TURUN
YLIOPISTO
UNIVERSITY
OF TURKU

SELF-AWARE RESOURCE MANAGEMENT IN EMBEDDED SYSTEMS

Elham Shamsa

TURUN YLIOPISTON JULKAISUJA – ANNALES UNIVERSITATIS TURKUENSIS
SARJA – SER. F OSA – TOM. 11 | TECHNICA – INFORMATICA | TURKU 2022

TURUN
YLIOPISTO
UNIVERSITY
OF TURKU

SELF-AWARE RESOURCE MANAGEMENT IN EMBEDDED SYSTEMS

Elham Shamsa

TURUN YLIOPISTON JULKAISUJA – ANNALES UNIVERSITATIS TURKUENSIS
SARJA – SER. F OSA – TOM. 11 | TECHNICA – INFORMATICA | TURKU 2022

University of Turku

Faculty of Technology
Department of Computing
Information and Communication Technology
Doctoral Programme in Technology (DPT)

Supervised by

Prof. Pasi Liljeberg
University of Turku

Assoc. Prof. Amir M. Rahmani
University of California Irvine

Senior Researcher Anil Kanduri
University of Turku

Reviewed by

Prof. Cristiana Bolchini
Politecnico di Milano

Assoc. Prof. Muhammad Shafique
New York University Abu Dhabi

Opponent

Prof. Jari Nurmi
Tampere University

ISBN 978-951-29-8907-2 (PRINT)
ISBN 978-951-29-8908-9 (PDF)
ISSN 2736-9390 (PRINT)
ISSN 2736-9684 (ONLINE)
Painosalama, Turku, Finland 2022

UNIVERSITY OF TURKU
Faculty of Technology
Department of Computing
Information and Communication Technology
SHAMSA, ELHAM: Self-Aware Resource Management in Embedded Systems
Doctoral dissertation, 144 pp.
Doctoral Programme in Technology (DPT)
May 2022

ABSTRACT

Resource management for modern embedded systems is challenging in the presence of dynamic workloads, limited energy and power budgets, and application and user requirements. These diverse and dynamic requirements often result in conflicting objectives that need to be handled by intelligent and self-aware resource management. State-of-the-art resource management approaches leverage offline and online machine learning techniques for handling such complexity. However, these approaches focus on fixed objectives, limiting their adaptability to dynamically evolving requirements at run-time.

In this dissertation, we first propose resource management approaches with fixed objectives for handling concurrent dynamic workload scenarios, mixed-sensitivity workloads, and user requirements and battery constraints. Then, we propose comprehensive self-aware resource management for handling multiple dynamic objectives at run-time. The proposed resource management approaches in this dissertation use machine learning techniques for offline modeling and online controlling. In each resource management approach, we consider a dynamic set of requirements that had not been considered in the state-of-the-art approaches and improve the self-awareness of resource management by learning applications characteristics, users' habits, and battery patterns. We characterize the applications by offline data collection for handling the conflicting requirements of multiple concurrent applications. Further, we consider user's activities and battery patterns for user and battery-aware resource management. Finally, we propose a comprehensive resource management approach which considers dynamic variation in embedded systems and formulate a goal for resource management based on that.

The approaches presented in this dissertation focus on dynamic variation in the embedded systems and responding to the variation efficiently. The approaches consider minimizing energy consumption, satisfying performance requirements of the applications, respecting power constraints, satisfying user requirements, and maximizing battery cycle life. Each resource management approach is evaluated and compared against the relevant state-of-the-art resource management frameworks.

KEYWORDS: On-chip resource management, Embedded systems, Self-aware systems

TURUN YLIOPISTO
Teknillinen tiedekunta
Tietotekniikan laitos
Tieto- ja viestintätekniikka
SHAMSA, ELHAM: Self-Aware Resource Management in Embedded Systems
Väitöskirja, 144 s.
Teknologian tohtoriohjelma (DPT)
Toukokuu 2022

TIIVISTELMÄ

Nykyaikaisten sulautettujen järjestelmien resurssienhallinta on haastavaa johtuen dynaamisista työkuormista, rajallisista energia- ja tehobudjetteista sekä sovellus- ja käyttäjävaatimuksista. Monimuotoiset ja dynaamiset vaatimukset johtavat usein ristiriitaisiin vaatimuksiin, jotka älykkään ja itsetietoisen resurssienhallinnan on ratkaistava. Nykyaikaiset lähestymistavat resurssienhallintaan hyödyntävät koneoppimistekniikoita mutta keskittyvät staattisiin tavoitteisiin mikä rajoittaa niiden käyttöä ja ennen kaikkea soveltamista dynaamisiin vaatimuksiin käytön aikana.

Tässä väitöskirjassa esitetään ensin staattisten tavoitteiden resurssienhallintamenetelmiä samanaikaisten dynaamisten työkuormaskenaarioiden, sekaherkkien työkuormien sekä käyttäjien vaatimusten ja tehonkulutusrajoitusten ratkaisemiseksi. Tämän jälkeen esitetään itsetietoinen resurssienhallintamenetelmä useiden dynaamisten tavoitteiden käsittelyyn ajon aikana. Tässä väitöskirjassa ehdotetut resurssienhallintamenetelmät käyttävät koneoppimistekniikoita mallinnukseen ja järjestelmän ohjaukseen. Resurssienhallintamenetelmässä otetaan huomioon dynaamiset vaatimukset joita ei ole otettu aikaisemmin tässä mittakaavassa huomioon. Tätä kautta resurssienhallinta paranee. Sovellukset luokitellaan useiden samanaikaisten sovellusten ristiriitaisten vaatimusten käsittelemiseksi ottaen huomioon käyttäjän toiminnan sekä tehonkulutuksen.

Väitöskirjan lopuksi esitetään kattava resurssienhallintamenetelmä jossa otetaan huomioon sulautettujen järjestelmien dynaaminen vaihtelu ja määritellään resurssienhallinnan tavoite tämän perusteella. Tässä väitöskirjassa esitetyissä menetelmissä tarkastellaan energiankulutuksen minimointia, sovellusten suorituskykyvaatimusten täyttämistä , tehorajoitusten toteutumista, käyttäjien vaatimusten täyttämistä ja akun käyttöiän maksimointia. Jokaista resurssienhallintamenetelmää arvioidaan ja verrataan relevantteihin aikaisempiin resurssienhallintamenetelmiin.

ASIASANAT: resursihallinta, sulautetut järjestelmät, itsetietoiset järjestelmät

Acknowledgements

All my academic growth and achievements during my Ph.D. cannot be made without supports of many people. As my Ph.D. journey is near to end, I would like to take this opportunity to thank these people who inspired me during this phase of my life.

Firstly, I would like to express my sincere gratitude to my supervisors Prof. Pasi Liljeberg, Assoc. Prof. Amir M. Rahmani, and Dr. Anil Kanduri for the continuous support of my Ph.D. study. I want to especially thank Dr. Anil Kanduri for his insightful comments and encouragement which intended me to widen my research from various perspectives. I want to additionally thank Prof. Liljeberg and Assoc. Prof. Rahmani for helping me to cope with the problems I encountered during my Ph.D.

Besides my supervisors, I would like to thank all the collaborators and co-authors Prof. Nikil Dutt, Prof. Axel Jantsch, Prof. Samarjit Chakraborty, Asst. Prof. Nima taherinejad, and Dr. Alma Pröbstl. I want to especially thank Asst. Prof. Nima taherinejad for inspiring me to grow my research area and his scientific support.

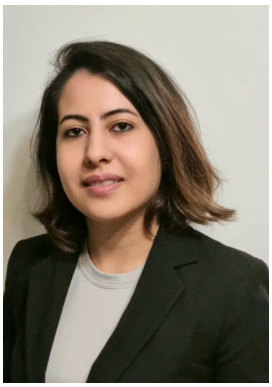
This research was supported by the University of Turku Graduate school, Nokia Foundation, and TES Foundation's grants. I want to thank these organizations for supporting young researchers.

I want to thank my friends for their advise and moral support. Their friendly accompaniment has made the difficult steps of my doctoral journey easy to move on. I would like to especially thank Dr. Fahime Farahnakian, Dr. Mohammad Hashem Haghbayan, Dr. Iman Azimi, Mr. Arman Anzanpour, and Mrs. Fatemeh Sarhadi.

I especially thank my parents, who have always been supporting me in every stage of life including my undergraduate and postgraduate studies. I also thank my brother and my sister for their love and encouragement.

Lastly and most importantly, endless thanks to my beloved husband, Sadjad for backing me up always and unconditionally. It Could not have been possible to reach this step without Sadjad's support and encouragement.

09.05.2022
Elham Shamsa



ELHAM SHAMSA

Elham Shamsa received the B.Sc. degree in computer engineering-hardware in 2012, and the M.Sc. degree in computer architecture in 2014, from Amirkabir University of Technology, Tehran, Iran. Her current research interests include system on-chip resource management, multi- objective resource management techniques, high-performance energy-efficient architectures, and self-aware systems.

Table of Contents

Abbreviations

AB	Application Bias
BCL	Battery Cycle Life
CB	Configuration Bias
CGDA	Configurable Goal Driven Autonomy
CPU	central Processing Unit
DoP	Degree of Parallelism
DVFS	Dynamic Voltage and Frequency Scaling
FoM	Figure of Merit
GDA	Goal Driven Autonomy
GPU	Graphics Processing Unit
HB	Heart Beat
HMP	Heterogeneous Multi-core Platform
HP	High Performance
IL	Imitation Learning
IPE	Instruction Per Epoch
IPS	Instruction Per Second
IT	Information Technology
LMS	Least Mean Square
LP	Low Power
LR	Linear Regression
QoE	Quality of Experience

QoS	Quality of Service
RL	Reinforcement Learning
RP	Relative Progress rate
RT	Regression Tree
RTM	Something something
SoC	System on Chip
SOC	State of Charge
TDP	Thermal Design Power
UBAR	User and Battery Aware Resource management
WB	Weighted Bias

List of Original Publications

This dissertation is based on the following original publications, which are referred to in the text by their Roman numerals:

I **Elham Shamsa**, Anil Kanduri, Pasi Liljeberg, and Amir M. Rahmani. Concurrent Application Bias Scheduling for Energy Efficiency of Heterogeneous Multi-Core platforms. IEEE Transactions on Computers (TC), 2021.

II **Elham Shamsa**, Anil Kanduri, Amir M. Rahmani, and Pasi Liljeberg. Energy-Performance Co-Management of Mixed-Sensitivity Workloads on Heterogeneous Multi-core Systems. In Proceedings of the 26th Asia and South Pacific Design Automation Conference (ASPDAC), 2021.

III **Elham Shamsa**, Alma Pröbstl, Nima TaheriNejad, Anil Kanduri, Samarjit Chakraborty, Amir M. Rahmani, and Pasi Liljeberg. UBAR: User- and Battery-aware Resource Management for Smartphones. ACM Transactions on Embedded Computing Systems (TECS), 2021.

IV **Elham Shamsa**, Anil Kanduri, Nima TaheriNejad, Alma Pröbstl, Samarjit Chakraborty, Amir M. Rahmani, and Pasi Liljeberg. User-centric Resource Management for Embedded Multi-core Processors. In 33rd International Conference on VLSI Design and 19th International Conference on Embedded Systems (VLSID), 2020.

V **Elham Shamsa**, Anil Kanduri, Amir M. Rahmani, Pasi Liljeberg, Axel Jantsch, and Nikil Dutt. Goal-Driven Autonomy for Efficient On-chip Resource Management: Transforming Objectives to Goals. In Design, Automation & Test in Europe Conference & Exhibition (DATE), 2019.

1 Introduction

Continuous advances in embedded devices such as smartwatches, smartphones, domestic appliances, fitness trackers, and healthcare monitoring devices increase the complexity of resource management in such systems. Embedded systems have limitations on energy budget, number and type of processor cores, operating frequency, and thermal safety ranges [1; 2]. Advanced embedded systems consume high amount of power and energy, which may raise the system's temperature and harm the device. In addition, such smart devices should be capable of handling concurrent and unknown applications and satisfy their performance requirements at run-time [3; 4]. The applications' requirements may be conflicting with the system's constraints. For example, enhancing the performance of an application requires a higher frequency level or number of cores, increasing the power and temperature of the device. Such increase may cause power threshold violation and harm the device [5; 6]. Further, concurrent applications may compete for resources. Handling such competition, as well as conflicting requirements, and constraints need intelligent resource management [5; 7]. In this dissertation, we first introduce and discuss various resource management challenges and then present the solutions for addressing these challenges. Finally, we provide the evaluations of the intelligent resource management approaches based on our original publications.

This chapter presents an overview of challenges and research problems in resource management for heterogeneous embedded systems as well as research objectives and possible methods for tackling those challenges. In addition, it presents the thesis contributions and outlines. This chapter consists of four sections: (i) resource management challenges in embedded systems, (ii) aim and objectives of this research, (iii) contributions, and (iv) organization.

1.1 Resource Management Challenges in Embedded Systems

In embedded systems, there are entities such as i) user, ii) applications, iii) processors, and iv) battery, which affect the resource management decisions at run-time. Such entities issue their requirements and dynamically generate various objectives. Since the requirements are issued from different sources, they may be conflicting, orthogonal, or overlapping. Such requirements can change at any given time and lead

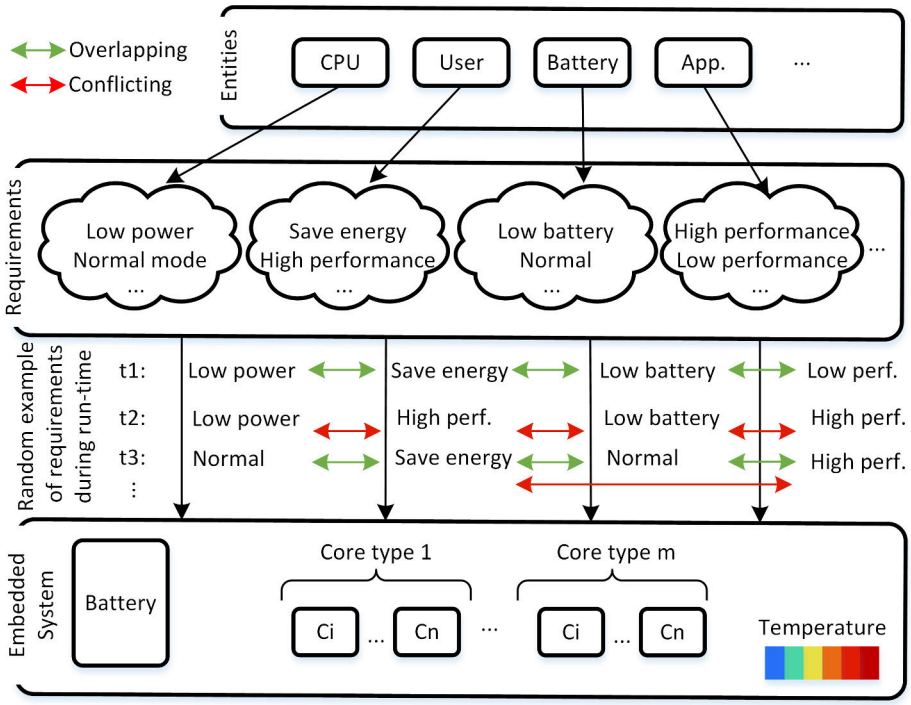


Figure 1. Various entities, requirements and objectives in the embedded systems, which change at run-time.

to new objectives [5]. Considering such variations as well as resource constraints jointly at run-time leads to a complex problem of resource allocation. A resource allocation strategy should consider (i) energy budget i.e., battery, (ii) thermal budget, (iii) number of processors which can be assigned to each application, and (iv) the type of processors i.e., big or LITTLE in heterogeneous processors. Figure 1 shows the various entities, their requirements, and objectives, which are generated dynamically at run-time. The figure shows the series of requirements that can be issued at any given time from various entities. Such requirements may be overlapping (shown by green arrows) or conflicting (shown by red arrows). Resource management framework must intelligently assign the limited resources in order to satisfy conflicting requirements of different entities [5; 8]. In this dissertation, we consider various conflicting objectives at run-time and present intelligent frameworks for efficient resource management.

Increasing performance and saving power are often two conflicting requirements, which are competing for resources. Providing high performance for applications would increase the power consumption, whereas scaling down the resources for power saving degrades the performance [9]. Furthermore, the execution of multiple concurrent applications, which can be compute- or memory- intensive, creates a

new competition between applications for consuming the resources. Prioritizing such applications based on their characteristics and requirements is one of the other challenges in resource allocation. We discuss such challenges in more detail in Chapters 3-6 and present resource management approaches for handling the conflicting objectives and competing entities.

1.2 Aims and Objectives of the research

The aim of this research is to provide a self-aware resource management framework for satisfying the various entities in embedded systems. Considering the conflicting requirements as well as limited resources and constraints in the embedded systems make resource management challenging. A self-aware resource management framework can monitor the requirements and available resources at any given time and optimize the resource consumption while satisfying the requirements. The objective of this research is to provide such self-awareness to address the challenges of on-chip resource management. Self-aware resource management needs to i) learn applications' characteristics to develop application-awareness, ii) consider users' activity to develop user-awareness, and iii) consider system dynamics to be system-aware. Therefore, the objectives of this research are to develop i) application-aware resource management, ii) user-aware resource management, iii) system-aware resource management, and finally iv) self-aware resource management. The details of each of the aforementioned objectives are provided in the following.

1. **Application-aware resource management**. The sub-objectives which develop the application awareness are:

    - *Characterizing the applications.* Various types of applications can execute on embedded systems. An application can be i) compute or memory-intensive, ii) single or multi-threaded, and have iii) shorter or longer execution time. Such characteristics can be collected by measuring run-time variables such as power consumption, CPU utilization, and Instruction Per Second (IPS) for an application when it is run in the various configurations of an embedded system. By collecting such data, we can characterize applications and design predictive models for guiding resource management's decisions.
    - *Categorizing applications based on their requirements.* The applications can have various requirements. Some applications are latency-sensitive and the others are throughput-driven. The applications which require a certain fixed level of performance within a strict deadline are latency-sensitive. Other class of applications that do not have any strict latency constraints, but require a higher overall performance are throughput-

driven applications. By categorizing the applications based on such requirements, resource management can prioritize the applications in an optimized manner.

- *Prioritizing the applications based on their characteristics and their types.* The outputs of the two above objectives guide the resource management policy to prioritize the applications at run-time. Then, the resource management policy allocate the available resources to satisfy the application's requirements while optimizing the power and energy consumption.

2. **User-aware resource management.** The user awareness can be obtained by:

   - *Profiling user's activities for guiding the resource management decisions.* Various users can have different habits of plug-in and plug-out for their battery-powered devices. Profiling the user's plug-in and plug-out events and using the profiled data for predicting the user's activities can guide the resource allocation's decisions.
   - *Handling the conflicts between system's and user's requirements.* Another aim of this research is handling the conflicting requirements. User's and system's requirements can be conflicting at run-time, thus there is a need for a controller that considers such conflicts and decides for an optimized resource allocation. Machine learning methods could handle such conflicts by evaluating various resource allocation decisions and learning the optimized actions.

3. **System-aware resource management.** System-awareness can be provided by considering the system constraints such as power budget and temperature threshold at run-time. Monitoring the power and temperature of the system at run-time and predicting violation of them based on the system's thresholds is system awareness.

