimal decision making criteria of the Decision Maker component that can control the number and timing of GreenC5's transformations so that the overall energy savings are at the maximum level. Also, the purpose is to see how the criteria values impact the energy savings and how the GreenC5 transforms and adapts when the criteria change. The ultimate goal is to find the optimal values of *k1* and *k2* that can be used as the default decision making criteria in the GreenC5 data structure. In this case study, the goal is not to explore all possible values, but to explore the values of *k1* and *k2* only at some granularity, in order to understand and get some general ideas of how the decision making criteria impact the potential energy savings and transformation and adaptation behaviors of the GreenC5. The

study is also to demonstrate how an exploration of decision making criteria can be conducted.

8.4.3.1 Experimental Setup

By utilizing the GreenC5 and Intel Power Gadget class libraries, a new experimental project is created and executed on COMP3. The project is to automate the use of GreenC5 with different values of decision criteria parameters. The goal is to search

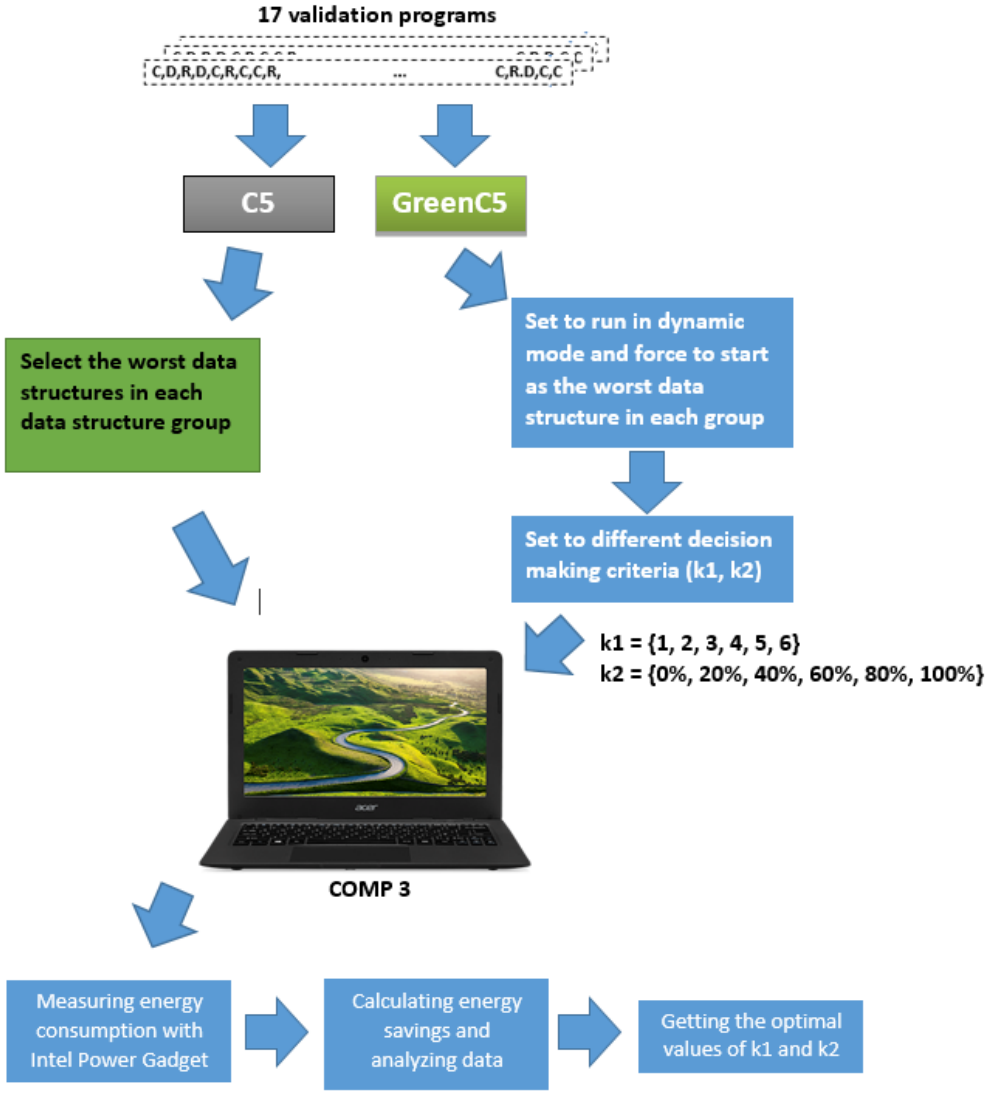


Figure 36. Decision Making Criteria Exploration Process

for *k1* and *k2* values that make GreenC5 instances perform with highest potential energy savings. The energy savings are calculated by comparing with the base energy consumption of the least favorable C5 data structures in each of the data structure groups. The experimental method is described in Figure 36. In the method, the 17 validation programs are used as the input workload to both the C5 data structures and GreenC5. For GreenC5, the decision making criteria are varied with different values of *k1* and *k2* in each the program execution. The energy consumption data are captured by the Intel Power Gadget and the internal data structure transformations of the GreenC5 are also traced for analysis.

In this experiment, to speed up the energy profiling process, only the first 120,000 CRUD operations of the 17 programs are used in each execution. Because of the limited size of the CRUD operations, the values of *k1* to be explored are only from 1 to 6. The values of *k1* and *k2* are varied at the following granularity:

k1 = {1, 2, 3, 4, 5, 6}

k2 = {0%, 20%, 40%, 60%, 80%, 100%}

For example, for *k1* = 6, the maximum number of transformations can take place is 2 because the prediction and decision making are made every 10,000 observed CRUD operations (value of L). For *k1* =1, the maximum number of transformation can take place is 12. On the other hand, the percentage values of *k2* are multiples of 20. Together, values of *k1* and *k2* thresholds in the decision making criteria controls the number of transformations that can result in different amount of overall potential energy savings.

8.4.3.2 Experimental Result

From all program executions, we found that the number of actual transformations of the GreenC5's internal data structure ranges from 0 to 6. There are few program executions with some values of *k1* and *k2* that cause zero transformation during the program executions—for example, in program executions of simulated program #1 and #2, where the values of *k1* and *k2* are set to 4 and 100% and 6 and 100% respectively. This is due to the fact that, during the program executions, there is no window in the CRUD sequence that the prediction probability values (PD) produced by the predictor component reaches 100%. Also, there are many of the program executions, mostly in ICollection, ICollection and ICollectionBag groups that have at least 1 program transformation, In this case, it is mainly because that GreenC5 instances are set to start as the worst data structure choices in each data structure group. And, the GreenC5 instances are able to adapt/transform to the more energy-efficient data structures. Also, there is no additional transformation during each program execution because either the predictor predicts the same energy-efficient data structure or because there is no window for the transformation made by the decision maker. On the other hand, there are also many program executions that shows multiple transformations during the program executions of the first 120K CRUD operations. Many can be seen in IList, IListBag and IListSet data structure groups. These result examples with multiple transformations demonstrate that the transformations

of the GreenC5 data structure are indeed dynamic. To see dynamic transformations of the GreenC5, users can also use our GreenC5 simulator to simulate the scenarios.

To understand the dynamic transformation and adaptation process in our GreenC5 data structure, Figure 37 shows one example of the actual program executions from our experiment showing how the dynamic transformation and adaptation process work. The figure depicts program executions of the Simulated Program # 9 in the IList data structure group. It explains how and when the predictions and transformations take place and how different decision making criteria can change the pattern and number of transformations that results in different amount of energy consumption and potential energy savings. From the figure, the program workload sequence contains 120,000 CRUD operations that can be sub-divided into 12 L-long subsequences, where L = 10,000 operations. Each subsequence has different features of %C, %R, %U, %D, N and G that can be observed by the Green component of the GreenC5. The observed features are, at runtime, then sequentially input to the ANN Classifier component and translated/mapped into a sequence of 12 energy-efficient C5 data structures in the IList group. Among the choices in the IList group, in this example, the Classifier identifies HashedLinkedList and HashedArrayList as the energy-efficient data structures to best perform different subsequences of the program workload. The classified data structure sequence is also shown in the figure (one that is produced by the ANN Classifier).

The IList group has 5 available choices of C5 data structures—ArrayList (AL), LinkedList, (LL) HashedArrayList (HAL), HashedLinkedList (HLL) and SortedArray (SA). Among the choices in this group, we have already identified that LinkedList is the

least preferable data structure because it consumes highest energy consumption when performing the same operations. So, LinkedList is used as the base data structure for comparing with the GreenC5 and calculating the potential energy savings. In this example, LinkedList uses about 332.41 joules on COMP3 to perform the program's workload. To search for an optimal decision making criteria in this experiment, GreenC5 is varied with different values of *k1* and *k2* in the decision making criteria. The bottom three sequences in the figure show actual transformations of the internal data structure with different decision making criteria. In the first sequence with the SC>=1 && PD>=60% decision making criteria, there are total of 3 transformations that take place dynamically and form a GreenC5's internal data structure sequence of LL, HAL and HLL data structures. The sequence or the transformation/switching pattern of internal data structure consumes 72.61% less energy than just staying statically as a LinkedList. With different decision criteria, the numbers of transformations and the switching patterns of the GreenC5 are also different, consuming different amount of energy. In the second sequence of SC>=5 && PD>=60% decision criteria, there are total of 2 transformations that form another transformation/switching pattern, consuming 66.80% less energy than the LinkedList. This pattern has lower potential energy saving than the previous one. The last sequence is when the decision making criteria being set to SC>5 && PD>=100%. This criteria makes no change in the transformation and switching pattern. So, the GreenC5 stays statically as a LinkedList. In this case, there is no gain in energy saving. The potential energy saving number of -2.55% in the figure indicates some overhead in the GreenC5 and energy fluctuations of the base system. Also, as you can see in the three sequences, different

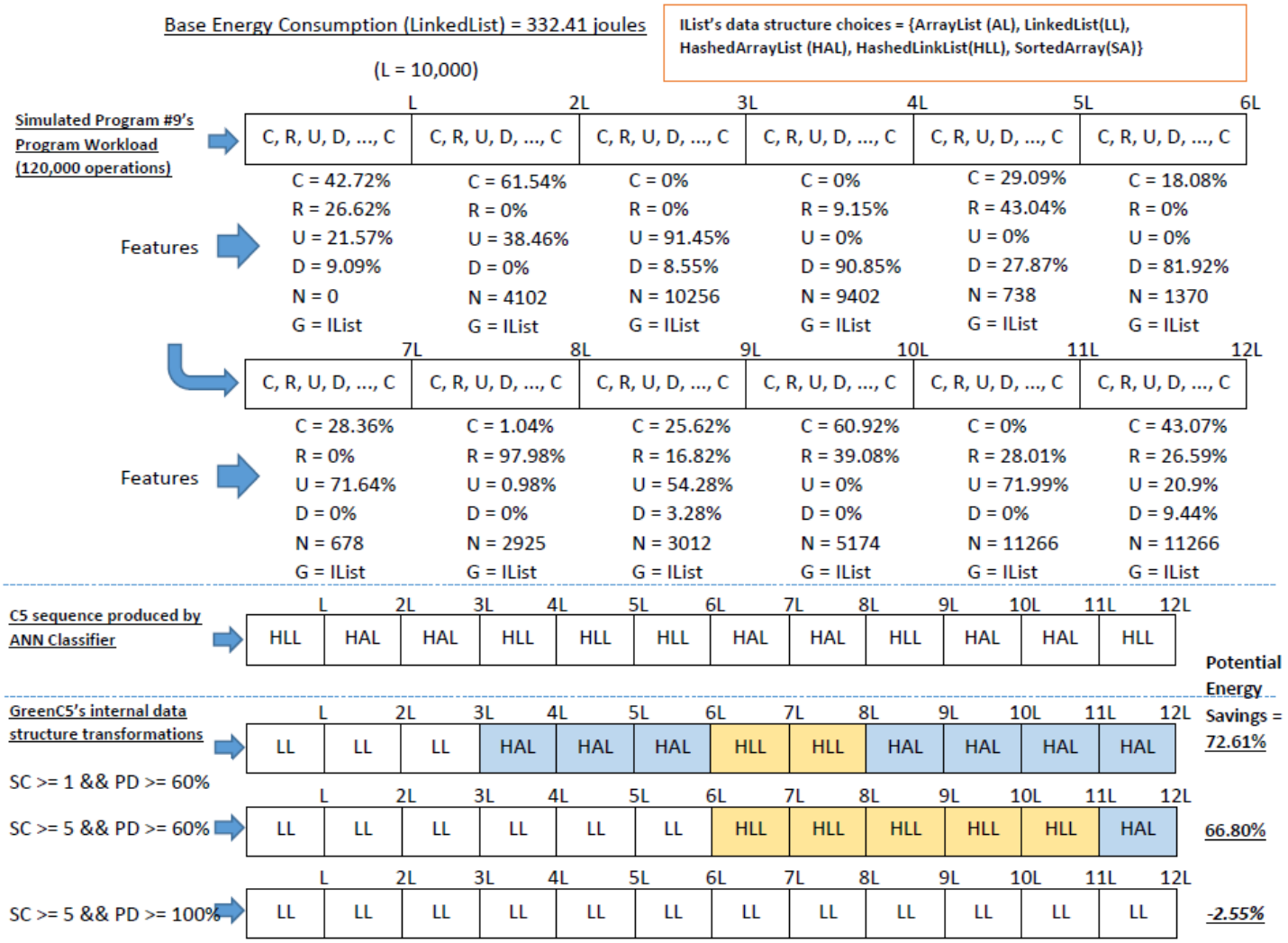


Figure 37. GreenC5's Internal Dynamic Transformation Example

transformation pattern and different timing of each transformation do impact the energy consumption of the dynamic-mode GreenC5 data structure. The decision making criteria of the one with highest potential energy saving should be selected as the optimal decision making criteria. The goal of this additional case study is to do a brute-force search in a larger solution space for better decision making criteria.

Ultimately, a full exploration of the decision making criteria should be done with wider range and higher granularity of *k1* and *k2* values. However, it will take a very long time for the exhaustive search to be completed. Therefore, in this exploration, the search is limited by some values of *k1* and *k2* as specified in the previous section. From the two runs of program executions, the energy data of all 17 program executions are averaged, analyzed and plotted into surface graphs of potential energy savings as shown in Figure 38 and Figure 39. The x and y-axis of the graphs are the *k1* and *k2* values of the decision making

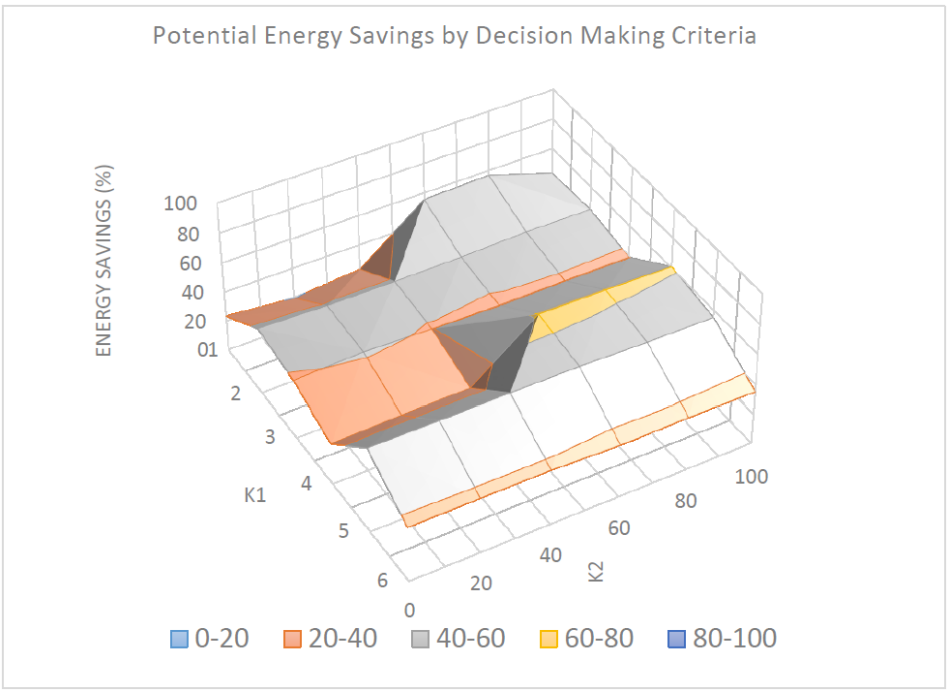


Figure 38. Overall Potential Energy Savings by Decision Making Criteria

criteria, while the z-axis are the percentage potential energy savings produced by each combination of the two decision making criteria. The legends beneath the graphs present different ranges of potential energy saving numbers and areas on each graph surface.

Figure 38 shows the overall potential energy savings by decision making criteria. The energy saving values are the overall average of all program executions. Overall, in Figure 38, the combination of SC and PD criteria (*k1* and *k2* values) clearly impact the potential energy savings because the graph clearly shows uneven surface. From the experiment, the overall average potential energy savings, ranging from 19% to almost 62%, are plotted on the graph. To determine the optimal decision criteria, we select ones in the ranges of *k1* and *k2* values that produce the highest potential energy savings and ones in the highest areas on the graph surface. Also, for the PD criteria (the probability value of a prediction), the *k2* values are also limited by the number of available data structure choices in each data structure group. Because the PD value is produced by a bigram-based predictor, the PD values are always in the range between the percentage of *1/n* and 1, where *n* is the number of data structure choices in each data structure group. This criteria is also used in the selection of a proper optimal decision making criteria. Overall, by looking at the energy saving data and graph surface, our estimated decision making criteria is determined to be SC>=4 && SC<=5 and PD>=60%. This criteria values are clearly in the top area of the surface graph in Figure 38 (60-80 area). These optimal criteria are to be set as the default decision making criteria of our GreenC5 data structure. Surprisingly, our

previous random-select decision-making criteria of SC>=5 && PD >=60% falls right on this optimal area.