4. **Comprehensive self-aware resource management.** The main goal of this research is to provide a comprehensive self-aware resource management framework for embedded systems. The sub-objectives of this goal are:

   - *Combining application awareness, user awareness, and system awareness for comprehensive resource management.* A comprehensive self-aware resource management can be obtained by the combination of the application, user, and system awareness.
   - *Learning the best action for resource management in response to the system, workload, and user variation at run-time.* Leveraging the machine learning methods can improve the self-awareness of a system. Thus, various machine learning techniques are used in this research to learn and

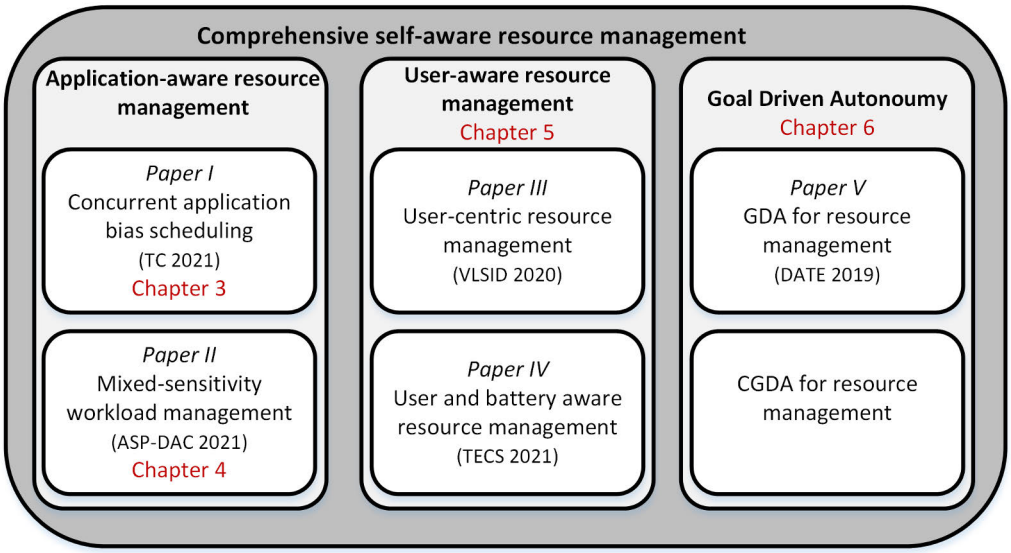


Figure 2. Illustration of research contributions and paper cohesion.

predict optimized resource allocation decisions at run-time in response to the system, workload, and user variations. As such variations generate different objectives at run-time, another aim of this research is to formulate the objectives and transform those into final goals for resource management.

1.3 Contributions

The research contributions, the corresponding publication and research objectives, and the chapters in which those are discussed are illustrated in Figure 2. Further, contributions of this research are summarized in the following.

- **Bias Scheduling:** This dissertation proposes a bias scheduling strategy for handling conflicting requirements of concurrent applications in heterogeneous multi-core platforms. The bias scheduling strategy leverages applications' characteristics for minimizing energy consumption while honoring performance requirements. A predictive model is designed by collecting each application's power-performance profile, then this model, combined with an online predictive strategy, is used to make resource allocation decisions. This contribution is published in paper I (shown in Figure 2 and attached at the end of this manuscript) and presented in Chapter 3 in this manuscript.

- **Mixed sensitivity workload management:** This dissertation presents an energy-performance co-management of mixed-sensitivity workloads on heterogeneous

multi-core systems. The presented framework prioritizes mixed types of applications at run-time and searches through the configuration space to find the optimal configuration for each application. The selected configurations satisfies the performance requirements of applications while minimizing the energy consumption. This contribution is published in paper II (shown in Figure 2 and attached at the end of this manuscript), and is presented in Chapter 4 of this dissertation. By proposing the frameworks in paper I and paper II, the first aim of this dissertation has been achieved.

- **User and Battery Awareness:** This dissertation presents a user- and battery-aware resource management framework for embedded multi-core processors. The proposed approach considers dynamic workload scenarios, user preferences, and user plug-in/out patterns at run-time to provide a suitable trade-off between Battery Cycle Life (BCL) and Quality of Experience (QoE). This framework exploits the user activity profiles to personalize the BCL-QoE trade-off by learning user's habits. This contribution is published in papers III and IV (attached at the end of this manuscript), and in this dissertation is presented in Chapter 5. By proposing the frameworks in paper III and paper IV, the second aim of this dissertation has been achieved.

- **Goal-Driven Autonomy:** This dissertation proposes Goal-Driven Autonomy (GDA) for efficient on-chip resource management. The GDA allows systems to generate and prioritize goals in response to the workload and variations in system dynamics. A comprehensive self-aware resource management framework can be designed using GDA in resource management. The proposed GDA framework has abstracted System, Application, and User as significant entities that are affected by resource allocation decisions, and considers the objectives of these entities at run-time to generate a suitable goal for the resource management. The GDA framework has been improved by proposing Configurable GDA (CGDA) which uses a configurable hierarchy for goal management to reflect the application's performance requirements in lieu of the system's power constraints and user requirements. This contribution is published in paper V (attached at the end of this manuscript) and presented in Chapter 6 in this manuscript. By publishing papers V, the third and forth aims of this dissertation has been achieved.

1.4 Organization

This dissertation consists of two major parts. Part I presents a summarized overview of the research which is organized into Chapters 1-7. Part II consists of original publications.

Part I. Chapter 1 introduces the motivation of this work and presents the re-

search problems, research objectives, an overview of the research contributions, and organization of this dissertation. Chapter 2 provides preliminary background on resource management, self-awareness, and the existing resource management strategies. Chapter 3-6 present specific research problems and proposed solutions, which form the core contributions of this dissertation (as shown in Figure 2). These chapters review, discuss and present the objectives of this research and the frameworks which are proposed for providing comprehensive self-aware resource management. Finally, Chapter 7 summarizes and concludes the study .

Part II. This part consists of Papers I-V, which are originally published articles in various conference proceedings and journal series. Each paper presents the details of the frameworks, which are introduced and discussed in Part I.

2 Background and Related Work

In this chapter, we first provide a brief overview of the necessary background on Heterogeneous Multi-core Platforms (HMP), run-time resource management, and self-awareness. Then, we present the relevant related works on the aforementioned concepts.

2.1 Heterogeneous Multi-core Platform (HMP)

Modern embedded systems are increasingly using HMPs for providing a balance between energy consumption and performance of applications. HMP platforms integrate processors with different capabilities i.e., general-purpose processors, graphics processors, image processors, etc., for power-performance benefits [10; 11]. Further, HMPs can provide processors with various computation capabilities i.e., *performance oriented* cores with higher performance capability and computational power than *low power* cores. For example, an embedded HMP of Odroid XU3 [12] has 8 cores, organized into a big cluster - with 4 ARM A15 cores for providing high performance and a LITTLE cluster - with 4 ARM A7 cores for low power operation. HMPs provide power-performance benefits, however, they also make resource management more complex. The number of available configurations for resource management is relatively high by considering various types and number of cores, which are performing in various ranges of frequencies. Therefore, intelligent resource management is required for exploiting the benefits of HMPs. In this research, we develop such intelligent resource management approaches and present in chapters 3-6. The presented resource management frameworks are evaluated on real HMP of Odroid XU3.

2.2 Run-time Resource Management

Run-time resource management techniques (RTM) assign the available system resources to applications based on the objectives and goals at run-time. Resource management considers system constraints such as power consumption threshold, thermal safety, energy budget, and available cores while satisfying applications' requirements. A diverse set of applications can create dynamic workload scenarios, which require dynamic scaling of the allocated resources. RTM techniques have to consider such workload variations, as well as user requirements, and system and environment

variations to optimize the resource allocation. This chapter provides an overview of various resource allocation knobs as well as resource allocation objectives and goals, and existing strategies for self-aware resource management. This dissertation presents intelligent RTM approaches for HMPs (in chapters 3-6) which select and configure knob settings for achieving run-time objectives and goals.

2.2.1 Resource Allocation Knobs

This subsection lists primary resource allocation knobs and existing approaches that actuate these knobs for resource allocation.

Dynamic Voltage and Frequency Scaling

Dynamic voltage and frequency scaling (DVFS) is one of the actuation knobs which is extensively used in modern processors for power reduction or temperature control. Most of the RTM frameworks [13; 10; 14; 3] have used DVFS for assigning appropriate operational frequencies to functional blocks and satisfying performance requirements of applications as well as energy minimization. The DVFS technique saves energy by lowering the voltage and frequency to optimum levels based on the relationship between power and frequency. Decreasing the frequency level by using the DVFS leads to lower power consumption, however, it also decreases performance of applications. Therefore, RTM techniques have to consider performance targets and then set the DVFS knob. In this research, we use the DVFS knob in the resource management techniques for selecting the level of frequency based on the running applications at run-time and their performance requirements.

Application Mapping

Application mapping is binding new applications to the appropriate set of cores. The RTM frameworks [10; 4; 2] use application mapping as the first step of resource allocation for assigning the cores to the running applications. Such assignment is based on the performance requirements and characteristics (single/multi -threaded, high/low compute intensity) of the applications. The multi-threaded applications may require to execute on more than one core for the efficient energy-performance, while the single-threaded applications are more efficient when execute on a single core. The HMPs provide various clusters that can be considered for efficient application mapping based on the applications' performance requirements. For example, an application with a lower performance requirement can be mapped to the low power cluster at the arrival time. Concurrent execution and arrival of applications create a competition for mapping of applications to the limited cores, which should be handled by resource management. In this dissertation, we consider such challenges in

application mapping and present application characterization methods for handling the challenges.

Power Gating

Power gating is a resource allocation knob for reducing power consumption by shutting off the supply voltage of idle functional units. Power gating encapsulates a sleep transistor as part of the functional units to turn off the supply voltage of that unit. This technique shuts down the core and turns it on when needed, which leads to energy and performance overhead. Despite the overhead, there are several resource management approaches [15; 16; 17; 18] that leverage power gating to enhance power and energy efficiency.

CPU Utilization

CPU utilization refers to the amount of usage of processing resources. Some of the applications require less than 100% of CPU time, whereas some requires much more. Thus, by using *CPU Utilization* as a controlling knob, a processor can be shared between several applications based on the amount of their requirements. Several resource allocation frameworks [19; 20] use CPU utilization knob to allocate the processing resources among various applications on a time-shared basis. Further, setting a fraction of time slice in a processor to idle reduces the CPU utilization, leading to lower power and energy consumption [20].

Degree of Parallelism

Degree of Parallelism (DoP) refers to the number of an application's threads that are executing in parallel. The higher DoP leads to higher parallel threads and thus higher performance in the application. Maximum performance gain by using DoP depends on the nature of applications (single- or multi -threaded), amount of serial computation, and the number of cores in a platform for scheduling parallel threads. DoP knob allows the resource allocation to parallelize the applications which require more performance, while limiting such allocation for applications with lower requirements or single thread applications. DoP knob is used in several resource management frameworks [10; 21; 22; 23] for optimizing the performance based on the applications' nature and requirements. In this dissertation, we uses DoP in the RTM approaches for maximizing performance-energy efficiency.

Task Migration

Whenever a new application arrives, the resource management controller maps it to the initial set of cores by using the Application Mapping knob. However, the initial mapping may not be optimal when other applications enter the system. Thus, Task migration (TM) is required for selecting the optimal type and number of cores for each application at run-time. The applications may need to migrate from a cluster in an HMP to another cluster i.e., from the big cluster to the LITTLE for saving energy or vice versa for improving performance. The TM decision depends on applications' characteristics. Some applications are more compute intensive and require high performance cores (i.e., big cores) while some other applications perform better in low power cluster. TM allows resource management to search through various configurations in an HMP and find the best set of cores for each application, then move the application to the selected set of cores. These knobs ares used by several resource management frameworks [10; 2; 24] for satisfying application requirements based on their characteristics. In this dissertation, we use TM for updating application to core mapping based on dynamic concurrent workloads.

2.2.2 Resource Allocation Objectives and Goals

Resource allocation decisions have to satisfy i) application, ii) system and iii) user level requirements. Such requirements can be formulated as *Objectives* or *Goals*. The application level requirements can be fixed level of performance, Quality of Service (QoS) target in terms of latency, Instruction Per Second (IPS), or Heartbeat [25]. In this dissertation, we consider performance requirements in terms of IPS and Heartbeat. System level requirements can be utilizing the resources within the power and energy budget, as well as thermal safety limit. The other system requirements are providing reliable operation, and slowing down the aging effect in the system. In this dissertation, we consider Thermal Design Power (TDP) as system constraint, beside limited energy budget and battery aging. User level requirements can be variable at run-time, for example, high performance when a high intensive application is running, and saving energy when the battery level is low. In summary, the requirements that construct objectives and goals of resource managements are defined in three levels:

- **Application level**: performance and QoS.
- **System level**: respecting i) energy budget and ii) TDP. Providing reliable operation and decreasing aging effects.
- **User level**: providing i) high performance, ii) saving energy.

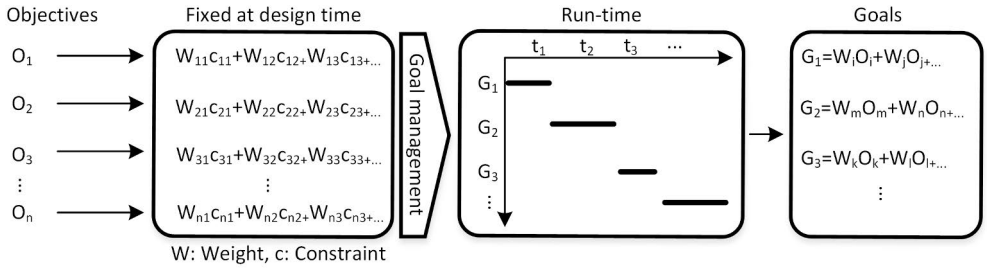


Figure 3. Objective and goal in resource management

Objective

Different levels of requirements can be expressed by various *constraints* as a parameter that controls what resource management is interested in achieving, for instance by keeping it within specific limits [5]. By considering such constraints, the objective is defined in [5] as a cost function that linearly combines different constraints through different weights (i.e., priorities) for achieving a particular result. Resource management controller can have single or multiple objectives. The term *multi-objective* refers to an objective function that has more than one constraint. State-of-the-art resource management policies [2; 21; 9] have single or multiple objective(s) such as maximizing performance, minimizing power consumption, honoring power budgets, ensuring thermal safety, or a mix of the aforementioned. Such objectives are typically fixed at design time, and resource management policies try to meet those at run-time. The left box in Figure 3 illustrates the formulation of objectives. In this dissertation, we propose multi-objective resource management frameworks in chapters 3-5. The constraints in our proposed methods are power and energy consumption, performance of applications, and battery aging.

Goal

Based on the definition in [5], goal is an abstract level defined plan of the system that can be updated at run-time in response to the environment or the system's own state. Updating a goal results in re-prioritizing objectives and updating the weights in the objective function. The Goal can be formulated as a weighted combination of different objectives. Multiple conflicting objectives can be unified through goal formulation, forming a hierarchy of goals with different priorities [26; 8]. Figure 3 illustrates goals and objectives in on-chip resource management content. As shown in Figure 3 the objectives are weighted sums of several constraints which are fixed at design time. A goal management controller can dynamically generate goals at run-time based on the requirements, system state, and environment changes. The goal can be a weighted combination of one or more objectives as shown in the Figure. In this

dissertation, we propose a framework in Chapter 6 for formulating objectives to goals at run-time. We present a goal management technique for resource management in HMPs.

Objective vs Goal

Objectives are weighted sums of several constraints which are fixed at design times, whereas goals are weighted combinations of objectives that dynamically changes at run-time. A goal is an achievable outcome from the existing objectives which generates intelligently at run-time.

2.3 Self-Awareness in Resource Management

Self-awareness is a notation that has been widely used in psychology as well as various domains of computing such as organic computing, autonomic computing, adaptive systems, and self-organizing systems [7]. Self-awareness enables the ability to monitor the environment's as well as self's states and requirements and reacting to the requirements based on the current state. Such capability is essential for efficient resource management. A self-aware resource management framework can monitor the requirements and available resources at any given time and optimize resource consumption while satisfying the various entities' requirements. This dissertation presented self-aware resource management approaches in Chapters 3-6 and evaluated those over state-of-the-art approaches. In this section, a brief introduction of self-awareness is provided followed by the benefits of self-awareness in resource management.

2.3.1 What Is Self-Awareness?

Self-awareness in IT systems is defined by the combination of three self-* features as follows [27]:

- *Self-reflective*: Being aware of i) execution environment and hardware infrastructure, ii) operational goals such as satisfying performance or optimizing energy efficiency, and iii) dynamic changes in (i) and (ii). This feature of self-awareness is used in this research for monitoring the running applications, user's requests, and sensory data of hardware in embedded systems, and then generating objectives and goals for resource management. The resource management approaches which are presented in Chapters 3-6 are self-reflective.

- *Self-predictive*: Being capable of predicting the impact of dynamic changes e.g. changing in performance requirement as well as the impact of potential adaption actions such as resource allocations. In this research, we leverage the

self-predictive feature for predicting the effect of knob settings on performance and energy consumption and then, make the best resource allocation decision based on that.

- *Self-adaptive*: Adapting proactively as the environment changes to ensure that the system goals such as maximizing performance and energy efficiency are met. This feature is used in this research for adapting resource management decisions to the applications, user, and system changes to ensure the goals.

Figure 4 shows the architecture of a self-aware system that contains all crucial elements of self-awareness. The self-aware system consists of self-monitoring and environment monitoring which collect data by using i) sensors, ii) static and dynamic models, and iii) episodic history, then send the collected data to the decision making element (as shown in Figure 4). An accurate evaluation of self and environment is the essential preliminary for effective decision making and goal management. The decision making element considers the current state of self and environment as well as the goal which is generated by goal management based on the goal hierarchy, then makes a decision (e.g., resource allocation decision). The decisions which consist of a set of actuation knobs are transferred to the execution engine (e.g., HMP) and apply to the hardware platform. As shown in Figure 4 learning is the essential part of each element in a self-aware system. The learning techniques can be used for generating models, predicting the decision effects, and generating a suitable goal.

The three self-* features of self-awareness are reflected in Figure 4. The *self-reflective* feature of the self-awareness is located in the static and dynamic models in the self-monitoring as well as goal management, and goal hierarchy. The *self-predictive* feature is shown in dynamic self-models, and the *self-adaptive* feature is reflected in the decision making, goal management, and goal hierarchy.