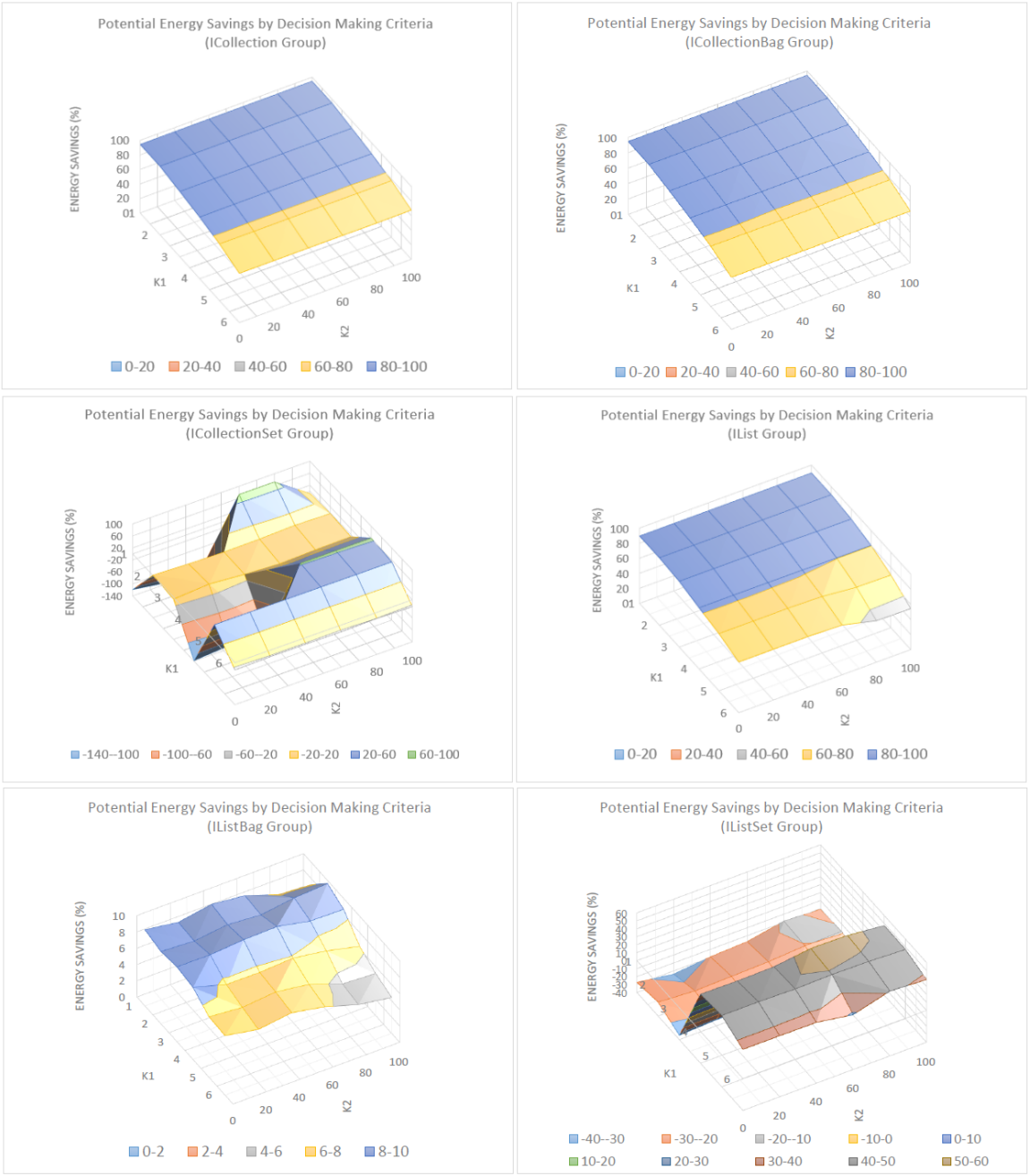


Figure 39. Potential Energy Savings by Decision Making Criteria and Data Structure Group

In addition, Figure 39 presents surface graphs of GreenC5's potential energy savings by decision making criteria and by data structure group. These six graphs when combined together will becomes the graph shown in Figure 38. Clearly, the surface graphs and potential energy savings by data structure are different. The graphs of ICollection, ICollectionBag and IList groups look similar. For these three graphs, it seems that the PD criteria (*k2* values) do not have much impact on the energy savings. As you can see along the *k2*-axis, the potential energy savings changes minimally when *k2* values change. In contrary, along the *k1*-axis, the energy savings change more dramatically when *k1* values change. From these three graphs, the higher amount of energy savings can mostly be made when values of *k1* are smaller. For the other three graphs, both *k1* and *k2* parameters have some impact on the GreenC5's transformation patterns and potential energy savings. In particular, for the graphs of ICollection and IListSet graphs, the optimal areas can clearly be spotted on the graph surface.

Table 7. Recommended Optimal Decision Making Criteria by Data Structure Group

Data Structure Group	**# of Data Structure Choices**	**Decision Making Criteria**	
		Criteria 1 (k1)	**Criteria 2 (k2)**
ICollection	9	SC>=1 && SC<=2	PD>11%
ICollectionBag	4	SC>=1 && SC<=2	PD>=25%
ICollectionSet	5	SC=4	PD>=60%
IList	5	SC>=1 && SC<=2	PD>=20%
IListBag	2	SC>=2 && SC<=3	PD>=50%
IListSet	3	SC<=4 && SC<=5	PD>=60%
Overall (default)		**SC>=4 && SC<=5**	**PD>=60%**

Finally, for better energy savings, we recommend that the decision making criteria of GreenC5 should be different by the data structure group property. The list of our recommended decision making criteria by data structure group is presented in Table 7. These optimal decision making criteria are derived from both analyzing the potential energy saving data and looking at the surface graphs. They are not considered to be the best optimal criteria but certainly valid for future version of our GreenC5 collection. From the result, our previous decision making criteria of SC>=5 && PD >= 60% is still considered valid and can still be used as the default criteria. If you notice the *k2* values in the PD criteria column, the bigram probability value criteria of *1/n* and 1 is also used in determining the criteria. This is the reason why the *k2* values are always not less than the percentage of *1/n*, where *n* is the number of data structure choices by group (shown in the second column).

CHAPTER 9: CONCLUSION AND FUTURE WORK

9.1 Conclusion

With a vision to see smart and adaptive green objects and components be part of green software programs in the future, we have detailed a concept, an architecture and prototype for building adaptive green data structures that can intelligently adapt for energy efficiency. We provide empirical evidence that there exists energy saving opportunities in C5 dynamic data structures, which may be present in other interface-based, object-oriented dynamic data structures as well. Using a "select the right data structure for the right workload" approach, we demonstrate how the C5 data structure selection process can be automated with machine learning tools such as Artificial Neural Networks and n-gram based predictors. The results show that the models can accurately classify and predict data structures for energy efficient computing.

Our green data structure prototype, GreenC5, demonstrates how the concept can actually be implemented. Our simple technique of decision making can help the GreenC5 to decide when to transform dynamically for energy efficiency. The predictive model can help the data structure know how to adapt by correctly transforming to different data structure choices. The result shows that the GreenC5 can efficiently adapt for energy efficiency with minimum overhead. The a priori knowledge can potentially be universal where only one single model is needed and can be used in different machines, improving the user-friendliness of the data structure. The work could also be applied in other interface-

based objects as well and is an essential groundwork for building fully functional adaptive green data structures in the future.

Along with the adaptive green data structure, we also include an additional study of the cache system on an FPGA development board and demonstrate an in-depth manual process of a power-performance tradeoffs using the Pareto optimality principle. The results suggest that the Pareto optimal configurations do exist in the cache configurations and some optimal configurations might not be as expected when analyzing the live power consumption data. We observe that the optimal configurations are sparse in the cache design space, are inconsistent across the benchmark and counterintuitive in many cases, making power-performance optimization processes hard to implement without analysis from actual data. From this study, we learn that even something very low-level like the cache system can impact the power-performance analysis significantly and unpredictably. This might not be captured if the analysis is done using data from a power model. As a result, we also suggest the need for tools and methodologies that operate directly on data gathered from the systems themselves.

9.2 Future Work

This dissertation proposes a working prototype of an adaptive green data structure, the GreenC5. However, the processes of gathering power data and machine learning are done in an offline manner. The analysis is done on the power data gathered by a hardware-based meter, in which might not be practical in real life and is limited only to the offline learning capability. Our future work will therefore focus on incorporating the green data structure with alternative, more accurate, finer-grained and more practical power meters.

One of the alternative tools that will be used in our future GreenC5 projects is the Intel Power Gadget as described in one of our additional case studies. With a more accurate software-based meter, the future work can include adding the reinforcement and online learning capabilities to the adaptive green data structure, in which the data structure can learn and adapt to the workload automatically with minimal or no base knowledge. The future work can also include searching for better architectures and designs of the adaptive green data structure, with different methods of dynamic selection, decision making and self-adaptation and transformation for energy efficiency. We also suggest that the concept is to be applied in other data structures of different programming languages and platforms and other interface-based software components and objects.

BIBLIOGRAPHY

[1] M. P. Mills, "The cloud begins with coal. Big data, big networks, big infrastructure, and big power—an overview of the electricity used by the global digital ecosystem", *Digital Power Group*, August 2013. http://www.tech-pundit.com/wp-content/uploads/2013/07/Cloud_Begins_With_Coal.pdf?c761ac

[2] Greenpeace, "The iPad, internet, climate change link in the spotlight", *Greenpeace International*, 30 March 2010. http://www.greenpeace.org/international/en/news/features/ipad-cloud-climate-change-290310/

[3] J. Michanan, R. Dewri, and M. J. Rutherford, "Understanding the power-performance tradeoff through Pareto analysis of live performance data", *in International Green Computing Conference (IGCC)*, November 2014.