2.3.2 Benefits of Self-Awareness in Resource Management

Self-awareness enables a system to deal more effectively with the complexities that come from i) system itself, ii) environment and iii) various goals and objectives [7]. Therefore, leveraging self-awareness in resource management enables management to deal more effectively with i) various number and type of processors, ii) various type of applications, iii) competition of concurrent applications, iv) users' requirements, and v) system's constraints. Self-awareness can improve energy efficiency as well as performance, and any other objectives for on-chip resource allocation. For example, in [28], a resource allocation is proposed in which local monitoring is integrated into complete system-level evaluation to optimize energy consumption. In this framework, the system load balancing, scheduling, and task allocation are significantly improved by using a comprehensive self-assessment. In the other framework which is proposed in [29], dynamic adaption is achieved by using a smart interface

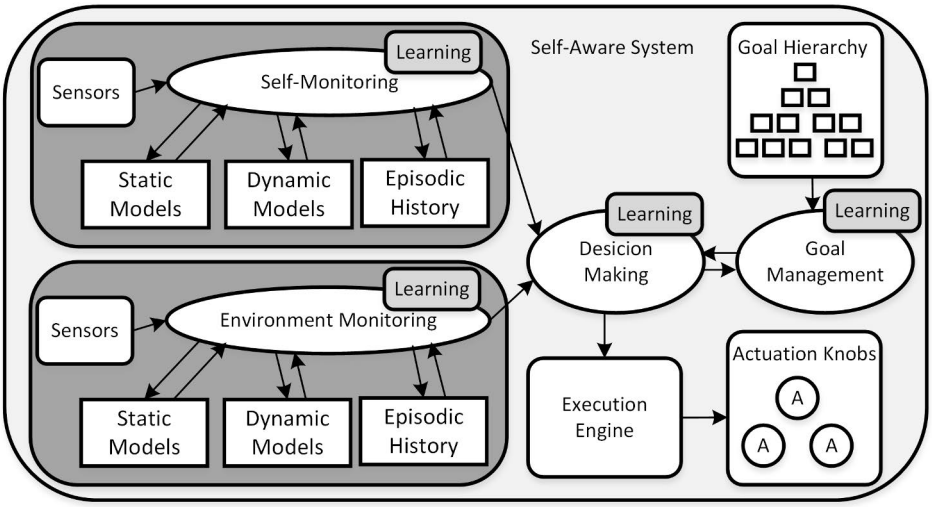


Figure 4. Illustration of a self-aware system [7].

between platform and application. The platform is in charge of reaching the application's performance goals by continuously evaluating the system's performance and making appropriate resource allocation adjustments. An efficient self-model enables the system to allocate limited resources by considering the requirements and constraints.

In summary, self-awareness provides a solid foundation for adaptation decisions by offering a full assessment of the condition of the system and its environment and hence can improve the quality of adaptation. Moreover, self-awareness leads to increasing of efficiency as a result of better and more efficient resource utilization. Therefore, in this research, we leverages self-awareness in resource allocation for better goal management, adaptation, and decision making. All the presented resource management approaches in Chapters 3-6 are self-aware.

2.3.3 Self-aware Resource Management Approaches

The main challenges of resource management in embedded systems are the various number of objectives, the severity of resource constraints, and the unpredictability of the environment. For addressing such challenges, self-aware resource management has been proposed by other researchers including [30; 31; 32]. Self-aware systems adapt their behavior based on the environmental conditions and requirements to achieve the target goal at any given time. As shown in Figure 4, learning techniques are essentials in each element of the self-aware system. Therefore, recent resource management approaches leverage machine learning techniques for improving their efficiency. The machine learning techniques are categorized as i) offline approaches

ii) online methods, and iii) hybrid (combination of both), which have been used by state-of-the-art resource management frameworks as referred in the following.

Offline Machine Learning Techniques

Some of the resource management approaches design a machine learning model offline and use that at run-time. Such approaches collect data by executing applications in the same environment that will be used at run-time, and then create a model based on the data collected. The model is fixed at design time and cannot adapt to any unpredictable events. Many resource management approaches [33; 21; 2; 34] utilize offline machine learning techniques to build predictor models and then use the models at run-time and allocate the resources. Such approaches require extensive application profiling and sample computation for training an accurate predictor. The work presented in [33] builds a tree-based model by offline machine learning to find a suitable frequency of CPU and GPU, in which power consumption is minimized under performance constraints. This method requires a collection of power and performance metrics at various CPU and GPU frequencies using a diverse set of application workloads. Similarly, DyPO [33] uses extensive offline application profiling to create a library in which an optimal resource management configuration is assigned to each system and workload status. The statuses are defined based on features such as memory accesses, cache misses, CPU cycles, and instructions retired. The extracted offline library is used at run-time to identify the best possible configuration by measuring the aforementioned features. The major drawbacks of offline techniques are i) they are not scalable for large systems with high number of resource management configurations, and ii) they are not able to capture dynamic workload variation, limiting their efficiency to a known set of applications.

Online Machine Learning Techniques

For solving the problems of offline approaches, online machine learning techniques have been recently used in the resource management field [11]. In online learning, the characterization and model building are performed at run-time using various machine learning algorithms such as least mean squares (LMS) [35], recursive least squares [36; 37; 38], and Reinforcement Learning (RL) [39; 40; 41; 42]. RL is a model-free approach that evaluates the actions at run-time and assigns rewards to each action for future decisions. One popular method of RL is Q-learning in which the rewards are stored in a table for each pair of state-action. Several resource management approaches leverage Q-learning to make proper decisions at run-time [43; 44; 45]. The Q-learning method needs to define various states for the system and assign a reward to each state by executing actions at run-time. More number of states lead to more accurate prediction, which requires a larger table to

store Q-values. When size of the Q-table increases, additional memory is required, and the convergence time increases. Therefore, such approaches cannot be scalable for systems with a high number of states and actions (more details are presented in Section 6.2.5). To address these limitations, the Deep Q-learning approach is used for estimating the Q-value, eliminating the need for storing data [46; 42]. The aforementioned online models increase self-awareness of resource management approaches, however, the online models require a high number of run-time iterations to converge to the optimal resource management decision. To decrease the convergence time, online machine learning techniques can use results from offline analysis, decreasing the overhead and complexity of run-time processing. In this dissertation, we use combination of online machine learning techniques such as Q-learning with offline models to create self-aware resource management.

Hybrid Machine Learning Techniques

A combination of offline and online algorithms create a new class of machine learning techniques, known as the hybrid approaches. Hybrid approaches usually achieve better efficiency with lower run-time overhead and complexity, compared to pure online optimizations. Besides, hybrid approaches address the limitations of pure offline methods in adapting to the new workloads. Imitation Learning (IL) is a hybrid approach that transforms an optimal offline solution into an efficient online policy [47; 23]. The approach presented in [23] designs an optimal policy predictor offline by exploiting data characterization for target applications, then learns a suitable resource management policy online by imitating the behavior of the offline policy predictor. In [23], linear regression (LR) and regression tree (RT) algorithms are used to generate policies with minimal storage and run-time decision-making overheads.

In summary, offline machine learning techniques in resource management require extensive data collection (such as power consumption, energy and performance of applications), considering various configuration in the system. The offline models generated by such data collection are not scalable and adaptable to new systems and applications. Online machine learning techniques address the above limitations, while increasing run-time overhead and convergence time. Leveraging both offline and online methods' form hybrid machine learning approaches that optimize the prediction accuracy for known and unknown workloads compared to offline methods and decrease run-time overhead compared to online approaches. However, with a wide range of machine learning algorithms, finding a suitable hybrid method requires investigation of HMPs architecture, various resource allocation knobs, workload scenarios, and system goals. A suitable method must find the optimal trade-off between run-time overhead and prediction accuracy, by selecting appropriate machine learning algorithms. In this dissertation, we leverage combination of offline and online

machine learning techniques for improving the resource management techniques in achieving their goals.

3 Handling concurrent applications

In this chapter, we discuss the challenges of resource management when multiple concurrent applications execute in an HMP. As discussed in Chapter 1 and 2, execution of multiple applications concurrently rises complexities in prioritizing the applications and handling the available resources, such that all the applications meet their requirements. Different types of applications require different amounts of resources based on their characteristics. Studying application characteristics develops an understanding of applications' behavior in different resource management configurations that can enhance resource allocation decisions. In this chapter, we introduce bias as a characteristic for applications that can guide resource management decisions. Further, we present concurrent application bias scheduling for energy efficiency of HMPs and the evaluation results based on original publication, which is attached as Paper I.

3.1 Challenges of Handling Concurrent Applications

There are two levels of challenges in resource management of multiple concurrent applications viz., i) hardware level, and ii) application level. At the *hardware level*, different actuation knobs such as DVFS, DoP, and the type of active cores (among a heterogeneous set) provide a wide range of resource management configurations with different performance-energy trade-offs [21; 11]. Selecting a suitable configuration that provides optimal performance-energy trade-off among such wide range of potential configurations is challenging. At the *application level*, each application has specific characteristics, and executing different applications concurrently creates significant diversity in workload scenarios [4; 48]. This requires an intelligent resource allocation to meet the performance requirements of all the applications. Furthermore, an application's energy consumption profile under different resource configurations differ depending on the application's characteristics. For example, the energy efficient configuration for an application may be one LITTLE core, while for the other application is four big cores. Despite the number of resource configuration options for reducing energy consumption, the variety and complexity of applications make it an exhaustive exploration. For example, a configuration that provides the best performance-energy trade-off for a specific application may not be suitable when a different application arrives and executes concurrently.

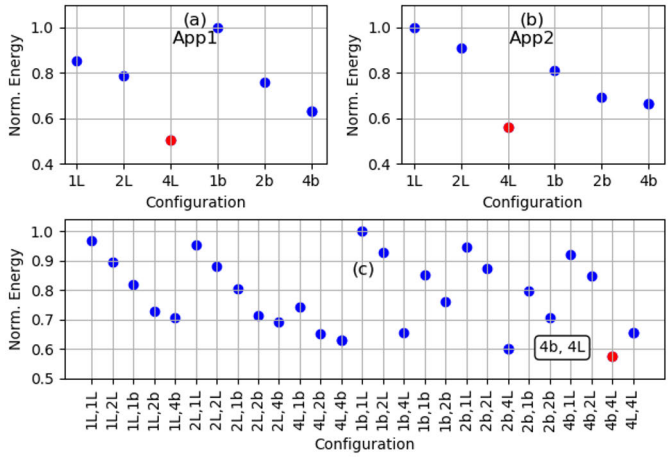


Figure 5. Energy measurement of three different test-cases on various possible configurations in an HMP with 4 big (b) and LITTLE (L) cores. (a) App1 executes individually, (b) App2 executes individually, (c) App1 and App2 execute concurrently on n(b/L),m(b/L) configurations, respectively. [49].

For clarity, we provide a real example by considering 3 different test cases in a real HMP platform, namely Odroid XU3 [12]. The Odroid XU3 contains 4 big and 4 LITTLE cores. We measure the energy consumption of the 3 different test cases in 6 possible resource management configurations. The 6 major configurations as $< num\_cores >< core\_type >$, including 1L, 2L, 4L, 1b, 2b, and 4b, where L and b denote LITTLE and big core type, respectively. The 6 configurations are comprehensive enough for showing power-performance characteristics of applications in the Odroid XU3 platform. We first execute two different applications separately in the aforementioned configurations and measure the energy consumption and performance of applications. Then, the two applications are executed concurrently in 27 possible configurations combining the big and LITTLE cores (shown in Figure 5). These 27 ($27 = 36 - 9$) configurations are the feasible configurations in which each core contains one application. The 9 unfeasible configurations are the configurations in which sum of allocated big or LITTLE cores are more that 4 (for example, 4b,2b or 3L,2L). Figure 5 shows the energy measurement of the 3 test cases. As shown in Figure 5 (a) and (b), for both applications, 4 LITTLE (4L) configurations (red points) provide the lowest energy consumption. However, when they execute concurrently (Figure 5 (c)), the combination of 4 big (4b) for the first application and 4 LITTLE (4L) for the second application (4b,4L) is the best configuration in terms of energy consumption. The number of possible configurations for multiple concurrent applications, considering different levels of frequency becomes even larger. Finding the optimal choice among such a large number of configurations requires an intelligent decision making framework which is presented in this chapter. The machine learn-

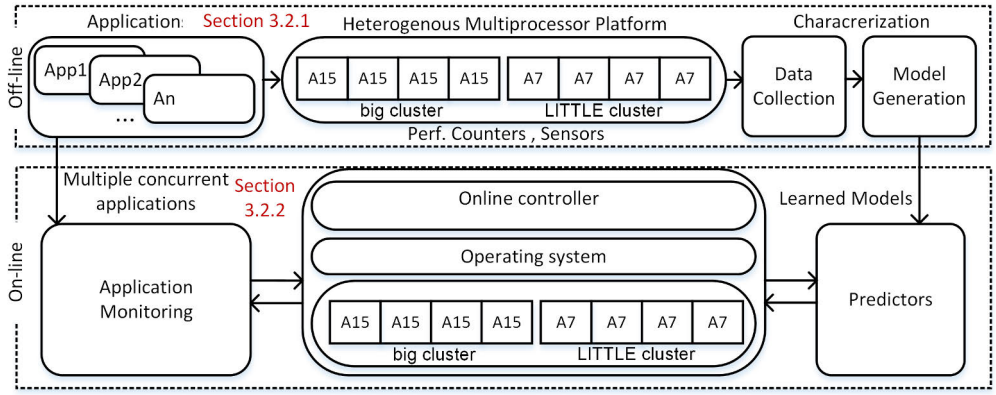


Figure 6. High-level architecture of concurrent application bias scheduling approach for energy efficiency of HMPs [49].

ing techniques provide such intelligence by monitoring various states of the system, learning, and making decisions. The presented solution leverages offline learning as well as online monitoring and control for selecting the best configuration under dynamic and concurrent workload scenarios. We extract applications' characteristics offline and design a model for predicting the application characteristics at run-time. Based on the predicted characteristics, we prioritize among the concurrent applications and assign the resources accordingly. More details on the proposed approach are provided in Section 3.2.

3.2 Solution: Bias Aware Resource Management

In this section, our resource management approach for handling concurrent applications on HMPs is presented. This approach leverages offline characterization and online control of applications, to address the challenges discussed in Section 3.1. A high-level architecture of the proposed method is shown in Figure 6. As shown in the Figure, in the off-line phase, the applications execute on the HMPs one by one, and the related data as i) performance and ii) energy consumption and iii) execution time is collected for each application. Then, based on the collected data, offline models for predicting bias and performance status are generated and used online. In the online phase, the arriving applications are monitored continuously, and an online controller makes resource allocation decisions based on the predictors' outputs.

In the presented approach, a notation of *bias* is defined for each pair of application-configuration. The bias is the ratio of performance-per-power for an application in a specific configuration to the best achievable performance-per-power when the application executes standalone. Understanding these biases for every application enables optimized resource management decisions at run-time. The relative value of the bi-

ases shows the priority of the applications for consuming the available resources. *Application Bias* (AB) is defined as a list of *Configuration Biases* (CB) for possible configurations (c) in an HMP. CB_c is formulated as:

$$CB_c = \frac{(Perf)_c/(Pow)_c}{(Perf)_{Bc}/(Pow)_{Bc}} \tag{1}$$

where $(Perf)_c$, $(Pow)_c$, $(Perf)_{Bc}$, and $(Pow)_{Bc}$ are performance and power consumption of the application when executes on configuration c and Bc, respectively. Bc is the best configuration which provides the lowest energy consumption with the highest possible performance for the application.

In addition, we define the relative amount of energy consumed for each application as *weight* of that application (W_A) which is shown in Equation 2.

$$W_A = \frac{Average(E_i)}{E_t} \tag{2}$$

where E_i denotes the amount of energy consumption of the application in 6 major configurations (i.e., 1b, 2b, 4b, 1L, 2L, 4L) of the platform, and E_t represents the total energy of the system (i.e., battery capacity). The *weighted bias* (WB) as formulated in Equation 3 provides characterization of applications in terms of relative energy consumption. Weighted biases are used for online management which guides resource management decisions for handling multiple concurrent applications. The application with higher weighted bias is chosen as the priority.

$$WB = W_A \times CB_c \tag{3}$$

where CB_c represents the configuration bias of the application when it is executed on configuration c. In the proposed approach, a model is generated for predicting WB of various applications at run-time by using the offline data. The details of model generation are provided in Section 3.2.1 and 3.2.2

3.2.1 Offline Characterization

Offline characterization consists of two phases viz., i) data collection, and ii) model generation, which are explained in the following.

Data Collection

For data collection, six major configurations including 1L, 2L, 4L, 1b, 2b, and 4b are considered. These 6 configurations are enough comprehensive for representing the power-performance characteristics of applications. The frequency levels of these configurations are fixed at 1.8 GHz for the big cluster and 1.4 GHz for the LITTLE

cluster. Then, we executed a set of applications on the aforementioned configurations to collect their power consumption and performance (in terms of IPS). The applications are from Rodinia benchmark suite [50] which are suitable applications for embedded systems' evaluation [51; 52; 11]. After collecting data for the two fixed levels of frequency, the energy consumption and performance can be estimated at the other levels of frequency by using the available power and performance models [6; 1; 53]. We use the data collected for generating *bias* and *performance* models. From the whole data which has been collected in this phase, 6 energy consumption values for the 6 major configurations are stored into *App_Info_Array*, as applications' characteristics for online management.

Model Generation

In this phase, two generic models viz., i) *bias model* and ii) *performance model* are designed by using a combination of data as (num_cores, $core\_type$, $freq(GHz)$, $E_1 - E_6(J)$) for each pair of application-configuration. $E_1 - E_6$ are the values of energy consumption for the 6 major resource configurations, which are stored in *App_Info_Array*. The *bias model* provides an estimation of applications' biases for each configuration, and the *performance model* predicts which configurations meet the performance requirement of each application. The performance requirements are defined at design time in terms of Instruction Per Second (IPS). The performance model is a classifier that determines whether a specific resource configuration satisfies the application's requirements or not. The output of bias model CB_c is an estimation of configuration bias for an application with 6 major energy consumption equal to $E_1 - E_6$ on configuration $< num\_cores >< core\_type >$ at frequency level $freq$. The bias model is generated by using linear regression as follows:

$$\begin{aligned} CB_c = \; & -0.3freq - 0.05core\_type + 0.04num\_cores \\ & + 0.01E_3 + 0.0017E_4 \\ & - 0.0005E_5 + 0.001E_6 + 0.86 \end{aligned} \tag{4}$$

where the input features of the model are (num_cores, $core\_type$, $freq$, $E_1 - E_6$). The constraints in the model are estimated by assuming the linear relationship between input features and output. Such relationship is estimated based on the collected data (input features) and corresponding CB for those.

Similarly, the performance model is generated as a classifier by using Ridge Classifier [54]. Ridge Classifier is easy to implement and has low prediction error in our test case. The output of the performance model is a binary label that shows whether a configuration $< num\_cores >< core\_type >$ at frequency level $freq$ can satisfy applications requirements (1) or not (0). The linear performance model is formulated

in Equation 5.

$$\begin{aligned} P = \; & 0.7freq - 0.74core\_type + 0.14num\_cores \\ & + 0.01E_1 - 0.07E_2 + 0.051E_3 - 0.0018E_4 \\ & + 0.013E_5 - 0.012E_6 - 0.17 \end{aligned} \tag{5}$$

where P is the classification value. The positive value for P shows label 1 (meeting the requirements), and the negative value represents label 0. The constraints in the performance model are estimated by using the offline data. The offline data is collected from execution of various applications on different configurations of an HMP. Therefore, the model can predict the CB and performance violation on the same HMP with considering various applications.

3.2.2 Online Management

The online management leverages the generated offline models, the online learner, and the controller for handling multiple concurrent applications. The details of each component for online management are described in the following subsections.

Application Monitoring

The *Application Monitoring* component is responsible for monitoring the arrival and exit of the applications at run-time. The *Application Monitoring* creates and updates a list of running applications and passes the list and the stored information for each application to the *Predictor*. The information has been stored in *App_Info_Array* which contains 6 values of energy consumption for executing the application on the 6 major configurations (mentioned in Section 3.1). The *App_Info_Array* represents the characterization of each application that is collected offline. For unknown applications, there is an *Online Learner* component which collects the data at run-time. The *application monitoring* periodically monitors the existing applications, updates the list of applications, and passes the application's characteristics to the *Predictor*.