[4] R. Kumar, K. I. Farkas, N. P. Jouppi, P. Ranganathan, and D. M. Tullsen, "Single-ISA heterogeneous multi-core architectures: The potential for processor power reduction", in *ACM/IEEE MICRO*, pp. 81–92, 2003.

[5] K. Baynes, C. Collins, E. Fiterman, B. Ganesh, P. Kohout, C. Smit, T. Zhang, and B. Jacob, "The performance and energy consumption of embedded real-time operating systems", in *IEEE Transactions on Computers*, vol.52, no.11, pp. 1454-1469, November 2003.

[6] F. Steimann and P. Mayer, "Patterns of Interface-Based Programming", in *Journal of Object Technology*, vol. 4, no. 5, pp. 75-94, July-August 2005. http://www.jot.fm/issues/issue_2005_07/article1

[7] T. King, "Dynamic data structures: Theory and application", *Academic Press*, 1992.

[8] R. Horvick, "Data Structures Succinctly Part 1", Technology Resource Portal, *Syncfusion Inc.*, 2012.

[9] N. Kokholm and P. Sestoft, "The C5 Generic Collection Library for C# and CLI", Technical Report ITU-TR-2006-76, *IT University of Copenhagen*, January 2006. https://www.itu.dk/research/c5/latest/ITU-TR-2006-76.pdf

[10] S. Millett and N. Tune, "Patterns, Principles, and Practices of Domain-Driven Design", *John Wiley & Sons*, 20 April 2015.

[11] J. Eastep, D. Wingate and A. Agarwal, "Smart data structures: an online machine learning approach to multicore data structures", in *Proceedings of the 8th ACM International Conference on Autonomic Computing*, pp. 11-20, 2011.

[12] E. G. Daylight, D. Atienza, A. Vandecappelle, F. Catthoor and J. M. Mendias, "Memory-access-aware data structure transformatins for embedded software with dynamic data accesses", in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol.12, no.3, pp. 269-280, March 2004.

[13] J. Flinn and M. Satyanarayanan, "Energy-aware adaptation for mobile applications", *ACM SIGOPS Operating Systems Review*, vol. 34, no. 2, pp. 13-14, 2000.

[14] J. Aguilar-Saborit, P. Trancoso, V. Muntes-Mulero and J.L. Larriba-Pey, "Dynamic adaptive data structures for monitoring data streams", *Data & Knowledge Engineering*, *Science Direct*, vol.66, pp. 92-115, March 2008.

[15] J. Ansel, "Autotuning programs with algorithmic choice", Doctoral Dissertation, *Massachusetts Institute of Technology*, 2014.

[16] "Watts Up? Plug Load Meters", *Watts Up?*. https://www.wattsupmeters.com/secure/products.php?pn=0

[17] G. Franco, "A-Star (A*) Implementation in C#", *CodeGuru*, 6 September 2006. http://www.codeguru.com/csharp/csharp/cs_misc/designtechniques/article.php/c12527/AStar-A-Implementation-in-C-Path-Finding-PathFinder.htm

[18] S. Natav, "Huffman Encoding - From Implementation to Archive", *CodeProject*, 18 May 2009. http://www.codeproject.com/Articles/36415/HuffMan-Encoding-From-Implementation-to-Archive-Pa

[19] B. Lapthorn, "A Simple C# Genetic Algorithm", *CodeProject*, 21 August 2003. http://www.codeproject.com/Articles/3172/A-Simple-C-Genetic-Algorithm

[20] J. McCaffrey, "Neural Networks Using C#", *Technical Resource Portal*, Syncfusion Inc., 2014.

[21] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection", in *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, vol. 2, pp. 1137-1143, 1995.

[22] I. Millington, "Artificial Intelligence for Games", *Morgan Kaufmann Publisher*, Elsevier Inc., 2006.

[23] J. D. Vega, "Intel Power Gadget." *Intel Developer Zone*, 7 January 2014, https://software.intel.com/en-us/articles/intel-power-gadget-20

[24] S. Naumann, M. Dick, E. Kern and T. Johann, "The GREENSOFT Model: A reference model for green and sustainable software and its engineering", *Sustainable Computing: Informatics and Systems*, vol. 1, no. 4, (2011), pp. 294-304.

[25] S. S. Mahmoud and I. Ahmad, "A Green Model for Sustainable Software Engineering", *International Journal of Software Engineering and Its Applications*, vol. 7, no. 4, July, 2013.

[26] P. Lago and I. Crnkovic, "Framing sustainability as a property of software quality", *Article in Communication of the ACM*, October 2015.

[27] J. Michanan, R. Dewri and M. J. Rutherford, "Predicting Data Structures for Energy Efficient Computing", in *International Green and Sustainable Computing Conference (IGSC)*, December 2015.

[28] E. Freeman, E. Robson, B. Bates and K. Sierra. "Head first design pattern", *O'Reilly Media, Inc.*, October 2004.

[29] A. Gupta, T. Zimmermann, C. Bird, N. Nagappan, T. Bhat and S. Emran, "Detecting Energy Patterns in Software Development", *Microsoft Research. Technical Report*, 2011.

[30] F. Chen, J. Grundy, Y. Yang, J. Schneider and Q. He, "Experimental Analysis of Task-based Energy Consumption in Cloud Computing Systems", in *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*, pp. 295-306, 2013.

[31] I. A. Gul and W. Hasselbring, "Towards Power Consumption Reduction by User Behavior Monitoring at Application level", in *International Conference on Architecture of Computing Systems (ARCS)*, February 2010.

[32] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze and D. Grossman, "EnerJ: approximate data types for safe and general low-power computation", in *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation*, vol. 46, issue 6, pp. 164-174, June 2011.

[33] C. Sahin, F. Cayci, I.L.M Gutierrez, J. Clause, F. Kiamilev, L. Pollock, and K. Winbladh, "Initial explorations on design pattern energy usage", in *2012 First International Workshop on Green and Sustainable Software (GREENS)*, pp. 55-61, IEEE, 2012.

[34] D. Jurafsky and J. H. Martin, "Speech and Language Processing", *Prenctice Hall*, May 16, 2008.

[35] C. Sterling, “Energy Consumption tool in Visual Studio 2013”, *Microsoft Application Lifecycle Management*, July 10, 2013, http://blogs.msdn.com/b/visualstudioalm/archive/2013/07/10/energy-consumption-tool-in-visual-studio-2013.aspx

[36] J. Arora, “Introduction to optimum design”, *Academic Press*, 2004.

[37] “Atlys Board Reference Manual”, *Digilent Inc.*, 2012, http://www.digilentinc.com/Data/Products/ATLYS/Atlys_rm.pdf.

[38] “Atlys board support files for EDK BSB wizard. Supports EDK 13.2 - 14.2 for both AXI and PLB buses”, *Digilent Inc.*, 2012, http://www.digilentinc.com/Data/Products/ATLYS/Atlys_BSB_Support_v_3_6.zip.

[39] A. Gordon-Ross, F. Vahid, and N. D. Dutt, “Fast configurable-cache tuning with a unified second-level cache”, in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, pp. 80-91, 2009.

[40] Y. Hara., H. Tomiyama, S. Honda, H. Takada, and K. Ishii, “CHStone: A benchmark program suite for practical c-based high-level synthesis”, in *2008 IEEE International Symposium on Circuits and Systems (ISCAS 2008)*, pp. 1192-1195, IEEE, 2008.

[41] M. Harman, W. B. Langdon, Y. Jia, D. R. White, A. Arcuri, and J. A. Clark, “The GISMOE challenge: constructing the Pareto program surface using genetic programming to find better programs (keynote paper)”, in *2012 Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering (ASE 2012)*, pp.1-14, 3-7 Sept 2012.

[42] J. C. McCullough, A. Yuvraj, J. Chandrashekar, S. Kuppuswamy, A. C. Snoeren, and R. K. Gupta, “Evaluating the effectiveness of model-based power characterization”, in *USENIX Annual Technical Conference*, 2011.

[43] C. J. Petrie, T. A. Webster, and M. R. Cutkosky, “Using Pareto Optimality to Coordinate Distributed Agents”, *AIEDAM Special Issue on Conflict Management* vol. 9, pp. 269-281, 1995.

[44] R. Bekkererman, M. Bilenko and J. Langford, “Scaling up machine learning—parallel and distributed approaches”, *Cambridge University Press*, 2012.

[45] C. Su and A. M. Despain, “Cache design trade-offs for power and performance optimization: a case study”, in *Proceedings of the 1995 International Symposium on Low Power Design*, ACM, 1995.

[46] Xilinx Inc., “Field Programmable Gate Array (FPGA),” 2013, http://www.xilinx.com/training/fpga/fpga-field-programmable-gate-array.htm.

[47] N. Lopez, O. Aguirre, J. F. Espiritu, and H. A. Taboada, “Using game theory as a post-Pareto analysis for renewable energy integration problems considering multiple objectives”, in *Proceedings of the 41st International Conference on Computers & Industrial Engineering*, pp. 678-683, 2011.

[48] C. Zhang, F. Vahid, and W. Najjar, “A highly configurable cache architecture for embedded systems”, in *Proceedings of 30th Annual International Symposium on Computer Architecture*, IEEE, 2003.

[49] F. Rezzi, L. Collamati, M. Costagliola, and M. Cutrupi, “Battery management in mobile devices”, in *Frequency References on Power Management for SoC and Smart Wireless Interfaces*, *Springer International Publishing*, pp.147-168, 2014.

[50] R. Mittal, A. Kansal, and R. Chandra, “Empowering developers to estimate app energy consumption”, in *Proceedings of the 18th Annual International Conference on Mobile Computing and Networking*, pp. 317-328, ACM, 2012.

[51] S. Rivoire , P. Ranganathan and C. Kozyrakis, “A comparison of high-level full-system power models”, in *Proceedings of the 2008 Conference on Power Aware Computing and Systems*, pp.3-3, 2008.

[52] J. Horn, N. Nafpliotis, and D. E. Goldberg, “A niched Pareto genetic algorithm for multiobjective optimization”, in *Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence*, vol.1, pp. 82-87, 1994.

[53] S.Ruth, “Green IT More Than a Three Percent Solution?”, *Internet Computing*, IEEE , vol.13, no.4, pp.74-78, July-Aug. 2009.

[54] D. Wang, “Meeting Green Computing Challenges”, in *Proceedings of the International Symposium on High Density packaging and Microsystem Integration*, vol., no., pp.1-4, 26-28 June 2007.

[55] L. Smarr, “Project Greenlight: optimizing cyber-infrastructure for a carbon-constrained”, *World Computer*, pp.22-27. January 2010.

[56] BTW, “The Dividends from Green Offices”, *Bloomberg Business*, November 24. 2009, http://www.bloomberg.com/bw/stories/2009-11-24/btw

[57] “Going Green May Reduce Your Taxes—IRS Tax Tip 2010-66”, *IRS*, 06 June 2013. https://www.irs.gov/uac/Going-Green-May-Reduce-Your-Taxes

[58] "Consumer-goods' brands that demonstrate commitment to sustainability outperform those that don't", *Nielson*, 12 October 2015. http://www.nielsen.com/us/en/press-room/2015/consumer-goods-brands-that-demonstrate-commitment-to-sustainability-outperform.html

[59] K. S. Vallerio , L. Zhong , N. K. Jha, "Energy-Efficient Graphical User Interface Design", in *IEEE Transactions on Mobile Computing*, v.5 n.7, p.846-859, July 2006.

[60] G. Mackey, S. Sehrish, J. Bent, J. Lopez, S. Habib and J. Wang, "Introducing map-reduce to high end computing", *Petascale Data Storage Workshop*, vol. 3, pp.1-6, 17 November 2008.

[61] Y. Fei, S. Ravi , A. Raghunathan and N. K. Jha, "Energy-optimizing source code transformations for operating system-driven embedded software", in *ACM Transactions on Embedded Computing Systems (TECS)*, vol.7, no.1, pp.1-26, December 2007.

[62] P. Nastu, "82 Percent of Consumers Buy Green, Despite Economy", *Environmental Leader*, 5 February, 2009. http://www.environmentalleader.com/2009/02/05/82-percent-of-consumers-buy-green-despite-economy/#ixzz3y0gZzLUq

[63] Developer Network, "Three-Layered Services Application", *Microsoft*, 22 January 2016. https://msdn.microsoft.com/en-us/library/ff648105.aspx

[64] G. Qu and M. Potkonjak, "Energy minimization with guaranteed quality of service", In Proceedings of the International Symposium on Low Power Electronics and Design, p.43-49, 25-27July, 2000.

[65] L. Zhong and N. K. Jha, "Energy efficiency of handheld computer interfaces: limits, characterization and practice", in *Proceedings of the 3rd International conference on Mobile Systems, Applications, and Services*, 06-08 June 2005.

[66] R. Whitmam, "How much power does a black interface really save on AMOLED displays?", *GreenBot*, 21 October 2014. http://www.greenbot.com/article/2834583/how-much-power-does-a-black-interface-really-save-on-amoled-displays.html

[67] J. Williams and L. Curtis, "Green: The New Computing Coat of Arms?", *IT Professional*, vol.10, no.1, pp.12-16, January 2008.

[68] L. Benini and G. Micheli, "System-level power optimization: techniques and tools", in *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol.5 no.2, pp.115-192, April 2000.

[69] W. W. Eckerson, “Three Tier Client/Server Architecture: Achieving Scalability, Performance, and Efficiency in Client Server Applications”, in *Open Information Systems 10*, January 1995.

[70] K. C. Barr and K. Asanović, “Energy-aware lossless data compression”, in *ACM Transactions on Computer Systems (TOCS),* vol.24 no.3, pp.250-291, August 2006.

[71] C. E. Jones , K. M. Sivalingam , P. Agrawal and J. C. Chen, “A Survey of Energy Efficient Network Protocols for Wireless Networks”, *Wireless Networks*, vol.7 no.4, pp. 343-358, 1 August 2001.

[72] G. Mathur, P. Desnoyers , D. Ganesan and P. Shenoy, “Capsule: an energy-optimized object storage system for memory-constrained sensor devices”, in *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems*, October 31-November 03, 2006.

[73] J. Ousterhout , P. Agrawal , D. Erickson , C. Kozyrakis , J. Leverich , D. Mazières, S. Mitra , A. Narayanan , G. Parulkar , M. Rosenblum , S. M. Rumble , E. Stratmann and R. Stutsman, “The case for RAMClouds: scalable high-performance storage entirely in DRAM”, *ACM SIGOPS Operating Systems Review*, vol.43, no.4, January 2010.

[74] C. H. Chang, “A Low-Cost Green IT Design and Application of VHSP Based on Virtualization Technology,” in *International Conference on Computational Science and Engineering*, vol.3, pp.225-230, 29-31 August 2009.

[75] K. Lange, “Identifying Shades of Green: The SPECpower Benchmarks”, *Computer*, p.95-97, March 2009.

[76] “EnergyBench Benchmark Software”, *Industry-Standard Benchmarks for Embedded Systems*, 22 January 2016. http://www.eembc.org/benchmark/power_sl.php

[77] “EFM32 Wonder Gecko Development Kit”, *Silicon Lab*, 22 January 2016. https://www.silabs.com/products/mcu/lowpower/Pages/efm32wg-dk3850.aspx

[78] J. Flinn and M. Satyanarayanan, “PowerScope: A tool for profiling the energy usage of mobile applications”, in *Proceedings of the Second IEEE Workshop on Mobile Computer Systems and Applications*, pp.2, 25-26 February 1999.

[79] A. Kansal, F. Zhao, J. Liu, Nupur Kothari, and A. Bhattacharya, “Virtual Machine Power Metering and Provisioning” , in *ACM Symposium on Cloud Computing (SOCC)*, ACM., 10 June 2010.

[80] J. D. Vega, “Using the Intel® Power Gadget 3.0 API on Windows”, *Intel Developer Zone*, 7 January 2014. https://software.intel.com/en-us/blogs/2014/01/07/using-the-intel-power-gadget-30-api-on-windows

[81] A. Sinha and A. P. Chandrakasan, “JouleTrack: a web based tool for software energy profiling”, in *Proceedings of the 38th annual Design Automation Conference*, p.220-225, June 2001.

[82] J. Baliga, R. W. A. Ayre, K. Hinton, R. S. Tucker, “Green Cloud Computing: Balancing Energy in Processing, Storage and Transport,” in *Proceedings of the IEEE*, no.99, pp.1-19, 29 August 2010.

[83] H. Hoffmann, S. Sidiroglou, M. Carbin, S. Misailovic, A. Agarwal, and M. Rinard, “Power-Aware Computing with Dynamic Knobs”, *MIT Computer Science and Artificial Intelligence Laboratory Technical Report*, May 14, 2010.

[84] N. Amsel and B. Tomlinson, “Green tracker: a tool for estimating the energy consumption of software,” in *Proceedings of the 28th of the international conference extended abstracts on Human factors in computing systems*, 10–15April 2010.

[85] S. Roberts, S. Wright, D. Lecomber, C. January, J. Byrd, X. Oró and S. Jarvis, “POSE: A Mathematical and Visual Modelling Tool to Guide Energy Aware Code Optimisation”, in *6th International Green and Sustainable Computing Conference*, 14-16 December 2015.

[86] S. Murugesan S, “Harnessing Green IT: Principles and Practices”, *IT professional*, vol. 10, no. 1, pp. 24-33, January 2008.

[87] A. Y. Zomaya, and Y. C. Lee, “Energy Efficient Distributed Computing Systems”, *Wiley*, vol.88, 2012.

[88] G. Fettweis and E. Zimmermann, “ICT energy consumption—trends and challenges”, in *Proceedings of the 11th International Symposium on Wireless Personal Multimedia Communications*, vol. 2, no. 4, p. 6, September 2008.

[89] “America's Data Centers Consuming and Wasting Growing Amounts of Energy”, *Natural Resources Defense Council (NRDC)*, 6 February 2015. http://www.nrdc.org/energy/data-center-efficiency-assessment.asp

[90] “Energy-Efficient Computing Systems, dynamic adaptation of Quality of Service and approximate computing”, *Europa*, 27 November 2014. http://ec.europa.eu/newsroom/dae/document.cfm?doc_id=8756

[91] “Approximate computing' improves efficiency, saves energy”, *Purdue University*, 17 December 2013. http://www.purdue.edu/newsroom/releases/2013/Q4/approximate-computing-improves-efficiency,-saves-energy.html

[92] “Ericsson energy and carbon report”, *Ericsson*, November 2014. http://www.ericsson.com/res/docs/2014/ericsson-energy-and-carbon-report.pdf

[93] W. M. Adams, “The future of sustainability. Re-thinking environment and development in the twenty-first century: technical report”, *IUCN*, 2006.