Predictor

The *Predictor* leverages the bias and performance models that are generated offline for predicting the bias and performance satisfaction for every pair of application-configuration in the platform. The *Predictor* receives the *App_Info_Array* for each existing application from *Application Monitoring* or *Online Learner*, then generates a bias list for each application (using Equation 4). The bias list contains the configuration biases for every combination of $num\_cores, core\_type$ and $frequency$ level that meets the performance requirements of the application. To avoid excessive

calculation, we use the performance model (Equation 5) to check if the configuration fulfills the performance requirements of the application. Then, the bias for that application-configuration pair is calculated and added to the bias list. Each bias in the bias list corresponds to a single configuration.

Configuration Selection

The *Configuration Selection* is responsible for selecting the configuration that provides the lowest energy consumption while satisfying the performance requirements for each application. The *Configuration Selection* checks the applications list which is provided by *Application Monitoring*, and if there is only one application, the *Configuration Selection* selects the lowest energy consumption among $E1 - E6$ from the *App_Info_Array*, and the configuration link to that energy is the final configuration. Otherwise, the *Configuration Selection* considers bias lists of applications and investigates through the whole feasible configuration for the concurrently running applications for selecting a combination of configurations with the lowest total energy consumption. We define a feasible configuration as a configuration in which there are no two applications that are executing on the same core. Execution of more than one application on a single core leads to relatively higher energy consumption and lower performance [34]. Exploring through unfeasible configurations increases the online and offline overhead of resource management. For example, for two concurrent application, 25% of configurations are unfeasible which requires offline data collection and online exploration for configuration selection. For decreasing such overhead, we neglect the unfeasible configuration for resource management.

The highest bias in the bias list of an application is linked to the configuration with the lowest energy consumption for that application when executing individually. However, that configuration may not be optimal by considering the weighted bias of the other concurrent applications. Therefore, for handling multiple applications, the *Configuration Selection* calculates a weighted sum of biases for every application-configuration pair that is feasible and exists in the bias lists and then creates a new combined bias list for the existing applications. The highest value in this new list refers to the best combination of configurations for the concurrently running application. This new configuration may change the previous optimal configuration for one application, but it reduces the total energy consumption of the system.

Online Learner

The *Online Learner* is used for handling unknown applications. The proposed framework leverages offline data collection for characterizing the applications and stores part of the collected data as the application's characteristics. However, for unknown applications, there is no previously collected data. Therefore, when a new unknown

application is detected by Application Monitoring, *Online Learner* is activated to collect the data for characterizing the unknown application. The online learner maps the application to the 6 major configurations (as mentioned in Section 3.2.1) and measures the power consumption and performance of the application in the current frequency level. The power consumption and performance for the other frequency levels can be estimated by power and performance models in [6; 1; 53]. If the cores are not free for allocating the unknown application and collecting the data in all the 6 major configurations, the controller updates the current application-to-core mapping to release at least one LITTLE and one big core and collects data by assigning the unknown applications to the released cores. By using the collected data, CB_c can be calculated based on Equation 1. The weight cannot be calculated for the unknown applications, thus the resource allocation decisions will be made by considering CB_c. The weight of an unknown application can be measured after the completion of the application and stored for future use. Considering the weight improve the predictions and resource management for unknown applications.

Performance Controller

The *Performance Controller* fine-tunes the frequency level of the platform in order to satisfy the applications' performance requirements. The frequency level has been selected in the *Configuration Selection* phase, however, the applications interference may lead to decreasing the performance of concurrent applications. Therefore, *Performance Controller* monitors the performance of the applications at run-time and adjusts the frequency level based on the application's requirements. When performance of an application is less than the requirements the *Performance Controller* increases the frequency one level (100 Mhz) and monitors the performance. The frequency level increases step by step to provides the suitable performance for all the applications.

3.3 Evaluation

In this section, we present the evaluation of the proposed framework reported on Paper I, which is attached at the end of this manuscript. The framework is compared against state-of-the-art resource allocation approaches for handling concurrent applications on HMPs viz. i) DyPO [21], and ii) AdaMD [10]. DyPO considers the application bias for guiding resource allocation, which we also consider in this work. Thus, we select DyPO for comparison against our work to show the effect of using weighted bias for multi- application workload scenarios. DyPO used offline data collection for finding the best energy-performance trade-off at run-time. AdaMD is the most similar work to our approach, which relies on online data collection for estimating speed-up to guide the resource allocation decisions. However, the AdaMD

Table 1. List of the applications which are used for evaluation [49]

Application	Computation	Domain	Norm. compute int.	Abbreviation
Particle filter	Structured grid	Filtering problem	0.625	Pf
Heartwall	Structured grid	Medical imaging	0.47	Hw
Streamcluster	Dense linear algebra	Data mining	1	Sc
Srad	Structured grid	Image processing	0.44	Sr
Kmeans	Dense linear algebra	Data mining	0.8	Km
Leukocyte tracking	Structured grid	Medical imaging	0.97	Lt
Breadth-first search	Graph traversal	Graph algorithms	0.53	Bf

overlooks the effect of the total energy consumption per application as a weight. Furthermore, the proposed method is compared with Linux *Powersave*, *Performance*, *Ondemand*, *Interactive*, and *Conservative* governors, which are standard baselines. The *Powersave* governor set the frequency of the device to the lowest for saving power consumption. In contrast, the *Performance* governor sets the frequency to the maximum level for satisfying the performance requirements. *Ondemand*, *Interactive*, and *Conservative* governors set the frequency based on the utilization threshold [55; 48]. The experimental setup and the results of the comparison are presented in the following.

3.3.1 Experimental Setup

In this section, we present the experimental setup i.e., platform and workload scenario, which has been used for evaluation.

Platform

Our evaluation platform is the Odroid XU3 [12] HMP with Samsung Exynos 5422 processor. The processor contains two clusters as big and LITTLE, which are customized for high performance and low power operation, respectively. The big cluster contains 4 ARM A15 cores that perform within 0.2 to 2 GHz frequency, and the LITTLE cluster has 4 ARM A7 cores with 0.2 to 1.4 GHz range of frequency. The frequency levels of big and LITTLE clusters are adjustable individually using DVFS (with a 0.1 GHz step). The Odroid XU3 platform provides on-board power sensors for measuring the power consumption of each cluster. Further, the performance of applications can be measured in terms of IPS by using the available operating system's performance counters in the platform. The proposed framework is implemented in C++ as a Linux user-space process, and the monitoring and resource allocation operates periodically at a parametrizable epoch. Throughput this thesis, the same platform has been used for evaluation purposes.

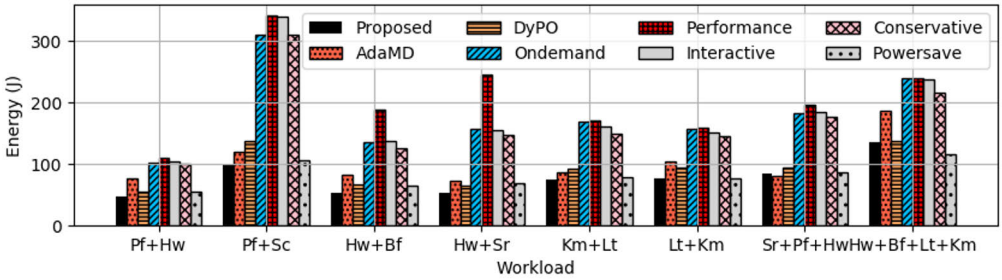


Figure 7. Comparison of energy consumption of the proposed method against standard baseline resource management approaches over various concurrent workload scenarios [49].

Workload

For evaluating the proposed method, we use a combination of single and multi-threaded applications from Rodinia benchmark suite [50], summarized in Table 1. The applications arrive and leave the system in an unknown manner. Rodinia benchmark applications cover a wide range of computation intensity as listed in Table 1 and are suitable for embedded processors evaluation [51; 52]. The computation intensity of the applications is measured as normalized speedup based on definition in [10]. The higher speedup shows the higher computation intensity of an application.

3.3.2 Experimental Results

We compare the energy consumption of the system and execution time of applications using the proposed framework against the state-of-the-art approaches and standard baselines. Figure 7 shows the energy consumption of different workload scenarios containing 2, 3, or 4 concurrent applications over several resource management frameworks. The black bar shows the energy consumption of each workload scenario when using the proposed framework, which has the lowest energy consumption compared to the other approaches. As shown in Figure 7, *Powersave* method also provides relatively low energy consumption, however, this approach neglects the performance requirements, leading to higher execution time. In the first two workload scenarios, DyPO maps both applications to 4 LITTLE cores, which is the best configuration for the individual run of each application. However, this mapping leads to higher energy consumption than the proposed method. In the third and fourth workload scenarios, the best configurations for individual run of *Hw* is 4 LITTLE core and 1 LITTLE for *Bf* and *Sr*. The proposed method assigns 4 LITTLE cores to *Hw* at first, and then by the arrival of the second application, the controller releases one core for the second application. This mapping strategy leads to lower energy consumption compared to the other approaches. The fifth and sixth workload scenarios in Figure 7 show different order of arrival for two applications that af-

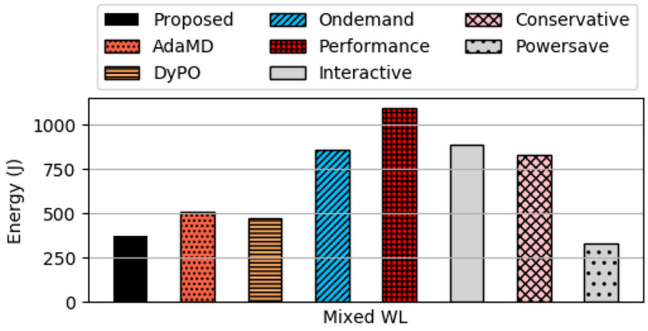


Figure 8. Comparison of energy consumption over mixed workload scenario [49].

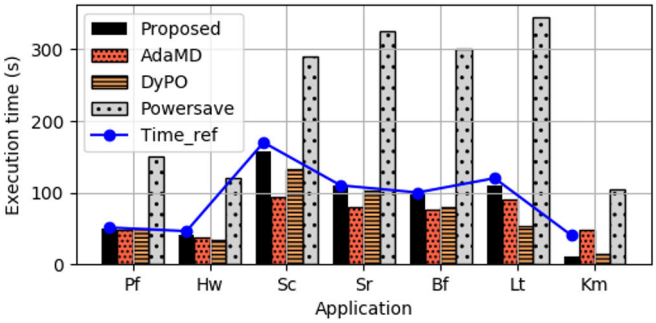


Figure 9. Execution time of running applications in dynamic workload scenario, using various frameworks [49].

fect energy consumption based on the resource management approaches. In DyPO, the selected configurations for both *Km+Lt* and *Lt+Km* are the same, which leads to higher energy consumption compared to the proposed method. The proposed method maps each application to the best configurations at first, and updates the configuration by the arrival of the next applications. Figure 7 shows AdaMD provides lower energy consumption for 3 concurrent applications compared to the proposed method, however, it leads to significantly higher energy consumption for 4 concurrent applications scenario. Further, the energy consumption of a mixed workload scenario which combines all the workloads in Figure 7 is presented in Figure 8. The result shows lower energy consumption for the proposed method compared to the other approaches.

The execution time of applications that are run in the mixed workload scenario is presented in Figure 9. The black bar in the figure shows the execution time of the applications over the system with the proposed resource management framework, and the blue circles show the deadline of applications. For all the applications, the black bars meet the expected deadline. Thus, the proposed method respects the performance requirement of all the applications while providing lower energy consumption

compared to the other approaches. As shown in Figure 9, DyPO and AdaMD also meet the deadlines, however, they consume more energy by using more resources. Although the Powersave approach consumes relatively low energy consumption, it neglects the deadlines for all the applications.

The experimental results over the real-world HMP show relatively lower energy consumption for concurrent workload scenarios by using weighted bias for prioritizing the applications. Furthermore, the proposed approach respects the performance requirements of all the concurrent applications while providing the lowest possible total energy consumption.

4 Handling Mixed Sensitivity Workload Scenarios

In chapter 3, we presented a resource management strategy for handling concurrent applications that have similar performance requirements. However, real workload scenarios in embedded systems often contain a combination of mixed sensitivity applications i.e., applications which are i) latency-sensitive or ii) throughput-driven. Latency-sensitive applications require a certain fixed level of performance within a strict deadline [56], while throughput-driven applications do not have any strict latency constraints, but require a higher overall performance [57]. In this chapter, we discuss the challenges of handling mixed sensitivity workloads in HMPs and present a resource management strategy to address these challenges. The proposed solution uses offline characterization as well as online management to prioritize latency-sensitive and throughput-driven applications. The objective of the resource management approach is satisfying the performance of mixed sensitivity workload with minimum energy consumption. The proposed method estimates the priority of latency and throughput applications by considering offline collected data as well as online dynamics, then assigns the available resources to the applications based on the priorities. The evaluation and comparison of the proposed method against standard baselines are also presented in this chapter based on Paper II, which is attached at the end of this manuscript.

4.1 Challenges of Handling Mixed Sensitivity Workload

Concurrent execution of latency-sensitive and throughput-driven applications can lead to interference and conflicting requirements in embedded systems with limited power and energy budget. Considering limited resources, prioritizing among different types of applications with diverse requirements, and optimizing energy consumption make resource management challenging. Core-level heterogeneity in HMP platforms provides power and performance efficient cores for the optimized management of concurrent mixed sensitivity workloads. Similarly, resource management knobs such as DVFS, DoP, and TM provide more flexibility in finding the optimal energy-performance trade-off. Although the aforementioned resource management knobs offer relatively higher efficiency, they also expose a wide range of resource allocation choices. Finding the suitable configurations through such a wide range

of resource allocation for concurrent mixed sensitivity applications is complex and time-consuming. The latency-sensitive applications have strict deadlines which must be considered by resource management, while the throughput-driven applications require to consume a high amount of available resources for providing relatively high throughput. The competition between concurrent latency-sensitive and throughput-driven applications makes decision making even more complex. Intelligent decision making is required for prioritizing among the mixed sensitive applications and selecting the configurations which satisfy both latency and throughput requirements of applications. In summary, there are two major challenges for handling mixed sensitivity workloads:

1. Prioritizing among the latency-sensitive and throughput-driven applications in a mixed-sensitivity workload based on their run-time requirements.

2. Finding the best configuration that satisfies performance requirements of mixed sensitive applications and minimize energy consumption among a wide range of configurations in an HMP platform.

For clarification, the first challenge is demonstrated by an example in Figure 10. Figure 10 shows performance of two applications i.e., *ParticleFilter* (A1) and *Streamcluster* (A2) from Rodinia benchmark suite [50] over three different prioritizing strategies. Rodinia benchmark suite is a suitable benchmark for evaluating HMPs [11; 51]. A1 is throughput-driven and A2 is latency-sensitive. The prioritizing strategies are i) higher priority for throughput-driven applications (*P1*), ii) higher priority for latency-sensitive applications (*P2*), iii) smart and adaptive prioritizing strategy (*P3*). The red dotted lines in Figure 10 show the strict deadline for A1 and the blue dotted lines shows the acceptable range of throughput for A2. Higher throughput than the upper bound leads to resource over-utilization, whereas lower throughput leads to user dissatisfaction. As shown in Figure 10, in *P1* strategy, higher priority for A1 leads to faster execution of A1 beyond the required performance range, overusing the available pool of resources. The remaining resources are not enough for executing A2, resulting in A2 missing the target deadline. In *P2* strategy, higher priority is assigned to A2 which leads to over-utilization of resource by A2, while A1 fails in meeting the target performance requirements. P3 is an adaptive and smart prioritizing strategy that dynamically updates the priorities at run-time. P3 assigns a higher priority to A1 at first. Upon the arrival of A2, the priority of A1 decreases and A2 acquires the higher priority. At this point, the resource management strategy searches for the best configuration through the wide range of configurations, which arises the second challenge. In this example, the resources are divided between A1 and A2 to satisfy A1's target performance and achieve a throughput in the required range for A2. Finding such configuration in an HMP platform is complex and challenging. For addressing the aforementioned challenges, we present an energy-performance

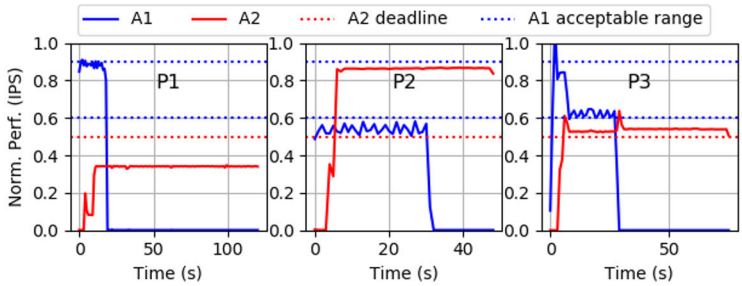


Figure 10. Comparison of three different prioritizing approaches for handling A1 and A2 which are throughput-driven and latency-sensitive applications, respectively [58]

co-management framework based on a smart prioritizing strategy, and provide a resource management method for finding the optimized configuration among wide range of configurations in an HMP platform.

4.2 Proposed Solution: Energy and Performance Co-Management of mixed-sensitivity Workloads

In this section, we present a framework for handling the resource management challenges for mixed-sensitivity Workloads. The proposed framework consists of two major parts which are shown in Figure 11 and detailed in Section 4.2.1 and 4.2.2.

4.2.1 Offline Characterization

The characteristics of applications are required for model generation and efficient resource management. The models generated by using applications' characteristics guide the resource management for prioritizing mixed-sensitive workload and selecting the optimal configurations among the wide range of available configurations. Optimal configuration for an application is which satisfy the application's performance requirement (latency/throughput) while minimizing the energy consumption. The characterization of applications can be done by offline data collection. In the presented approach, we consider progress rate as a characteristic for applications. The progress rate (*Prog*) of each application at any given time is defined as the ratio of the completed instructions at that particular time to the total instructions that the application should execute. In an HMP platform, each resource configuration (such as combinations of core type, DoP, and frequency) can deliver a distinct progress rate for each multi-thread application. Such progress rate variation can be a unique characteristic for applications to use for generating models and predicting the efficiency of each application in various configurations. We collect data for calculating the progress rate of applications and model generation. The data are

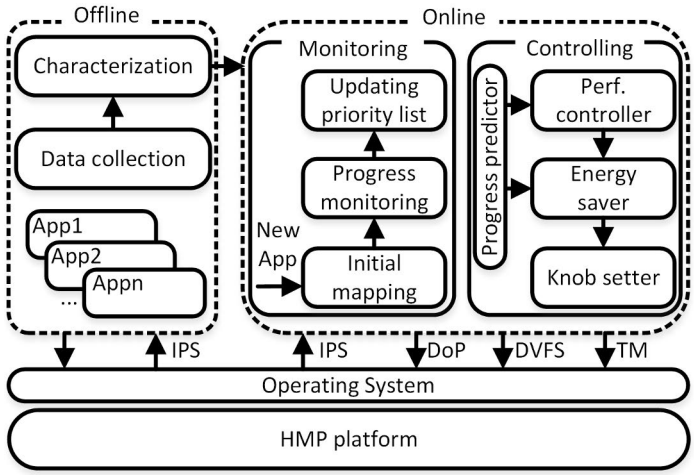


Figure 11. High-level architecture of proposed method for handling performance and energy of mixed-sensitivity workloads [58].

collected for a set of workloads from Rodinia and Parsec [59] benchmark suites. We measure the performance of applications in terms of IPS over various configurations in the Odroid XU3 HMP platform. The configurations are combinations of $< num\_cores >< core\_type >$, including 1-4 big and LITTLE cores in a fixed frequency of 1.4 GHz for the LITTLE cluster and 1.8 GHz for the big cluster. For the other frequency levels, we estimate IPS by using performance models presented in [6; 1; 53]. The progress rate can be calculated as:

$$Prog = \frac{IPS_t}{Total\_IPS} \tag{6}$$

where IPS_t is the number of instructions that have been completed until time t, and $Total\_IPS$ shows the total instructions of the application. The progress rate is considered as an application-specific unique characteristic. The progress rate for a particular application is different in different resource configurations of an HMP. We define two other characteristics by using progress rate which are *Core_Speedup* and *Parallelism_speedup*. *Core_Speedup* is the performance gain of changing the assigned configuration of an application from LITTLE (L) cluster to big (b) cluster, and the *Parallelism_speedup* is the performance gain of an application by assigning one more core to that application. Equation 7 and 8 show the calculation of these two speedups.

$$Core\_Speedup = \frac{Prog_b}{Prog_L} \tag{7}$$

$$Parallelism\_speedup = \frac{Prog_{n+1\_core}}{Prog_{n\_core}} \tag{8}$$

where $Prog_b$ and $Prog_L$ are the progress rates of an application by assigning one big and one LITTLE core, respectively, and $Prog_n$ and $Prog_{n+1}$ are the progress rates of the application when executed on n and $n+1$ cores of the same cluster, respectively. We also consider Frequency_speedup for resource management, which can be calculated as in Equation 9. The Frequency_speedup is not application specific.

$$Frequency\_speedup = \frac{freq_t + 100}{freq_t} \tag{9}$$

where $freq_t$ is the frequency at time t. *Core_speedup* and *Parallelism_speedup* are calculated offline for the applications and are used for online management.