[94] B. Penzenstadler, V. Bauer, C. Calero and X. Franch, “Sustainability in software engineering: A systematic literature review”, in *Proceedings of the International Conference on Evaluation and Assessment in Software Engineering*, pp. 32–4, 14-15 May 2012.

[95] F. Fakhar, R. U. Rasool and O. Malik. “Distributed Green Compiler”, in *Proceedings of the 2011 Fourth IEEE International Conference on Utility and Cloud Computing (UCC '11)*, pp.421-426, 2011.

[96] G. Calandrini, A. Gardel, I. Bravo, P. Revenga, J. L.Lázaro and F. J. Toledo-Moreo, “Power measurement methods for energy efficient applications”, *Sensors*, 2013.

[97] “The Green Grid Data Center Power Efficiency Metrics: PUE and DCiE”, *Metrics & Measurments White Paper*, The Green Grid, retrieved on 23 January 2016. http://www.thegreengrid.org/sitecore/content/Global/Content/white-papers/The-Green-Grid-Data-Center-Power-Efficiency-Metrics-PUE-and-DCiE.aspx

[98] H. Chen and W. Shi, “Power measuring and profiling: the state of art,” Handbook of Energy-Aware and Green Computing, *Chapman and Hall/CRC, Boca Raton*, 2012.

[99] L. Ardito, “Energy-aware Software”, *Dissertation, Tese de Doutorado, Politecnico di Torino*, 2014.

[100] P. Zhou, B. W. Ang and K. L. Poh, “Measuring environmental performance under different environmental DEA technologies”, *Energy Economics*, vol.30, no.1, pp.1-14, 2008.

[101] O. Soysal, S. Ayyorgun and M. Demirbas, “PowerNap: An energy efficient MAC layer for random routing in wireless sensor networks”, in *2011 8th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON)*, pp. 10-18, June 2011.

[102] S. Abdulsalam, Z. Zong, Q. Gu and M. Qiu, "Using the Greenup, Powerup and Speedup Metrics to Evaluate Software Energy Efficiency," *6th International Green and Sustainable Computing*, 14-16 December 2015.

[103] X. Zhang, "Operating system-level on-chip resource management in the multicore era", *Doctoral dissertation*, University of Rochester, 2010.

[104] D. Lyon, "The Discrete Fourier Transform, Part 6: Cross-Correlation", *Journal of Object Technology*, pp. 17-22, 2010.

[105] "Sustainability," *English Dictionary*, Collins. http://www.collinsdictionary.com/dictionary/english/sustainability

[106] "Sustainable Development", *International Institute for Sustainable Development*, 23 January 2016. https://www.iisd.org/topic/sustainable-development

[107] C. Calero and M. Piattini, "Green in Software Engineering", *Springer*, 3 April, 2015.

[108] "Software Engineering for Sustainability", *Institute for Software Research, University of California, Irvine*, retrieved on 23 January 2016. https://isr.uci.edu/content/software-engineering-sustainability

[109] "Defining Life Cycle Assessment (LCA)", *US Environmental Protection Agency*, 17 October 2010. http://www.gdrc.org/uem/lca/lca-define.html

[110] S. Wang, H. Chen and W. Shi, "SPAN: A software power analyzer for multicore computer systems", *Sustainable Computing: Informatics and Systems*, vol. 1, no. 1, pp. 23-34, 2011.

[111] N. Amsel, Z. Ibrahim, A. Malik and B. Tomlinson, "Toward sustainable software engineering: NIER track", in *2011 IEEE 33rd International Conference on Software Engineering (ICSE)*, pp. 976-979, 2011.

[112] C. Canal, J. M. Murillo and P. Poizat, "Software Adaptation", *L'Objet*, vol. 12, no. 1, pp. 9–31, 2006.

[113] P. Oreizy, N. Medvidovic and R. N. Taylor, "Runtime software adaptation: framework, approaches, and styles", in *Companion of the 30th International Conference on Software Engineering*, pp. 899-910, ACM, May 2008.

[114] H. Hoffmann, "JouleGuard: energy guarantees for approximate applications", in *Proceedings of the 25th Symposium on Operating Systems Principles*, pp. 198-214, ACM, October 2015.

[115] F. Alessi, P. Thoman, G. Georgakoudis, T. Fahringer and D. S. Nikolopoulos, "Application-level Energy Awareness for OpenMP", in *OpenMP: Heterogenous Execution and Data Movements*, *Springer International Publishing*, pp. 219-232, 2015.

[116] G. Valetto and G. Kaiser, "A case study in software adaptation", in *Proceedings of the First Workshop on Self-Healing Systems*, pp. 73-78, ACM, November 2002.

[117] S. D. Ramchurn, P. Vytelingum, A. Rogers and N. R. Jennings, "Putting the 'Smarts' into the Smart Grid: A Grand Challenge for Artificial Intelligence," in *Communications of the ACM*, vol. 55, vo. 4, pp. 86-97, April 2012.

[118] C. Reinisch, M. J. Kofler, and W. Kastner, "ThinkHome: A smart home as digital ecosystem", in *2010 4th IEEE International Conference on Digital Ecosystems and Technologies (DEST)*, pp. 256-261, IEEE, April 2010.

[119] T. A. Nguyen and M. Aiello, "Energy intelligent buildings based on user activity: A survey", *Energy and Buildings*, vol. 56, pp. 244-257, 2013.

[120] A. Vojdani, "Smart integration," *Power and Energy Magazine*, IEEE, vol.6, no.6, pp. 71-79, 2008.

[121] M. Lorenz, L. Wehmeyer, and T. Dr¨ager T. "Energy aware Compilation for DSPs with SIMD instructions", in *Proceedings of Languages, compilers and tools for embedded systems: software and compilers for embedded systems LCTES/SCOPES '02*, pp. 94-101, 2002.

[122] N. Z. Azzemi, "A Multiobjective Evolutionary Approach for Constrained Joint Source Code Optimization", in *Proceedings of ISCA 19th International Conference on Computer Application in Industry (CAINE 2006)*, Las Vegas, Nevada, USA, pp. 175–180, 2006.

[123] I. Meedeniya, B. Buhnova, A. Aleti, and L. Grunske, "Architecture-Driven Reliability and Energy Optimization for Complex Embedded Systems", *in Proceedings of the International Conference on the Quality of Software Architectures (QoSA'10)*, *Springer Verlag*, pp. 52–67, 2010.

[124] J. P. Katoen, M. Khattri, and I. S. Zapreev, "A Markov reward model checker", in *Proceedings of the QEST:International Conference on the Quantitative Evaluation of Systems*, IEEE Computer Society, pp. 243–244, 2005.

[125] F. Marcelloni, and M. Vecchio, "Enabling energy-efficient and lossy-aware data compression in wireless sensor networks by multi-objective evolutionary optimization", *Information Sciences*, vol. 180, pp. 1924-1941, 2010.

[126] S. Liu, R. Srivastava, C. E. Koksal, and P. Sinha, "Pushback: A hidden Markov model based scheme for energy efficient data transmission in sensor networks", *Ad Hoc Netw*, pp. 973-986, July 2009.

[127] C. Mandery, "Distributed N-Gram Language Models: Application of Large Models to Automatic Speech Recognition", Doctoral Dissertation, *Informatics Institute*, 2011.

[128] "Internet of Things' connected devices to almost triple to over 38 billion units by 2020", *Juniper Research*, retrieved on 27 January 2016. http://www.juniperresearch.com/press/press-releases/iot-connected-devices-to-triple-to-38-bn-by-2020

[129] L. Hardesty, "Energy-friendly chip can perform powerful artificial-intelligence tasks", *MIT News on Campus and Around the World*, 3 February 2016. http://news.mit.edu/2016/neural-chip-artificial-intelligence-mobile-devices-0203

[130] B. Steigerwald, R. Chabukswar, K. Krishnan, and J. D. Vega, "Creating energy efficient software", *Intel White Paper*, 2008.

[131] W. E. Deming, "The New Economics for Industry, Government", *The MIT Press*, 2nd edition, p. 35, 2000.

[132] J. Gray, "Sort Benchmark Homepage", retrieved on March 8, 2016. http://www.sortbenchmark.org

[133] J. Zhang, A. Musa and W. Le, "A comparison of energy bugs for smartphone platforms", in *1st International Workshop on the Engineering of Mobile-Enabled Systems (MOBS)*, pp. 25-30, IEEE, 2013.

[134] "Trepn Power Profiler", *Qualcomm Developer Network*, retrieved on March 2016. https://developer.qualcomm.com/software/trepn-power-profiler

[135] M. Gordon, L. Zhang, B. Tiwana, R. P. Dick, Z. M. Mao and L. Yang, "PowerTutor: a power monitor for android-based mobile platforms", *University of Michigan*, retrieved on March 2016. http://ziyang.eecs.umich.edu/projects/powertutor/

[136] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang, "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, ACM, October 2010.

[137] “Measure Energy Impact with Instruments”, *iOS Developer Library*, retrieved on March 2016. https://developer.apple.com/library/ios/documentation/Performance/Conceptual/EnergyGuide-iOS/MonitorEnergyWithInstruments.html

[138] W. H. Wang and V. De, “Intel Labs ISSCC 2014 Highlights Energy Efficiency Research”, *Intel Labs Energy Efficiency Research*, retrieved on March 2016. http://www.intel.com/newsroom/kits/isscc/2014/pdfs/Intel_Labs_Energy_Efficiency_Research.pdf

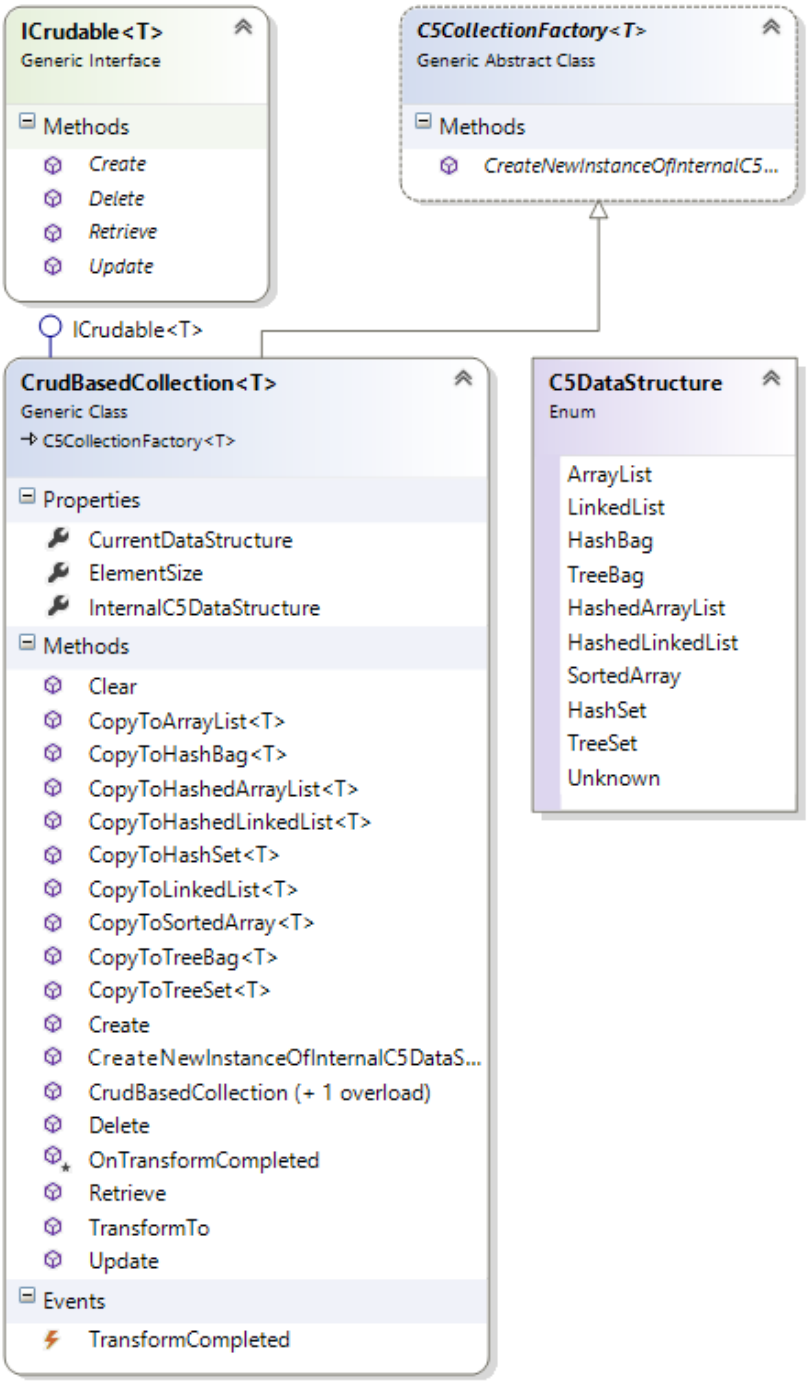
[139] “CodeXL – Powerful debugging, profiling & analysis”, *AMD Developer Central*, retrieved on March 2016. http://developer.amd.com/tools-and-sdks/opencl-zone/codexl/

[140] C. Yoon, D. Kim, W. Jung, C. Kang and H. Cha, “Appscope: Application energy metering framework for android smartphone using kernel activity monitoring”, in *Presented as part of the 2012 USENIX Annual Technical Conference (USENIX ATC 12)*, pp. 387-400, 2012.

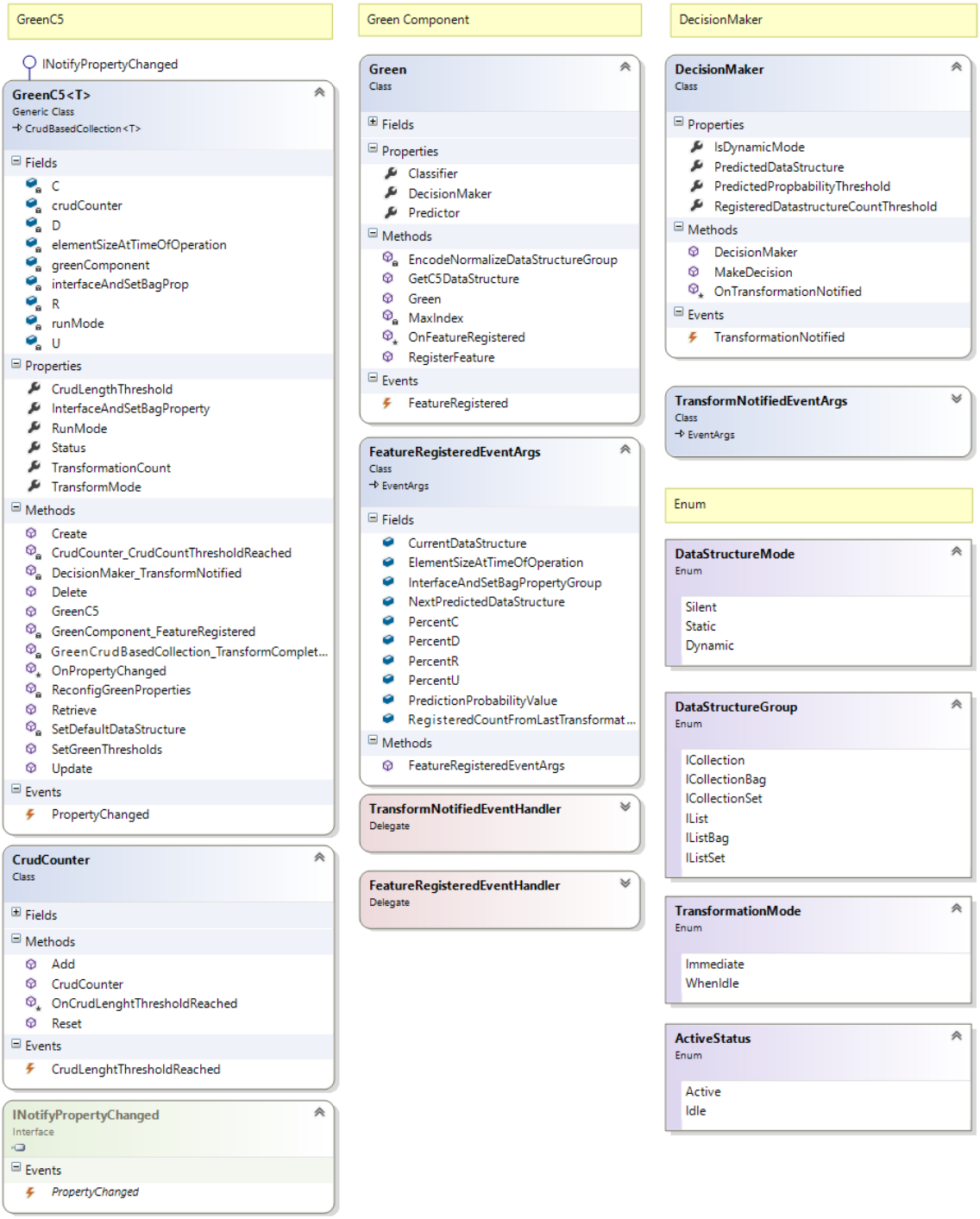
[141] J. Michanan, “DUGreen Project”, *GitHub*, retrieved on April 2016, https://github.com/michanaj/DUGreen-Project.