4.2.2 Online Management

The online management consists of two major phases viz., i) *Monitoring* and ii) *Controlling*, which are explained in the following.

Monitoring

Figure 11 shows different phases of *Monitoring*: i) initial mapping, ii) progress monitoring, and iii) updating priority list.

Initial mapping: When a new latency-sensitive or throughput-driven application arrives, the first step is the initial mapping of that application. The mapper first checks the cores' statuses (as idle/busy) and assigns idle cores to the new applications based on their type. For simplicity, we select 1 big core for initial mapping of throughput-driven applications and 1 LITTLE core for latency-sensitive applications. The minimum number of cores are selected for initial mapping which can be increased at run-time in the next stages of resource management. If there are no idle core in the system, the mapper changes the core mapping in order to release at least one core. After the initial mapping, the other phases of *monitoring* and *controlling* adjust the application-to-core mapping and frequency to satisfy the requirements of all the applications.

Progress monitoring: After the initial mapping of each new application, the *Progress monitoring* periodically calculates and monitors the progress rate of running applications in terms of IPS. The period of monitoring is parametrizable. The progress rates are used for updating priority lists.

Updating priority list: Prioritizing the applications in mixed-sensitivity workload scenarios is necessary for efficient co-management of performance and energy. The limited resources (i.e., number of cores, frequency range) in embedded systems as well as the competition between mixed-sensitive applications makes prioritizing more important. For efficient co-management of performance and energy, two pri-

ority lists viz., i) performance priority list, and ii) energy priority list are required. The lists are updated by considering the progress rate of running applications. Relative Progress rate (RP) is defined in Equation 10 and 11 for latency-sensitive (RP_L) and throughput-driven (RP_T) applications. Each application issues a target Instruction Per Epoch (IPE) as the requirement, which should be satisfied at run-time. For latency-sensitive applications, the requirement is strict - the priority of such applications will be increased when the progress rate of the application is lower than the target requirement. In contrast, throughput-driven applications have a range of acceptable progress rates. The progress rate lower than that range lowers the user satisfaction, while the progress rate higher than the upper limit results in resource over-utilization. Thus, we define the RP of throughput-driven applications considering both the lower and upper limits as lim_l and lim_u.

$$RP_L = \frac{Prog_t}{Prog_T}, \tag{10}$$

$$RP_T = \begin{cases} \frac{Prog_t}{lim_l} & \text{if } Prog_t < lim_l \\ 1 & \text{if } lim_l < Prog_t < lim_u \\ \frac{Prog_t}{lim_u} & \text{if } Prog_t > lim_u \end{cases} \tag{11}$$

where $Prog_t$ shows progress rate at epoch t, and $Prog_T$ is the target progress rate for latency-sensitive applications. The RP value lower than 1 represents a progress rate that is lower than the target requirements. In this case, the priority of the application with $RP < 1$ will be increased in the performance priority list. The optimal value for RP is 1, in which case the applications meet the requirements within optimal energy consumption. Thus, an application with $RP = 1$ has a low priority in both performance and energy priority lists. Applications with $RP > 1$ meet their requirements with high energy consumption, thus they are assigned low priority in performance priority list, but high priority in energy priority list. The performance and energy priority lists are updated at run-time based on the RP value for every running application. These two lists contain different applications at any given time, which are sent to the performance controller and energy saver for energy-performance co-management. To prevent toggling between performance management and energy saving, a threshold is considered for switching between these two modes. For example, applications with $RP > 1.2$ are placed in energy priority list instead of $RP > 1$.

Controlling

The *controlling* phase of the proposed framework consists of i) progress predictor, ii) performance controller, iii) energy saver, and iv) knob setter, which are described in the following.

Progress predictor: The progress predictor uses applications' characteristics that

are collected offline to predict the progress rate of applications after knob setting. These predictions are transferred to *performance controller* and *energy saver* for resource allocation decisions. The *Core_speedup*, *Parallelism_speedup*, and *Frequency_speedup* are used to predict the progress rate after actuation of i) DVFS, ii) DoP, and iii) TM. The *performance controller* and *energy saver* require the predicted values for selecting the actuation knob settings for optimal performance-energy trade-offs. Thus, *performance controller* virtually i) increases frequency by one step, or ii) increases degree of parallelism by one, or iii) moves the application from LITTLE cores to big cores in order to increase applications' performance, and then requests for predicting $Prog_{pred-perf}$. The progress predictor uses the following equation for prediction.

$$Prog_{pred-perf} = Prog_t \times speedup, \tag{12}$$

where $Prog_t$ is the current progress rate, and speedup is frequency_speedup, Parallelism_speedup, or Core_speedup based on the knob which is virtually actuated by the *performance controller*. Similarly, the *energy saver* requests for predicting $Prog_{pred-en}$ after virtual decrease of DVFS level, or DoP, or migration of application from big cluster to LITTLE. The progress predictor uses the following equation for predicting $Prog_{pred-en}$.

$$Prog_{pred-en} = Prog_t \times 1/speedup, \tag{13}$$

Performance controller: The *performance controller* receives performance priority list from *monitoring* module, and then searches for a suitable actuation knob for the applications with higher priority. *Performance controller* starts with virtual actuation of DVFS, since it has the lowest overhead compared to the other actuation knobs, and receives the $Prog_{pred-perf}$ from progress predictor. The *performance controller* calculates RP for the predicted progress (RP_{pred}) based on Equations 10 and 11, where $Prog_t$ is replaced by $Prog_{pred}$. If $RP_{pred} >= 1$, the virtual actuation knob setting has satisfied the performance requirements. Else, the performance controller tries other virtual actuation knob settings until RP_{pred} becomes ≥ 1. Virtual actuation in performance controller follows this order - i) increasing the frequency step by step, ii) increasing DoP by one core, and iii) migration of the application. The migration is from LITTLE to big, or from low number of big cores to higher number of LITTLE cores (eg., 1 big to 3 LITTLE). The performance controller finds the optimal combination of knob settings through virtual actuation and checks the prediction of progress rate. This procedure is repeated for all the applications in the priority list in decreasing order of priority, until the performance priority list becomes empty. The monitoring module updates both performance and energy priority lists periodically.
Energy saver: The *energy saver* checks the energy priority list when the perfor-

mance priority list becomes empty. Similar to the performance controller, the energy saver virtually actuates DVFS, DoP, and TM to find the optimal configuration for each application in the priority list. The *energy saver* virtually decreases frequency or parallelism level, or move application from big to LITTLE and requests progress rate estimation from *progress predictor*. If the predicted value for RP is $1 =< RP_{pred} < 1.2$ for latency applications and $RP = 1$ for throughput applications, the virtual actuation knob settings are optimal. The optimal virtual actuation knob settings will be sent to *knob setter* module for real actuation in the platform. After updating the energy priority list by *monitoring* module, the energy saver repeats the procedure for the other applications in the list.

Knob setter: The *knob setter* performs the real actuation of DoP, DVFS, and TM on the HMP platform. The optimal actuation settings are decided and sent by the *performance controller* and the *energy saver*.

4.3 Evaluation

The proposed framework in this chapter is experimented on a real HMP platform for evaluation purpose. The experimental setup and the evaluation results are presented in this section based on the original publication Paper II.

4.3.1 Experimental Setup

The platform which has been used for evaluation of the presented method is described in Section 3.3.1. The proposed framework in this chapter is a Linux user-space process that is periodically invoked at run-time. The duration of the period for controlling and monitoring is parametrizable and set to 1 s for experimental evaluation. The workload scenario for evaluation purposes is a combination of 6 latency-sensitive and throughput-driven applications from Rodinia benchmark suite [50]. The applications arrive and leave the system dynamically. In the created workload scenario, at any given time, at least 2 latency-sensitive and throughput-driven applications execute concurrently.

4.3.2 Experimental Results

For evaluation purpose, the proposed framework is compared against two state-of-the-art resource management approaches viz., DyPO [21] and AdaMD[10]. DyPO uses offline data collection for maximizing performance per energy consumption of applications on the Odroid XU3 HMP platform. AdaMD uses offline generated models and online adaption for optimizing energy-performance trade-offs. These two approaches have the same objectives as our method and use the same platform for evaluation. Table 2 shows the energy savings and performance violations of the

Table 2. Comparison of our method with existing approaches [58].

Objective	DyPO	AdaMD	Proposed
Energy saving	45%	11%	22%
Perf. violation	91%	30%	14%

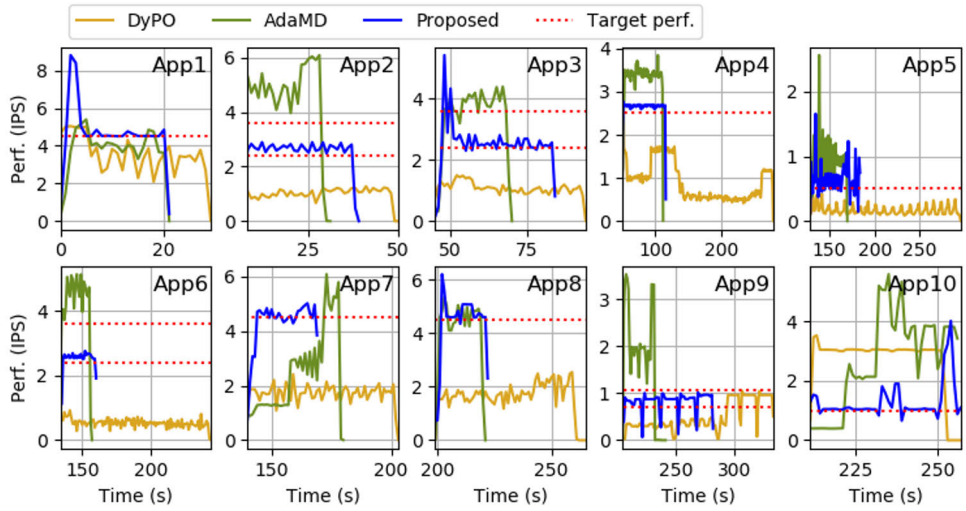


Figure 12. Performance of applications in a mixed-sensitivity workload scenario over three different resource management approaches. App1,7,8 : Heartwall, App2,3,6 : Particle filter, App4 : Streamcluster, App5 : Srad, App9 : bfs, App10 : Leukocyte tracking [58].

proposed framework against AdaMD and DyPO. The energy savings are compared to the standard baseline namely, Ondemand Linux governor [55]. As presented in Table 2, the proposed framework provides the least performance violations while saving 22% of energy. DyPO framework offers higher energy savings, however, with a 91% performance violation during run-time.

Figure 12 shows instantaneous performance of applications with 3 different resource management approaches. Each subplot shows performance of one application over DyPO (yellow line), AdaMD (green line), and the proposed method (blue line) as well as the target performance (red dotted line) of the application. The latency-sensitive applications (App1, App4, App5, App7, App8, and App10) have a strict performance target, shown in one dotted line. Throughput-driven applications (App2, App3, App6, and App9) have a target performance range, shown in two dotted lines. For all the applications, the proposed framework satisfies the performance requirements, while saving energy by keeping the performance of throughput-driven applications in the lower bound of performance target range. The AdaMD approach

over-utilizes resources resulting in higher energy consumption. AdaMD satisfies the applications' requirements by providing performance that is higher than the upper bound of target performance range. In contrast, the DyPO approach focuses on saving energy and thus violates the performance target of applications.

In summary, the proposed approach in this chapter satisfies the performance requirements of concurrent mixed-sensitivity applications, while optimizing energy consumption by extracting offline characteristics of applications and leveraging them at run-time. The experimental results show low performance violation compared to the two state-of-the-art approaches with relatively higher energy saving.

5 User and battery aware resource management

This chapter firstly presents the significance of considering user requirements, user activities, and battery pattern for resource management, and then proposes and evaluates a user and battery aware resource management framework for embedded systems. The contents of this chapter are based on Paper III and IV, which are attached at the end of this manuscript. In the following, resource management challenges in developing user and battery awareness, and the proposed solution are presented.

5.1 Background and Challenges

Increasing usage of battery powered embedded systems such as wearable devices and smart phones rise the importance of user and battery aware resource management. Considering user activities for maximizing the Quality of Experience (QoE) as well as Battery Cycle Life (BCL) is necessary for a smart device. There are challenges for handling conflicts of QoE and BCL and customizing the resource management based on user activities and requirements. The details of such challenges and the background of user and battery aware resource management are presented in subsections 5.1.1 and 5.1.2.

5.1.1 Background

The key parameters to be considered for user and battery-aware resource management include QoE, State of Charge (SOC), battery aging and BCL. The definitions and effects of each of these parameters on resource management are presented in the following.

QoE expresses a quantified level for user satisfaction while using an embedded system such as smart phone. QoE can be formulated as a weighted combination of performance an energy consumption as [60]:

$$QoE = \lambda \times Perf_N + (1 - \lambda)(1 - e_N) \tag{14}$$

where $Perf_N$ and e_N are the normalized performance and energy consumption, and λ is the weight of performance and energy at run-time which quantifies the user

preference. The user preference can change at run-time based on user activity. For example, when the level of battery charge (**SOC**) is low, the preference of user is saving energy, making energy as the higher priority metric than performance. In contrast, running a high intensive application by user may lead to higher preference for performance.

The user's preferences and the plug-in/out activities of user has direct effect on battery aging. **Battery aging** is defined as the loss of usable capacity of battery over time due to the loss of active materials [61; 62]. Battery aging is influenced by i) average SOC, ii) SOC swing, and iii) battery temperature [63]. Low temperature, average SOC, and SOC swing results in lower battery aging and higher BCL. **BCL** is the number of charge/discharge cycles before the battery fails to operate satisfactorily [63]. There are several state-of-the-art articles which model battery aging [64; 65] and propose methods for decreasing aging effect. Some of these approaches use aging-aware charging, which predicts when a user plug-out the device and manage the SOC to reach 100% right before plug-out, instead of remaining 100% for a long time. Such smart charging leads to decreasing the average SOC and thus decreasing the aging effect [66; 63; 67]. Lowering the average SOC during charging is acceptable for a user, however low SOC level at run-time may lower user satisfaction and decrease the QoE. Further, increasing QoE may require maximizing performance, leading to an increase in temperature and SOC swing - affecting the battery aging. Therefore, there is conflict between the objectives of decreasing battery aging (maximizing BCL) and increasing QoE.

State-of-the-art approaches [66; 61; 67] for increasing BCL propose various charging protocols to decrease the average SOC. Such approaches ignore the run-time activities which have considerable impact on battery aging. The user activities such as plug-in/plug-out and execution of variable workloads affect SOC pattern at run-time. Learning each user's activities can guide resource management for maximizing QoE as well as BCL. Each user has specific habits which requires individualized resource management policies. In this chapter, we propose a framework which considers users' activities as well as battery pattern for an optimal resource management.

5.1.2 Challenges

The first-order objective in interactive embedded systems such as smartphones and tablets is user satisfaction. Therefore, user-aware resource management is necessary for the efficient functionality of interactive embedded systems. Further, these devices are mainly powered by batteries, thus battery awareness is another necessary objective that should be considered in resource management. Providing both user and battery awareness rises two major challenges for resource management, which are listed in the following.

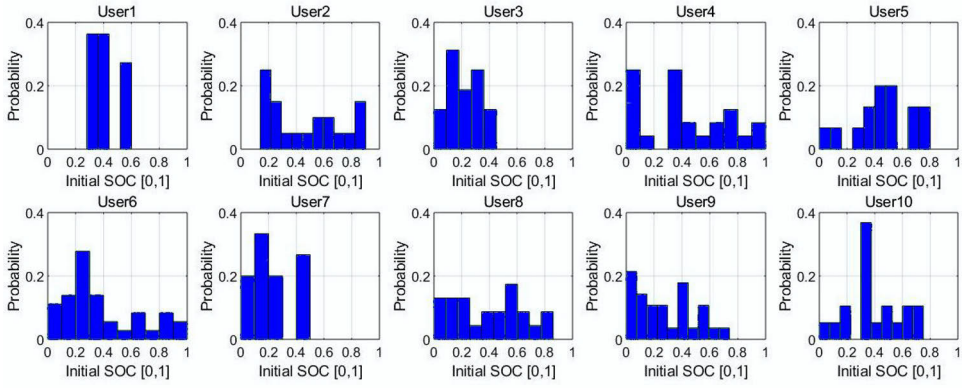


Figure 13. Plug-in probability pattern of 10 different users [68]

1. **User and battery objectives may be conflicting at run-time.** User requires high level of QoE which translates to high performance or low energy consumption depending on the user preferences. When a user executes a high intensive application demanding high performance, it leads to higher on-chip temperatures and SOC swing, increasing the battery aging. Therefore, maximizing QoE may lead to battery aging and create conflicting objectives for resource management. Prioritizing between these two objectives at run-time and providing efficient trade-off makes the resource management challenging.

2. **Each user has their own individual activities which affect the QoE and BCL.** The user's habits has major effect on battery aging. For example, when user plug-in the device overnight, it causes SOC level to remain at 100% during the night, increasing the average SOC and aging. Further, the initial level of SOC in which the user plug-in the device is different for different users, affecting average SOC as well as aging. Figure 13 shows the probability of plug-in event in different initial SOCs for 10 users. Learning such individual patterns is challenging, however improves the resource management.

An intelligent resource management can handle the aforementioned challenges by using machine learning techniques and run-time controlling. In this chapter, we present such resource management approach as User and Battery Aware Resource management (UBAR). The UBAR learns users' individual plug-in/out patterns and customize the resource management based on that to increase QoE and BCL.