APPENDICES

Appendix A: CRUD-Based C5 Collection Class Diagram



Appendix B: GreenC5 Class Diagram



Appendix C: Learning Algorithms

Algorithm C.1: Offline Supervised Learning and Accuracy Testing of the ANN Classifier
Input: Training sets, additional test sets, program validation sets
Output: Weights and biases (a priori model), Gaussian Normalization Values, Accuracy Results and X and Y decoded value dictionaries

```
//procedure accuracy testing for the testData using winner-takes all strategy
procedure Accuracy(testData)
    begin
        // Percentage correct using a winner-takes-all method
        numCorrect := 0
        numWrong := 0
        for each row in testData do
                xValues := the first numInput items of row// Get x-values, first 10
                tValues:= the last numOutput items of row// Get target values (actual y-value vector)
                yValues := NN.ComputeOutputs(xValues) //get the computed y-value vector from NN
                maxIndex := Helpers.MaxIndex(yValues) // Which cell in yValues has the largest value?
                if tValues[maxIndex] =1.0  then
                        numCorrect := numCorrect + 1
                else
                        numWrong := numWrong+ 1
                end if
        end for
        return numCorrect  / (numCorrect + numWrong);
    end
procedure Main();
    begin
        //1. Read data from dataset files
        trainingSet := read Training set from a file
        testSets := read Additional Test sets and program validation sets from files

        //2. Encode non-numeric Y value of datasets to numeric values using 1-of-N dummy encoding
        //and save the encoded dictionary for Y values
        Helpers.EncodeFile(trainingSet, 6, "dummy", out encodedYDictionary)
        foreach set in testSets do
            Helpers.EncodeFile(set, 6, "dummy", encodedYDictionary)//column 6 in the training set
        //(0based)
        end for

        //3. Encoded non-numeric X values of all datasets to numeric values using 1-of-(N-1) effects
        //encoding and save the encoded dictionary for X values
        Helpers.EncodeFile(trainingSet, 0, "effects", out encodedXDictionary)//column 0 in the
        //training set
        foreach set in testSets do
            Helpers.EncodeFile(set, 0, "effects", encodedXDictionary)//col 6 in the training set
        //(0based)
        end for

        //4. Normalize numeric X values of datasets using Gaussian Normalization (normalize number
        //to between -10 and 10) on columns 5,6,7,8,9 (number columns got expanded after the
```

```
        //encoding above)
        Helpers.GaussNormal(trainingSet, 5, out gaussStdvX2_ElmSize, out gaussMeanX2_ElmSize)
        Helpers.GaussNormal(trainingSet, 6, out gaussStdvX3_C, out gaussMeanX3_C)
        Helpers.GaussNormal(trainingSet, 7, out gaussStdvX4_R, out gaussMeanX4_R)
        Helpers.GaussNormal(trainingSet, 8, out gaussStdvX5_U, out gaussMeanX5_U)
        Helpers.GaussNormal(trainingSet, 9, out gaussStdvX6_D, out gaussMeanX6_D)
        foreach set in testSets do
                Helpers.GaussNormal(set, 5, gaussStdvX2_ElmSize, gaussMeanX2_ElmSize)
                        Helpers.GaussNormal(set, 6, gaussStdvX3_C, gaussMeanX3_C)
                        Helpers.GaussNormal(set, 7, gaussStdvX4_R, gaussMeanX4_R)
                        Helpers.GaussNormal(set, 8, gaussStdvX5_U, gaussMeanX5_U)
                        Helpers.GaussNormal(set, 9, gaussStdvX6_D, gaussMeanX6_D)
        end for

        //5. Split training set to 80% train set and 20% test set (Hold-Out validation method)
        // shuffle the order of training set rows randomly and split them to two sets
        Helpers.MakeTrainTest(traininSet, out trainData, out testData);

        //6. Create a 10-15-9 Neural Network
        numInput := 10
        numHidden := 15
        numOutput := 9
        NeuralNetwork NN := new NeuralNetwork(numInput, numHidden, numOutput);

        //7. Start training the NN on the trainData  (80% of the training set) with these parameters
        maxEpochs := 3000
        learnRate := 0.02
        momentum: = 0.01
        stopError:= 0.04
        NN.Train(trainData, maxEpochs, learnRate, momentum, stopError)

        //8. Get the weights and biases (contains both)
        weights := NN.GetWeights()

        //9. Accuracy testing on the testData (20% of the Training set)
        testAccuracy := Accuracy(testData)

        //10. Accuracy testing on the remaining additional test sets and program validation sets
        foreach set in testSets do
                testAccuracies := Accuracy(set)
        end for

        //11. Save all the weights and biases, Gaussian Normalization means and standard deviation
        //values, accuracy results and others info to a file
        results := { numInput, numHidden, numOutput, encodedXDictionary , encodedYDictionary
        ,weights, gaussStdvs, gaussMeans, testAccuracy, testAccuracies}
        Helpers.SaveToFile(results)
end
```

Algorithm C.2: Online learning, classifying and predicting the most energy efficient C5 data structure and prediction accuracy testing
Input: *NNValues*, Additional test sets, program validation sets, program CRUD sequences
Output: Accuracy results

```
//procedure to compute accuracy for the predicted DS sequence with the actual one
procedure ComputeAccuracy(predictedDSSeqList, actualDSSeqList)
    begin
        numCorrect := 0
        totalCount := actualDSSeqList.Count
        index := 0
        for each ds in actualDSSeqList do
                if predictedDSSeqList[index] = ds then
                        numCorrect := numCorrect + 1
                end if
                index := index+ 1
        end for
        return numCorrect  / totalCount
    end
procedure Main()
    begin
        //1. Read all parameter values for the NN classifier
        NNResults := read NN results from the result file
        numInput := NNResults.numInput,  numHidden:= NNResults.numHidden
        numOutput:= NNResults.numOutput
        weights:= NNResults.weights
        gaussMeanX2 := NNResults.gaussMeanX2,          gaussStdvX2:= NNResults.gaussStdvX2
        gaussMeanX3 := NNResults.gaussMeanX3,          gaussStdvX3:= NNResults.gaussStdvX2
        gaussMeanX4   := NNResults.gaussMeanX4,        gaussStdvX4:= NNResults.gaussStdvX2
        gaussMeanX5 := NNResults.gaussMeanX5,          gaussStdvX5:= NNResults.gaussStdvX2
        gaussMeanX6 := NNResults.gaussMeanX6,          gaussStdvX6:= NNResults.gaussStdvX2
        encodedXDictionary := NNResults.encodedYDictionary
        encodedYDictionary := NNResults.encodedYDictionary

        //2. Create a Neural Network with the same set of the tuned weights and biases
        NeuralNetwork NN := new NeuralNetwork(numInput, numHidden, numOutput);
        NN.SetWeights(weights)

        //3. Encoded the X1 (Interface) values to numeric data
        x1Interfaces:= {ICollecction, ICollectionBag, ICollectionSet, IList, IListBag, IListSet}
        x1EncodedValues := {}
        for each x1 in x1Interfaces do
                //this will encode x1 value to numeric values using 1-of-(N-1) effects encoding
                x1EncodedValue :=Helpers.Encoded(x1, "effects", encodedXDictionary)
                add x1EncodedValue to x1EncodedValues
        end for

        //4. Create a N-Gram predictor
        nValue := 2 //bi-gram
        NGramPredictor PREDICTOR = new NGramPredictor(nValue)

        //5. Start by reading program CRUD sequences from files (19 progCRUD files each contains
        //full CRUD sequences from start to end of the program)
```

```
    maxIteration := 5 //control how many times of each program sequence to be running
    iteration : = 0
    for each x1EncodedValue in x1EncodedValues do      //loop through each encoded Interface
       while iteration < maxIteration do
          for each programSeq in progCRUDSeqFiles do  //this will ignore the last remainder
                startSize :=0 //assume each data structure always start with size 0
                endSize := 0
                nGram := new List() //this is for storing N-gram sequence
                n-1Gram := new List() //this for storing N-1 sequence for predicting
                actualDSSeqList := read from program sequence file
                classifiedDSSeqList := new List() //to store the computed DS by the classifier
                predictedDSSeqList := new List() //to store the predicted DS sequence
                for each 10KSeq in programSeq do  //this will ignore the last remainder
                      xValuesNoX1 := translateToXValues(10KSeq, startSize , out endSize)
                      //normalize the X values
                      x2Normal:= (xValuesNoX1[0] - gaussMeanX2 )/ gaussStdvX2
                      x3Normal:= (xValuesNoX1[1] – gaussMeanX3)/ gaussStdvX3
                      x4Normal:= (xValuesNoX1[2] – gaussMeanX4)/ gaussStdvX4
                      x5Normal:= (xValuesNoX1[3] – gaussMeanX5)/ gaussStdvX5
                      x6Normal:= (xValuesNoX1[4] - gaussMeanX24)/ gaussStdvX4
                            xEncodedNormalizedValues :=
                            {x1EncodedValue} combines with { x2Normal, x3Normal,
                            x4Normal, x5Normal, x6Normal}
                      yValues := NN.ComputeOutputs(xEncodedNormalizedValues)
                      maxIndex := Helpers.MaxIndex(yValues)
                      computedDs :=  encodedYDictionary[maxIndex]
                      classifiedDSSeqList.Add(computedDs)
                      nGram := get the last nValue items in classifiedDSSeqList
                      n-1Gram := get the last nValue-1 items in classifiedDSSeqList
                      if nGram.Count = nValue then       //register the sequence to N-Gram
                            PREDICTOR.Register(nGram.ToArray());
                            nGram.Clear()
                      end if

                      if n-1Gram.Count = nValue-1 then
                            predictedDs := PREDICTOR.PredictNext(n-1Gram)
                            n-1Gram.Clear()
                      else
                            predictedDs := "Unknown"
                      end if
                      predictedDSSeqList.Add(predictedDs)
                      startSize:= endSize
                end for
                accuracy := ComputeAccuracy(predictedDSSeqList, actualDSSeqList)
          end for
          save accuracy result of each Interface and iteration to a file
       end while
    end for
end
```