5.2 Solution: UBAR

UBAR framework uses battery and user models for predicting user activities and battery patterns. By considering the predicted patterns, UBAR adjusts the resource

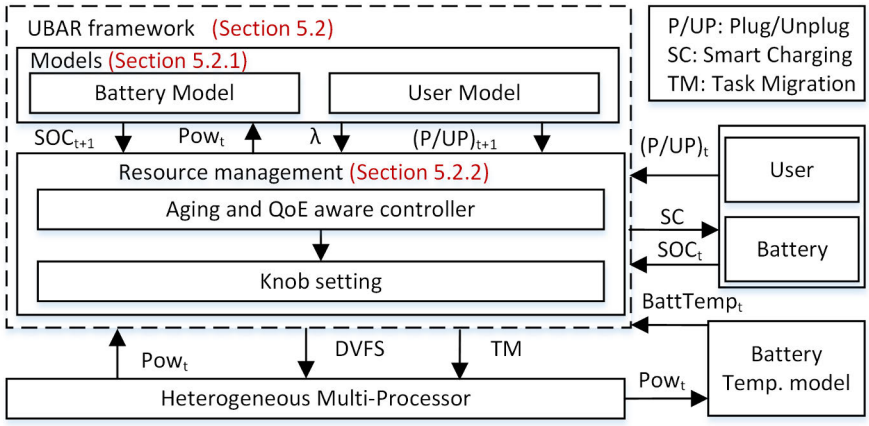


Figure 14. High level overview of UBAR framework [68].

management in order to satisfy the user's expectation while minimizing the aging effect. UBAR uses offline data collection as well as an online adaption for learning individual activities of users. Figure 14 shows the high level overview of the UBAR framework which consists of two major components viz., i) predictive models, and ii) resource management. The details of models and resource management are explained in the following subsections.

5.2.1 Predictive Models for Battery and User

The UBAR framework leverages i) battery model and ii) user prediction model for predicting the SOC pattern and user activities. The battery model consists of charging, discharging and temperature models, and the user model consists of plug-in and plug-out prediction. Further, aging model is used for evaluating the battery aging.

Battery Model

The battery model which is used in the UBAR framework considers various aspects of the battery viz., i) charging phase, ii) discharging phase, iii) capacity fading [69], iv) thermal coupling of CPU and battery [70], and v) battery aging. By considering the aforementioned aspects, we provide a comprehensive battery model for predicting SOC.

Charging model: The charging model is built using offline data which is collected from real batteries in charging phase. Linear regression model is used for designing charging model as in Equation 15.

$$SOC_t = \gamma \times t + SOC_0 \tag{15}$$

where γ is the regression coefficient which is set to 0.016, t represents the time which

passes (in second) after plug-in, and SOC_0 is the SOC level in which the user plug-in the device. Equation 15 provides a normal charging phase of battery, however some of the state-of-the-art articles propose *Smart Charging (SC)* [66] which delays charging over night for decreasing average SOC to lower aging. We added this option to UBAR, referred to as UBAR+ for decreasing the aging effect. In UBAR+ framework, for decreasing the aging effect, we use the SC as a control knob which predicts plug-out event and delays the charging accordingly to minimize the SOC average and aging effect.

Discharging model: The discharging model which is used in UBAR and UBAR+ is based on the model proposed in [69], which predicts the SOC level online. Equation 16 represents estimation of SOC level in the next cycle where the duration of each cycle is Δt.

$$SOC(t + \Delta t) = \frac{E(t + \Delta t) \times 100}{E_T} \tag{16}$$

where $SOC(t + \Delta t)$ and $E(t + \Delta t)$ represent SOC level and the energy remaining at time $t + \Delta t$, and E_T shows total energy of battery when it is fully charged (e.g. 28800 J in this chapter). $E(t + \Delta t)$ can be calculated as:

$$E(t + \Delta t) = E(t) - E_c \tag{17}$$

where $E(t)$ is the energy remaining at time t and E_c is the energy consumption (in J) over Δt, which can be calculated as:

$$E_c = \Delta t \times P_{device} + E_{loss} \tag{18}$$

where P_{device} is the power consumption of the device over Δt and E_{loss} is the internal energy loss of battery caused by rate capacity effect. The E_{loss} calculation is presented in Equation 19.

$$E_{loss} = \Delta t \times (i_b^2 R_{total} + i_b.v_{OC}.(1/\eta(i_b) - 1)) \tag{19}$$

where i_b , R_{total}, and v_{OC} are discharging current (amp), total internal resistance (ohm), and open circuit terminal voltage (volt) of battery respectively, and $\eta(i_b)$ shows battery discharging efficiency. The details of calculating each of the above parameters are available in the original Paper III which is attached at the end of this manuscript.

Battery temperature model: The temperature model which is used in UBAR framework is based on the model in [70] which considers coupling between the battery and the CPU. The model estimates battery temperature by considering environment temperature (T_{env}), and power consumption of battery (P_{bat}) and CPU (P_{cpu}). This model is used in the aging model for evaluation in this chapter. Equation 20 shows

such temperature estimation.

$$T_{bat} = T_{env} + \frac{R_{cpu-env}R_{bat-env}}{R_{bat-env} + R_{cpu-bat} + R_{cpu-env}}.P_{cpu} + \frac{R_{cpu-env}R_{bat_env} + R_{cpu-bat}R_{bat_env}}{R_{bat-env} + R_{cpu-bat} + R_{cpu-env}}.P_{bat} \quad (20)$$

where R_{i-j}s are the thermal resistances between i and j, i and j can be *CPU*, *environment* (env), or *battery* (bat). The value of resistances are available in [70] and Paper III, which is attached at the end of this manuscript. P_{cpu} can be measured at run-time by using the sensors in Odroid-XU3 platform which is used in the experiments in this chapter. P_{bat} can be calculated based on P_{cpu} by using the experimental results in [71] which shows P_{cpu} is on average 15% of P_{bat}.

User Prediction Model

User prediction model consists of plug-in and plug-out prediction which are designed based on collected data from real smartphone users.

Data collection: The data is collected from four users over 1 month to 1 year period by using *Battery Log* application [72]. The collected data consists of the users' actions i.e., plug-in/out as well as timestamps and SOC levels. Half of the data is used for training a model which predicts the SOC level or the timestamp at which a user plug-in/out the device, and the other half of data is used for evaluation.

Plug-in prediction: The plug-in prediction model predicts if the user plug-in the device at a given time and SOC or not. The model is trained with the data collected offline and update at run-time based on users' actions. The model is designed using Naive Bayes theorem [73] which calculates probabilities based on the frequency of each value in data set, and then calculates conditional probability as in Equation 21.

$$P(A|B) = \frac{P(B|A) \times P(A)}{P(B)} \quad (21)$$

where $P(A|B)$ is the probability A occurring given that B has occurred. Based on the above equation, we calculate two probabilities PG_{SOC} = (Plug-in)|(SOC=b) and PG_{time} = (Plug-in)|(time=t) by considering $P(A) = P$(Plug-in==true) and $P(B) = P$(SOC=b) or $P(B) = P$(time=t). The b is a SOC level from 0 to 100%, with step size of 10%, and t is the time of a day from 0 to 23h, with step size of 1h. $P(A)$ and $P(B)$ can be calculated based on the data collected offline, by considering the frequency of each events in the data set. $P(A|B)$ can be calculated based on Equation 21. In addition to PG_{SOC} and PG_{time}, we calculate special probabilities (PS) for each user which are calculated by considering the most recent activities of the users. The PS for a time or SOC level is set to P_{max} when the user plug-in their device mostly at that time or SOC level. The P_{max} can be

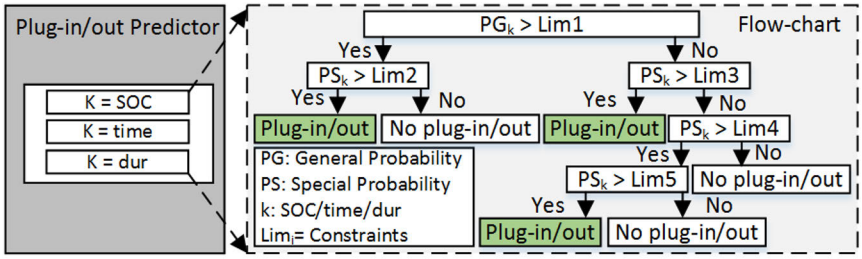


Figure 15. Making decision flow-chart for user prediction model [68].

adjustable, however in this work we set it to 0.7. The combination of PG and PS make the final prediction based on the flow-chart shown in Figure 15. As shown in the flow-chart, there are 5 constraints viz. $Lim_1, Lim_2, Lim_3, Lim_4$, and Lim_5 which guide the decision making.The flow-chart starts with comparison of PG and Lim_1 and then fine-tunes by comparing PS with the other constraints. The green boxes show the plug-in/out event occurances. In this work, the constraints set to $Lim_1 = 0.5, Lim_2 = 0.3, Lim_3 = 0.5, Lim_4 = 0.25$, and $Lim_5 = 0.4$.

Plug-out prediction: The plug-out prediction model is designed similar to plug-in prediction model, and calculates the plug-out event in a given SOC level or given charging duration. In the plug-out-prediction model $P(A) = P(\text{Plug-out=true})$ and $P(B) = P(\text{SOC=b})$ or $P(B) = P(\text{dur=td})$, where td is the charging duration time. PG_{SOC} and PG_{dur} are calculated by using the aforementioned $P(A)$ and $P(B)$. PS_{SOC} and PS_{dur} are also calculated similar to the plug-in probabilities and then by using the same flow-chart in the Figure 15, the plug-out event can be predicted.

Aging Model

For evaluating the BCL in this chapter, we use an aging model which considers electrochemical aging process over time [64]. The input parameters of the model are battery temperature, SOC swing and average SOC. The model estimates the damage of battery in each charging and discharging cycle. The details of calculation are available in the original Paper III which is attached to this manuscript. By accumulating the damages and subtracting that from the initial capacity, the remaining capacity can be calculated. BCL is the number of years taken to the capacity of the battery reach 80% of the initial capacity [66].

5.2.2 User and Battery Aware Resource Management

The resource management objective is maximizing BCL and QoE by actuating DVFS, TM, DoP, and application mapping. The resource management decisions are guided by i) current and predicted state of the device i.e., plugged, unplugged, charging,

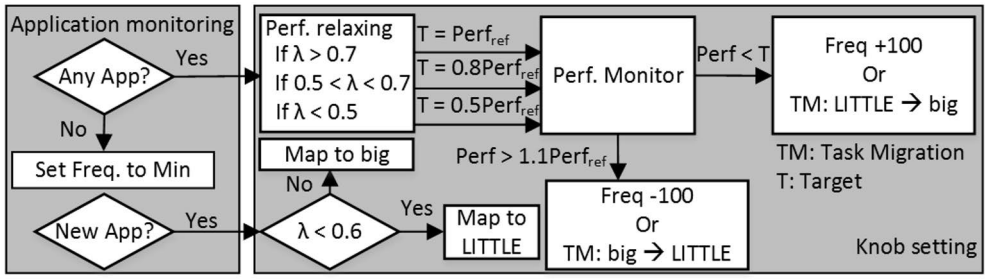


Figure 16. Resource management process in UBAR [68].

ii) current and predicted SOC, iii) set of running applications, and iv) instantaneous power consumption.

In practice, there is a parameter λ which is generated based on the SOC level and plug-in prediction at run-time, and represents preference on energy saving or performance increasing. λ is used in Equation 14 for calculating QoE level of users. λ is a value between 0 and 1 - a higher value denotes more preferences on higher performance (based on Equation 14). Thus, when user prefers energy saving (e.g., in lower SOC level), the λ is lower. Therefore, Equation 14 represents the relation between λ and SOC level.

$$\lambda = \alpha \times \frac{SOC}{100} \tag{22}$$

where α is a parameter which set based on plug-in prediction at run-time. No plug-in prediction leads to $\alpha = 1$, and plug-in prediction causes an increase in α up to 1.6 [74]. This leads to an increase in λ, showing when a plug-in event is more probable.

The resource management actions viz. i) decreasing power and ii) increasing performance are guided by λ. Each of this actions control the QoE level and BCL. Decreasing power leads to lower SOC swing, lower temperature, and thus higher BCL. Increasing performance satisfy the user requirement and thus increases QoE. The resource management actions are executed by actuating the knobs available in the platform. DVFS, task migration, and DoP are the knobs which are used in UBAR. The suiatable combination of knobs based on (λ) can maximize the QoE. Furthermore, by decreasing the performance and power consumption, the aging effect decreases and thus the BCL increases.

Figure 16 shows the overview of resource management in actuation of knob setting. As shown in Figure 16, the *Application monitoring* module monitors each arriving application. If no application exists, the frequency set to minimum for decreasing aging effect. Else, the *knob setting* maps the application to big or LITTLE cores based on λ. After initial mapping, the *performance monitor* adjusts the frequency, and the number and type of cores based on the performance references for

Table 3. Summary of the workload which is used for UBAR evaluation [68].

Application	Category	Summary
Dijkstra	Network	Constructs a large graph and then calculates the shortest path between every pair of nodes
Patritia	Network	Creates data structure for representing routing tables in network applications
Sha	Security	Secure hash algorithm
Rijndael	Security	An advanced and standard encryption and decryption method
Qsort	Automotive Control	Sort a large array of strings into ascending

applications. There is a *Performance relaxing* component which relaxes the performance target ($Perf_{ref}$) of each application based on the λ value. *Performance relaxing* decreases the $Perf_{ref}$ at run-time based on the user preferences. When the user prefers power saving over performance i.e., $\lambda < 0.5$, *Performance relaxing* set the Target = $0.5 \times Perf_{ref}$; otherwise the Target (T) is set to $0.8 \times Perf_{ref}$ if $0.5 < \lambda < 0.7$. Then, the *Performance Monitor* increases or decreases the frequency by one step (e.g., 100 Mhz), or moves the application based on the current performance of applications (Perf) and current target.

5.3 Evaluation

In this section, we present the experimental setup which is used for evaluating UBAR and UBAR+ framework as well as the experimental results.

5.3.1 Experimental Setup

Details of experimental setup i.e., platform and workload which have used for evaluation is presented in following.

Platform

The platform which has been used for evaluation of the proposed method in this chapter is the HMP platform of Odroid XU3 which is also used in Chapter 3 and presented in Section 3.3.1. The proposed framework is a Linux user-space process which is periodically invoked at run-time. The duration of resource management and plug-in/out prediction are parametrizable. In experiments, we set them to 0.5 s and 2 s, respectively. The battery model which is used for evaluation is 4 V, 2,000 mAh with 28,800 J capacity.

Workload

The workloads which are used for evaluation is a set of applications from Mibench [75] benchmark suite as foreground applications, and audio and video playback, and web surfing are chosen as background applications. Table 3 summarizes the Mibench ap-

Table 4. Summary of the compared approaches against UBAR [68].

Technique	Power	Perf.	Energy	QoE	Battery	Plug-in/out pattern	Rate capacity	Temp.	BCL
HP	X	✓	X	X	X	X	X	X	X
LP	✓	X	X	X	X	X	X	X	X
SC[66]	X	X	X	X	✓	✓	X	X	✓
Od	✓	✓	X	X	X	X	X	X	X
DyPO[21]	✓	✓	✓	X	X	X	X	X	X
BUQS[76]	✓	✓	✓	✓	✓	X	X	X	X
UC[74]	✓	✓	✓	✓	✓	✓	X	X	X
UBAR+	✓	✓	✓	✓	✓	✓	✓	✓	✓

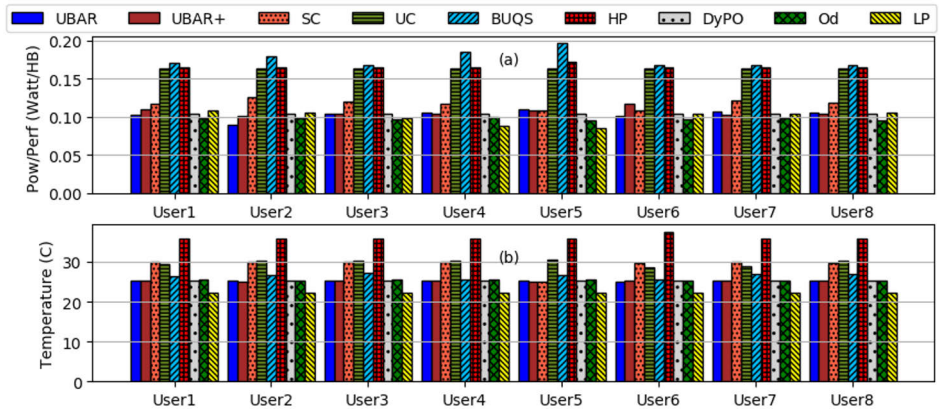


Figure 17. Comparison of (a) power/performance and (b) temperature of battery for 8 different users [68].

plications which are used in experiments. The applications arrive dynamically and execute concurrently. The applications are enhanced by Heartbeat API [25] for measuring performance. Each application issues Heartbeat periodically which can be measured at run-time for resource management.

Methodology

For evaluation purposes, we use the event patterns of 8 different smartphone users which consists of plug-in/out events based on the time and SOC level. The data of the event patterns is used for modeling user and battery in the experiments. The event patterns of users are different based on their plug-in/out habits during day (details are presented in the original paper [68] at the end of this manuscript). The experiments are simulated for 200 hours of using smartphones.

For evaluation purpose, we compare UBAR and UBAR+ against state-of-the-art approaches i) DyPO [21], ii) BUQS [76], iii) UC [74], iv) SC [66], and v) LINUX governor viz., Low Power (LP), High Performance (HP), and Ondemand. The summary of the aforementioned approaches is presented in Table 4. The Table shows the

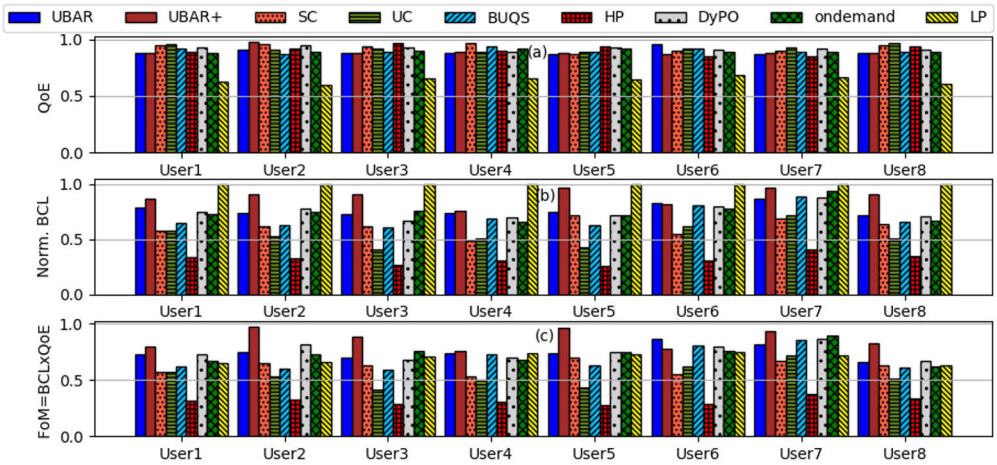


Figure 18. Comparision of (a) QoE, (b) BCL, and c) BCL×QoE for 8 different users [68].

parameters which are considered or ignored by each of the approaches. The last line of the Table 4 shows the proposed framework in this chapter which considers all the parameters that are mentioned in the table.

5.3.2 Experimental Results

In this section, the comparison of the proposed framework against state-of-the-art approaches in terms of i) power/performance, ii) temperature, iii) QoE, iv) BCL, and v) QoE×BCL is presented. Figure 17(a) shows the *power/performance* of various resource management approaches for 8 different users. The lower amount of *power/performance* is better in terms of both saving energy and satisfying performance requirement of applications. The value of *power/performance* for UBAR and UBAR+ is the lowest for all the users. The *LP*, *DyPO* and *od* frameworks also provide relatively low *power/performance*, however, Figure 18 shows UBAR and UBAR+ offers relatively better trade-off between BCL and QoE. UC, BUQS and HP have relatively high *power/performance* due to high power consumption. Figure 17(a) shows the trend of *power/performance* variation for different users are not dramatically different, which shows users' habit does not have major impact on *power/performance*. Similarly, the temperature is relatively low for all the users with UBAR and UBAR+ strategies. The *LP* approach leads to the lowest temperature which is suitable in terms of lowering aging effect. However, as shown in Figure 18(b), the amount of QoE is low for *LP*. *HP* framework has the highest temperature due to using the highest level of frequency regardless of users' and applications' requirements.

Figure 18 shows the comparison of main objectives of resource management in

this chapter. Figure 18(a) and (b) show that the *LP* approach provides the highest BCL but the lowest QoE by setting the frequency level to the lowest. However, we require a suitable trade-off between QoE and BCL by managing the actuation knobs based on the requirements at run-time. The UBAR and UBAR+ provide relatively high QoE while maximizing the BCL. Figure 18 shows UBAR+ offers the highest BCL after LP for all the users. Therefore, our method increases the BCL while still maintaining the QoE higher than 0.8 for all the users. For better evaluation, we propose a Figure of Merit ($FoM = BCL \times QoE$) and compare it for various approaches. As shown in the Figure 18(c), UBAR+ provides relatively high FoM for all the users. However, FoM is variable depending on users' habits. For example, UBAR+ has the highest FoM (high difference compare to the other approaches) for User 2 and 5. However, for User 6, the FOM of UBAR+ is even lower than BUQS. The reason for this variation is the predictability of users' habits. The users with higher predictable plug-in/out activities (User 2, 5) have better results in terms of FoM.

6 Goal Driven Autonomy for a self-aware resource management

In the previous chapters, we presented resource management approaches for handling fixed objectives. In chapter 3 and 4, the primary objective is performance satisfaction and the secondary objective is energy saving. In chapter 5, the objectives are QoE and BCL which are influenced by performance, power, and temperature. In this chapter, we present a generic resource management approach which can be used for handling multiple objectives which may change dynamically at run-time. The contents of this chapter are based on Paper V, which is attached at the end of this manuscript.

6.1 Challenges of Multi-objectives Resource Management

In this section, the challenges of multi-objective resource management are presented by an example. In the example, the state and objectives of 4 entities viz., i) system, ii) user, iii) battery, and iv) application are changing dynamically. The aforementioned entities have priorities in resource management in decreasing order. We consider two frameworks viz., i) with autonomy and ii) without autonomy to emphasize the need for autonomous resource management for handling multi-objectives. These two frameworks use i) *Low power* [9] and ii) *High performance* [77] policies for resource management. The *Low power* policy respects power constraints as a first-order priority while considering performance requirements as the second priority. In contrast, the first-order priority in high performance policy is meeting the performance requirements of applications. The example is presented in Figure 19, which shows the power consumption and corresponding performance (solid lines) for two applications in the two aforementioned frameworks. The dashed lines show Thermal Power Design (TDP) as power threshold (red) and performance references for the two applications (green and blue). Figure 19 also shows *state* i.e., the current status of the entities, and *urgency* i.e., the extent to which entities' objectives are met. The green to red boxes under urgency columns show low to high urgency. The states are i) operating under power budgets (shown in ✓) and violating power budget (shown in X) *for system*, ii) high performance (shown in H) and power saving (shown in L) command *for user*, iii) high SOC (H) and low SOC (L) *for battery*, and iv) high (H)

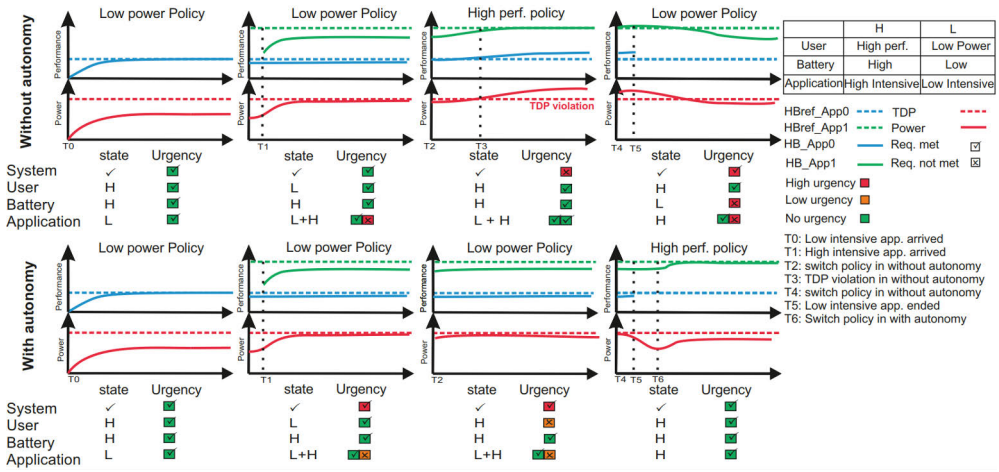


Figure 19. Motivational example for expressing the challenges of multi-objective resource management [8].

and low (L) performance requirements *for application*.

In Figure 19, at T0, a low intensive application arrives with low performance requirement, and the user command is high performance. In this situation, the urgency of all the entities are low and the requirements are met by using *Low power Policy*. The low power policy keeps the frequency low and assigns low power resources to the application. At T1, a high intensive application arrives increasing the urgency of the application, and the user command switches to power saving. In the autonomous framework, as the power consumption is near TDP, the urgency of the system also increases. Therefore, the decision of autonomous framework is to assign highest priority to the system entity, as well as user, and neglect the application's requirements (continue to use low power policy). At T2, the user command changes to high performance. This leads the non-autonomous framework to assign more resources to the application (by using high performance policy), responding to the application's urgency and user requirement. Such decision leads to power violation, using the non-autonomous framework, at T3. In contrast, the autonomous framework predicts such violation and prevents it. At T4, after power violation, the non-autonomous framework switches the policy to low power for decreasing the power consumption. This leads to lowering the performance of high intensive application, resulting in an increased urgency of the application entity. At T5, the low intensive application completes execution. At T6, the power consumption becomes low enough that the autonomous system switches to high performance policy for satisfying the performance requirements of high intensive application. A non-autonomous system leads to a constant switching between different policies to satisfy various objectives at runtime. On the other hand, the autonomous system intelligently considers the priorities

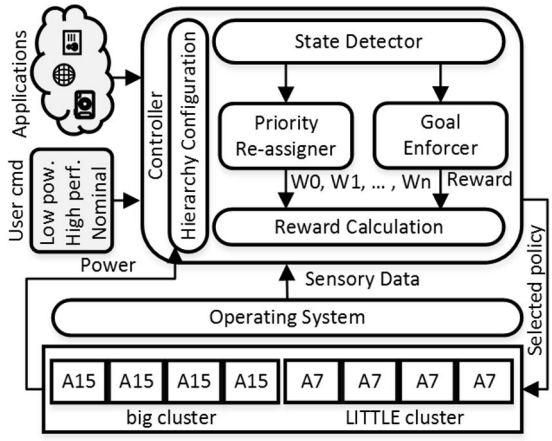


Figure 20. High level architecture of GDA framework for on-chip resource management [8].

of entities and predicts possible violations to potentially make optimal resource allocation decisions.

6.2 GDA for Multi-objective Resource Management

For handling the complexity of multi-objective resource management, we present Goal-Driven Autonomy (GDA) in this section. In this section, we leverage GDA conceptual model, which has been used in the artificial intelligence and robotics field [78; 79; 80] and adapt that for on-chip resource management purposes. GDA is used for goal management and guiding decision making in resource management. The high-level architecture of the GDA framework for on-chip resource management is presented in Figure 20. The Figure shows a controller which interacts with i) applications, ii) user, and iii) a Heterogeneous multiprocessing system (HMP). The controller receives sensory data from HMP and the requirements from applications and user then decides on resource management policy. The details of each sub-component in the Figure 20 are explained in the following sub-sections.

6.2.1 Background of Goal Driven Autonomy

Goal-Driven Autonomy (GDA) is a conceptual module that responds to unexpected events in a complex and dynamic environment. The GDA model expands Nau's approach in [81] as online planning by identifying specific subtasks within the controller. The controller interacts with an execution environment that monitors the world state and a planner which generates sequences of actions to achieve the current active goal. The GDA agent has a planning component to generate an expectation of environment state after executing each action in the execution environment.

GDA, as presented in [78] begins with a primary goal and then executes the necessary actions to attain the current goal by monitoring and interacting with the environment. When there is a conflict between the expected state and the present state, a goal formulator generates new goals. Despite the fact that GDA has never been used for on-chip resource management, there have been various studies on GDA and goal management in the other fields [79; 80; 82] such as psychology and artificial intelligence. A reactive goal management approach is presented in [83] to manage and nominate goals for an artificial agent by prioritizing them. Another framework as LGDA (Learning GDA) in [79] uses Q-learning for learning the goal selection function of an intelligent agent. Further, in the gaming scenarios, GDA has been used for controlling unexpected events [79; 80; 83].

6.2.2 State Detector

State Detector dynamically monitors the major entities which affect resource management and generates a state vector as $S = (s_1, s_2, s_3)$. We consider three major entities viz., i) system, ii) user, and iii) application. s_1, s_2, and s_3 represent the entities' requirements which are power state, performance state, and user command. The system states are i) violation ($P > TDP$), ii) potential violation ($TDP > P > 0.8 \times TDP$), and iii) no violation ($P < 0.8 \times TDP$). The user states are i) high-performance and ii) power-saving which are interactive user commands at run-time. The application states represent the extent of performance violation of the currently running applications by considering their requirements which are i) violation (more than half of the applications violate their requirements), ii) potential violation (less than half of the applications violate their requirements), and iii) no violation (no application violates its requirement). The state vector is transferred to the goal enforcer and priority re-assigner for further decisions.

6.2.3 Hierarchy Configuration

The *Hierarchy Configuration* defines a hierarchy for the entities which affect the priority of that entity. The hierarchy can be fixed [8] or configurable at run-time. We consider two frameworks with fixed (GDA) and configurable (CGDA) hierarchy levels. In the fixed hierarchy, system's goals are primary, user's goals are secondary, and applications' goals are tertiary. In the configurable hierarchy, we analytically and experimentally estimate hierarchies for system and application at run-time based on the Equation 23 and 24. A higher value of hierarchy represents higher priority for the goal generated by that entity.

$$H_{sys} = \begin{cases} 2.5\alpha & \text{if } P > 0.9\,TDP \\ 1.25\alpha & \text{if } 0.9\,TDP > P > 0.8\,TDP \\ \alpha & \text{if } P < 0.8\,TDP \end{cases} \tag{23}$$

$$H_{app} = (1 + \frac{n_{viol}}{n})C_{perf} \tag{24}$$

where H_{sys} and H_{app} are the *hierarchy* of *system* and *application*, P is power consumption at run-time, TDP is the power threshold which can be set based on the experimental setup, n is the total number of running applications, n_{viol} is the number of applications that violate the target performance, C_{perf} is the constant coefficient for adjusting hierarchy, and α is an adjustable value for determining the hierarchy level of *system*. Higher power consumption increases the system level hierarchy and higher number of violated applications increases the application level hierarchy. We experimentally set the α and C_{perf} to 2 for having equal hierarchy when the power is low enough and no application violates its requirements.

6.2.4 Priority Re-assigner

The *Priority Re-assigner*, updates the priority of goals periodically at run-time. The priority depends on two factors: i) hierarchy of goal, and ii) urgency. Urgency is defined as the extent of violation of a measured parameter e.g., power and performance. Equation 25 shows urgency calculation:

$$U_{pow} = \frac{P_{curr}}{P_{ref}}, U_{perf} = \frac{Perf_{max} - Perf_{curr}}{Perf_{max} - Perf_{ref}} \tag{25}$$

where U_{pow} and U_{perf} are urgency of power and performance which represents urgency of system and applications, P_{curr} and $Perf_{curr}$ are the current power consumption of the chip and current performance of application, P_{ref} is the power threshold for violation (TDP), $Perf_{ref}$ is performance reference for an application, and $Perf_{max}$ is maximum performance requirement. $U > 1$ shows violation of the parameters (power or performance). The value of $Perf_{ref}$ is determined at run-time based on *User Cmd*. When *User Cmd* is *power saving*, $Perf_{ref}$ is set to $Perf_{min}$, which is the minimum requirement of the application. When the *User Cmd* is *high performance*, $Perf_{ref}$ increases and is set to $\frac{Perf_{max}+Perf_{min}}{2}$. After urgency calculation, the priority of each of the aforementioned parameters can be calculated by Equation 26 using their hierarchy.

$$P_{pow} = U_{pow} \times H_{sys}, \quad P_{perf} = U_{perf} \times H_{app} \tag{26}$$

where P_{pow} and P_{perf} are the priority of power and performance parameters which represents system and applications priority.

6.2.5 Goal Enforcer

Goal Enforcer leverages Q-learning method for selecting resource management action based on the current state, which is issued by the state detector. Q-learning is a

reinforcement learning method [84] which consists of:

- State space S: a set of environment's states. In GDA framework *state detector* periodically generates the set of states.
- Action space A: a set of performable actions. In resource management context, an action can be knob setting such as DVFS, TM, and DoP, or a combination of several knob settings as a resource management policy. In GDA framework, the action is choosing low power and high performance policy which combines DVFS, and TM.
- Reward R: the reflection of success or failure of an action in meeting the target goals. Subsection 6.2.6 explains *Reward Calculator* in the GDA framework.

The Goal Enforcer monitors the set of states at any given time and selects the action with the highest reward value for that state. The rewards are calculated and stored by *Reward Calculator* after each action. In the initial case, when there is no stored reward, a random action is selected by *Goal Enforcer*.

6.2.6 Reward Calculator

The *Reward Calculator* estimates the efficacy of actions by using a weighted reward function as follows:

$$Reward = W_0 \times R_0 + W_1 \times R_1 + W_2 \times R_2 + ... + W_n \times R_n \tag{27}$$

The rewards can be calculated for measurable parameters i.e., power and performance, and the weights determined by *Priority Re-assigner*. In the GDA framework, reward is calculated as follows:

$$Reward = W_{pow} R_{pow} + W_{perf} R_{perf} \tag{28}$$

where R_{pow} and R_{perf} are calculated by Equation 29, 30. W_{pow}, W_{perf} are P_{pow}, P_{perf} which are calculated by Equation 26.

$$R_{pow} = \frac{P_{ref} - P_{curr}}{P_{ref}} \tag{29}$$

$$R_{perf} = \frac{\sum_{i=1}^{n} \frac{Perf_i - Perf_{imin}}{Perf_{imax} - Perf_{imin}}}{n} \tag{30}$$

where n indicates the number of running applications, and i represents each application. Higher reward value shows that the action satisfies the corresponding parameters. After execution of each action, the *Reward Calculator* calculates the reward and stores it in a table for the state-action pair.

Table 5. Summary of the used applications in the experiments.

Application	Computation	Domain	Arrival time (s)
djkstra-small	Graph traversal	network	0
basicmath	Mathematical computation	automotive	2
bitcount	Binary counting	automotive	9
sha	Hash algorithm	security	19, 21
qsort	Array sorting	automotive	22, 23

6.3 Evaluation

In this section, we present the experimental setup in which we evaluate the GDA and CGDA framework as well as the experimental results.

6.3.1 Experimental Setup

Platform

The platform which has been used for evaluation of the GDA and CGDA frameworks is the HMP platform of Odroid XU3 which is presented in Section 3.3.1. Each of the proposed framework is a Linux user-space process which is periodically invoked at run-time. For experimental purpose, we set the period to 1 s.

Workload

We use a set of applications from the MiBench benchmark suite [75] as well as synthetic micro-benchmarks for evaluation of GDA and CGDA frameworks. The chosen set of applications represents the behavior frequently encountered in heterogeneous embedded systems. The MiBench applications are summarized in Table 5. The applications are enhanced with heartbeat API [25] for performance measuring, and the heartbeats are registered every single run of each application.

6.3.2 Experimental Results

Table 6 presents comparison of the GDA and CGDA frameworks against two state-of-the-art resource management policies viz. i) LP policy [9], and ii) HP policy [77]. LP is a fixed objective policy that minimizes power consumption by compromising performance. LP assigns higher priority for minimizing power consumption and lower priority for performance satisfaction. On the other hand, HP assigns a higher priority for the satisfaction of applications' requirements compare to minimizing power consumption. Both LP and HP are multi-objective policies that use DVFS and TM for resource management. The objectives and their priorities are fixed at

Table 6. Comparison of proposed solution with existing approaches

Tech.	Obj	Cmd	Pwr viol.	Perf. viol.	Avg. pwr (W)
LP [9]	Power	X	0%	27%	2.86
HP [77]	Perf.	X	3%	0%	3.7
GDA [8]	Dynamic	✓	0%	14%	3.1
CGDA	Dynamic	✓	1%	2%	3.4

design time for HP and LP. However, GDA and CGDA handle dynamic objectives at run-time and also considers user command as an input for resource management. The highlighted cells in the Table 6 show the efficiency of GDA and CGDA frameworks in handling conflicting objectives. LP policy is suitable for low power objective and has no power violations, while HP policy is suitable for respecting performance requirements. Each of these two policies satisfies one objective and neglect the other. However, GDA and CGDA consider the objectives dynamically at run-time as well as the user command. The GDA framework decreases the performance violation compared to the LP, and power violation compared to the HP. GDA and CGDA consider the workload variation at run-time and prioritize the requirements based on that. These two frameworks switch between LP and HP by considering the priority of objectives and thus, result in lower power and performance violation compared to the fixed objective policies. The CGDA framework significantly decreases the performance violation compare to the GDA approach with negligible power violation. The reason is using of configurable goal hierarchy by CGDA instead of fixed goal hierarchy in GDA. The configurable goal hierarchy reflects the dynamics of performance requirements and available power budget better than the fixed goal hierarchy. Therefore, resource management can fully utilize the power budget and satisfy performance requirements higher.

Figure 21 shows the instantaneous power consumption of the platform when using three different resource management approaches over the same workload scenario. The workload is a combination of synthetic micro-benchmarks which arrive and exit the system dynamically. For thermal safety, TDP is set to 4 W in this experiment. The vertical dashed lines in Figure 21 shows the point when user command is switched. The red dashed line shows TDP. The workload scenario consists of high-intensive applications followed by low intensive applications. The HP policy leads to power violation at the beginning and high power consumption when the workload is low intensive (from 42 s to 81 s). The LP policy keeps the power consumption under TDP, however as reported in the Table 6, it causes high rate of performance violation. However, the GDA framework considers the user command and workload intensity to switch between the proper policies. At first, the user command is high performance and the workload scenario is high intensive. Thus, GDA prioritizes

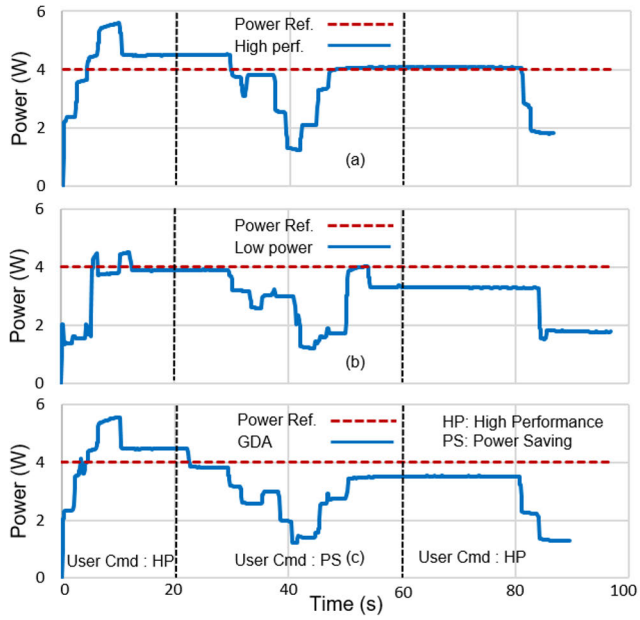


Figure 21. Comparison of the instantaneous power consumption of platform for three different approaches, (a) HP policy, (b) LP policy, and (c) GDA framework [8].

user's and workload's requirements and assigns higher weight to performance which causes power violation. After that, when user command switches to power saving, although the workload is still high intensive, GDA switches to low power for satisfying user and system's objectives. At last, the user command is high performance, but since the workload intensity is low, GDA selects a low power policy which can satisfy both user's and application's requirements.

Figure 22 illustrates the advantage of CGDA framework over GDA. The figure compares power consumption of system when using i) CGDA, ii) LP, and iii) HP frameworks over a dynamic workload scenario. The workload scenario is a combination of the applications in Table 5. For thermal safety, the TDP set to 5 W in this experiment. The highlighted parts in each plot show the efficiency of CGDA in comparison with the other approaches. From t = 0 to t = 20, workload intensity is low and the user requests low power mode. GDA, CGDA, and Low power (LP) frameworks consume low power and respect power constraint for low intensive applications. However, the high performance policy (HP) as shown in Figure 22 (c) consumes higher power compared to CGDA, which is not necessary. The HP violates the power consumption when the intensity of workload increases at t = 20 s. From t = 24 s to t = 40 s, four applications are running concurrently which create high intensive workload. The LP consumes low power, however it cannot meet requirements of all the applications. The GDA framework also falls short in satisfying the

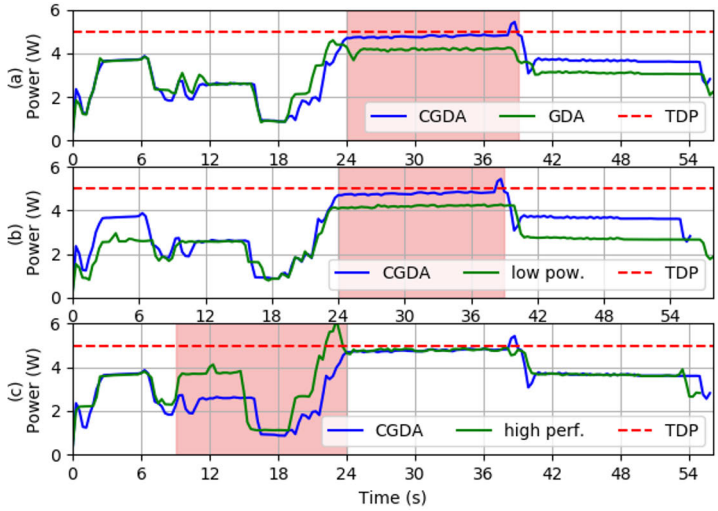


Figure 22. Comparison of instantaneous power consumption of CGDA framework against (a) GDA, (b) low power (LP), and (c) high performance (HP).

applications' requirements when the power consumption is near the threshold. GDA assigns higher priority for power, leading to the selection of low power policy. However, the CGDA framework enhances GDA by providing configurable goal hierarchy. The CGDA dynamically updates the hierarchy based on the extent of the available power budget, leading to a higher priority for performance from t = 24 s to t = 40 s. Therefore, CGDA satisfies all the applications while considering the power violation. In summary, the GDA and CGDA frameworks provide autonomy in resource management which leads to improvement in handling various entities' requirements at run-time. GDA and CGDA offer intelligent resource management approaches which prioritize the objectives dynamically, generate the goals based on the requirements, and then select suitable actions for achieving the goals. Such intelligent resource management provides an efficient trade-off between multiple conflicting objectives.

7 Conclusion

The need for intelligent and self-aware resource management in embedded systems is becoming more evident particularly by considering i) various types of applications with different intensity levels and different requirements, ii) core-level heterogeneity of the embedded systems and wide range of operating frequencies, iii) limitation of thermal, power, and energy budget, and iv) dynamic user requirements at run-time. In this dissertation, we introduced and discussed self-aware resource management approaches with various objectives by considering the aforementioned complexities and limitations. Resource management can have one or more objectives at run-time e.g., i) maximizing performance, ii) minimizing energy consumption, iii) increasing Battery Cycle Life (BCL), iv) increasing Quality of Experience (QoE), etc. Such objectives may be conflicting, orthogonal, or overlapping. In each chapter of this dissertation, we consider some of the objectives and discuss various challenges of resource management for pursuing those objectives. Then, we present solutions for addressing these challenges, commonly by using online and offline machine learning techniques. Chapters 3 and 4 discussed the challenges that arise by executing multiple concurrent applications with different characteristics and requirements. The objectives of resource management in these two chapters are i) minimizing energy consumption and ii) respecting performance requirements. Handling these two conflicting objectives for multiple concurrent applications that compete for resources became possible by using offline applications' characterization and designing prediction models for online management. Chapter 5 investigated the effects of user activities on battery aging and presented a user and battery- aware resource management. The resource management objectives in this chapter are i) maximizing QoE and ii) maximizing BCL. Chapter 5 presented a framework which uses machine learning techniques for predicting the users' activities and adapt the resource management for each user based on their activities and the predictions. The presented resource management approach increases BCL by decreasing the average State of Charge (SOC) in the battery. In chapters 3-5, the objectives are fixed at design time and the resource management approaches consider dynamic and unknown workload scenarios, user's requirements, and battery aging for achieving the pre-defined objectives. In chapter 6, we presented a general-purpose resource management framework for handling multiple objectives which are dynamic and variable at run-time. The GDA framework which is presented in chapter 6 is a decision making approach that prioritizes

the objectives at run-time and formulates goals for resource management. The goals are generated at any given time based on the requirements, the current state, and the predictions for the future state. Our experimental results in each chapter show the improvement of the resource allocation strategies compared to the state-of-the-art approaches. The evaluation results in chapter 6 show the advantages of GDA for managing the unknown and unpredictable environments for an embedded system compared to the fixed-objective resource management approaches.

List of References

[1] Anil Kanduri, Antonio Miele, Amir M Rahmani, Pasi Liljeberg, Cristiana Bolchini, and Nikil Dutt. Approximation-aware coordinated power/performance management for heterogeneous multi-cores. In *Proceedings of the 55th Annual Design Automation Conference*, pages 1–6, 2018.

[2] Ali Aalsaud, Rishad Shafik, Ashur Rafiev, Fie Xia, Sheng Yang, and Alex Yakovlev. Power–aware performance adaptation of concurrent applications in heterogeneous many-core systems. In *Proceedings of the 2016 International Symposium on Low Power Electronics and Design*, pages 368–373, 2016.

[3] Basireddy Karunakar Reddy, Amit Kumar Singh, Dwaipayan Biswas, Geoff V Merrett, and Bashir M Al-Hashimi. Inter-cluster thread-to-core mapping and dvfs on heterogeneous multi-cores. *IEEE Transactions on Multi-Scale Computing Systems*, 4(3):369–382, 2017.

[4] Ali Aalsaud, Ashur Rafiev, Fei Xia, Rishad Shafik, and Alex Yakovlev. Model-free runtime management of concurrent workloads for energy-efficient many-core heterogeneous systems. In *2018 28th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, pages 206–213. IEEE, 2018.

[5] Amir M Rahmani, Axel Jantsch, and Nikil Dutt. Hdgm: Hierarchical dynamic goal management for many-core resource allocation. *IEEE Embedded Systems letters*, 10(3):61–64, 2017.

[6] Anuj Pathania, Alexandru Eugen Irimiea, Alok Prakash, and Tulika Mitra. Power-performance modelling of mobile gaming workloads on heterogeneous mpsocs. In *Proceedings of the 52nd Annual Design Automation Conference*, pages 1–6, 2015.

[7] Axel Jantsch, Nikil Dutt, and Amir M Rahmani. Self-awareness in systems on chip—a survey. *IEEE Design & Test*, 34(6):8–26, 2017.

[8] Elham Shamsa, Anil Kanduri, Amir M Rahmani, Pasi Liljeberg, Axel Jantsch, and Nikil Dutt. Goal-driven autonomy for efficient on-chip resource management: Transforming objectives to goals. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1397–1402. IEEE, 2019.

[9] Thannirmalai Somu Muthukaruppan, Mihai Pricopi, Vanchinathan Venkataramani, Tulika Mitra, and Sanjay Vishin. Hierarchical power management for asymmetric multi-core in dark silicon era. In *2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–9. IEEE, 2013.

[10] Karunakar R Basireddy, Amit Kumar Singh, Bashir M Al-Hashimi, and Geoff V Merrett. Adamd: Adaptive mapping and dvfs for energy-efficient heterogeneous multicores. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(10):2206–2217, 2019.

[11] Nikita Mishra, Connor Imes, John D Lafferty, and Henry Hoffmann. Caloree: Learning control for predictable latency and low energy. *ACM SIGPLAN Notices*, 53(2):184–198, 2018.

[12] Hardkernel. ODROID-XU. URL `http://www.hardkernel.com/main/main.php`.

[13] Gereon Onnebrink, Florian Walbroel, Jonathan Klimt, Rainer Leupers, Gerd Ascheid, Luis Gabriel Murillo, Stefan Schürmans, Xiaotao Chen, and YwhPyng Harn. Dvfs-enabled power-performance trade-off in mpsoc sw application mapping. In *2017 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*, pages 196–202. IEEE, 2017.

[14] Yang Qin, Gang Zeng, Ryo Kurachi, Yixiao Li, Yutaka Matsubara, and Hiroaki Takada. Energy-efficient intra-task dvfs scheduling using linear programming formulation. *IEEE Access*, 7:30536–30547, 2019.

[15] Hao Zheng and Ahmed Louri. Ez-pass: An energy & performance-efficient power-gating router architecture for scalable nocs. *IEEE Computer Architecture Letters*, 17(1):88–91, 2017.

[16] Hossein Farrokhbakht, Hadi Mardani Kamali, and Natalie Enright Jerger. Muffin: Minimally-buffered zero-delay power-gating technique in on-chip routers. In *2019 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pages 1–6. IEEE, 2019.

[17] Davide Zoni, Andrea Canidio, William Fornaciari, Panayiotis Englezakis, Chrysostomos Nicopoulos, and Yiannakis Sazeides. Blackout: Enabling fine-grained power gating of buffers in network-on-chip routers. *Journal of Parallel and Distributed Computing*, 104:130–145, 2017.

[18] Di Zhu, Yunfan Li, and Lizhong Chen. On trade-off between static and dynamic power consumption in noc power gating. In *2019 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pages 1–6. IEEE, 2019.

[19] Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D Joseph, Randy H Katz, Scott Shenker, and Ion Stoica. Mesos: A platform for fine-grained resource sharing in the data center. In *NSDI*, volume 11, pages 22–22, 2011.

[20] Emanuele Del Sozzo, Gianluca C Durelli, EMG Trainiti, Antonio Miele, Marco D Santambrogio, and Cristiana Bolchini. Workload-aware power optimization strategy for asymmetric multiprocessors. In *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 531–534. IEEE, 2016.

[21] Ujjwal Gupta, Chetan Arvind Patil, Ganapati Bhat, Prabhat Mishra, and Umit Y Ogras. Dypo: Dynamic pareto-optimal configuration selection for heterogeneous mpsocs. *ACM Transactions on Embedded Computing Systems (TECS)*, 16(5s):1–20, 2017.

[22] Mohammed A Noaman Al-hayanni, Ashur Rafiev, Fei Xia, Rishad Shafik, Alexander Romanovsky, and Alex Yakovlev. Parma: parallelization-aware run-time management for energy-efficient many-core systems. *IEEE Transactions on Computers*, 69(10):1507–1518, 2020.

[23] Sumit K Mandal, Ganapati Bhat, Chetan Arvind Patil, Janardhan Rao Doppa, Partha Pratim Pande, and Umit Y Ogras. Dynamic resource management of heterogeneous mobile platforms via imitation learning. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 27 (12):2842–2854, 2019.

[24] Stavros Tzilis, Pedro Trancoso, and Ioannis Sourdis. Energy-efficient runtime management of heterogeneous multicores using online projection. *ACM Transactions on Architecture and Code Optimization (TACO)*, 15(4):1–26, 2019.

[25] Henry Hoffmann, Jonathan Eastep, Marco D Santambrogio, Jason E Miller, and Anant Agarwal. Application heartbeats: a generic interface for specifying program performance and goals in autonomous computing environments. In *Proceedings of the 7th international conference on Autonomic computing*, pages 79–88, 2010.

[26] Elham Shamsa, Anil Kanduri, Amir M Rahmani, Pasi Liljeberg, Axel Jantsch, and Nikil Dutt. Goal formulation: Abstracting dynamic objectives for efficient on-chip resource allocation. In *2018 IEEE Nordic Circuits and Systems Conference (NORCAS): NORCHIP and International Symposium of System-on-Chip (SoC)*, pages 1–4. IEEE, 2018.

[27] Samuel Kounev, Xiaoyun Zhu, Jeffrey O Kephart, and Marta Kwiatkowska. Model-driven algorithms and architectures for self-aware computing systems (dagstuhl seminar 15041). In *Dagstuhl Reports*, volume 5. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.

[28] Santanu Sarma and Nikil Dutt. Cross-layer exploration of heterogeneous multicore processor configurations. In *2015 28th International Conference on VLSI Design*, pages 147–152. IEEE, 2015.

[29] Henry Hoffmann. Coadapt: Predictable behavior for accuracy-aware applications running on power-aware systems. In *2014 26th Euromicro Conference on Real-Time Systems*, pages 223–232. IEEE, 2014.

[30] Henry Hoffmann, Jim Holt, George Kurian, Eric Lau, Martina Maggio, Jason E Miller, Sabrina M Neuman, Mahmut Sinangil, Yildiz Sinangil, Anant Agarwal, et al. Self-aware computing in the angstrom processor. In *Proceedings of the 49th Annual Design Automation Conference*, pages 259–264, 2012.

[31] Santanu Sarma, Nikil Dutt, Puneet Gupta, Nalini Venkatasubramanian, and Alexandru Nicolau. Cyberphysical-system-on-chip (cpsoc): A self-aware mpsoc paradigm with cross-layer virtual sensing and actuation. In *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 625–628. IEEE, 2015.

[32] Nikil Dutt, Amir M Rahmani, and Axel Jantsch. Empowering autonomy through self-awareness in mpsocs. In *2017 15th IEEE International New Circuits and Systems Conference (NEWCAS)*, pages 73–76. IEEE, 2017.

[33] Jurn-Gyu Park, Nikil Dutt, and Sung-Soo Lim. Ml-gov: A machine learning enhanced integrated cpu-gpu dvfs governor for mobile gaming. In *Proceedings of the 15th IEEE/ACM Symposium on Embedded Systems for Real-Time Multimedia*, pages 12–21, 2017.

[34] Ryan Cochran, Can Hankendi, Ayse K Coskun, and Sherief Reda. Pack & cap: adaptive dvfs and thread packing under power caps. In *2011 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 175–185. IEEE, 2011.

[35] Benedikt Dietrich, Swaroop Nunna, Dip Goswami, Samarjit Chakraborty, and Matthias Gries. Lms-based low-complexity game workload prediction for dvfs. In *2010 IEEE International Conference on Computer Design*, pages 417–424. IEEE, 2010.

[36] Ujjwal Gupta, Joseph Campbell, Umit Y Ogras, Raid Ayoub, Michael Kishinevsky, Francesco Paterna, and Suat Gumussoy. Adaptive performance prediction for integrated gpus. In *Proceedings of the 35th International Conference on Computer-Aided Design*, pages 1–8, 2016.

[37] Vinay Hanumaiah, Digant Desai, Benjamin Gaudette, Carole-Jean Wu, and Sarma Vrudhula. Steam: A smart temperature and energy aware multicore controller. *ACM Transactions on Embedded Computing Systems (TECS)*, 13(5s):1–25, 2014.

[38] Kai Ma, Xiaorui Wang, and Yefu Wang. Dppc: dynamic power partitioning and control for improved chip multiprocessor performance. *IEEE Transactions on Computers*, 63(7):1736–1750, 2013.

[39] Zhuo Chen and Diana Marculescu. Distributed reinforcement learning for power limited many-core system performance optimization. In *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1521–1526. IEEE, 2015.

[40] Fakhruddin Muhammad Mahbub ul Islam and Man Lin. Hybrid dvfs scheduling for real-time systems based on reinforcement learning. *IEEE Systems Journal*, 11(2):931–940, 2015.

[41] Quintin Fettes, Mark Clark, Razvan Bunescu, Avinash Karanth, and Ahmed Louri. Dynamic voltage and frequency scaling in nocs with supervised and reinforcement learning techniques. *IEEE Transactions on Computers*, 68(3):375–389, 2018.

[42] Ujjwal Gupta, Sumit K Mandal, Manqing Mao, Chaitali Chakrabarti, and Umit Y Ogras. A deep q-learning approach for dynamic management of heterogeneous processors. *IEEE Computer Architecture Letters*, 18(1):14–17, 2019.

[43] Rishad A Shafik, Sheng Yang, Anup Das, Luis A Maeda-Nunez, Geoff V Merrett, and Bashir M Al-Hashimi. Learning transfer-based adaptive energy minimization in embedded systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 35(6):877–890, 2015.

[44] Hao Shen, Ying Tan, Jun Lu, Qing Wu, and Qinru Qiu. Achieving autonomous power management using reinforcement learning. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 18(2):1–32, 2013.

[45] Rong Ye and Qiang Xu. Learning-based power management for multicore processors via idle period manipulation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 33(7):1043–1055, 2014.

[46] Qingchen Zhang, Man Lin, Laurence T Yang, Zhikui Chen, Samee U Khan, and Peng Li. A double deep q-learning model for energy-efficient edge scheduling. *IEEE Transactions on Services Computing*, 12(5):739–749, 2018.

[47] Ryan Gary Kim, Wonje Choi, Zhuo Chen, Janardhan Rao Doppa, Partha Pratim Pande, Diana Marculescu, and Radu Marculescu. Imitation learning for dynamic vfi control in large-scale

manycore systems. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 25(9): 2458–2471, 2017.

[48] Basireddy Karunakar Reddy, Geoff V Merrett, Bashir M Al-Hashimi, and Amit Kumar Singh. Online concurrent workload classification for multi-core energy management. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 621–624. IEEE, 2018.

[49] Elham Shamsa, Anil Kanduri, Pasi Liljeberg, and Amir M Rahmani. Concurrent application bias scheduling for energy efciency of heterogeneous multi-core platforms. *IEEE Transactions on Computers*, 2021.

[50] Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W Sheaffer, Sang-Ha Lee, and Kevin Skadron. Rodinia: A benchmark suite for heterogeneous computing. In *2009 IEEE international symposium on workload characterization (IISWC)*, pages 44–54. Ieee, 2009.

[51] Yi Ding, Nikita Mishra, and Henry Hoffmann. Generative and multi-phase learning for computer systems optimization. In *Proceedings of the 46th International Symposium on Computer Architecture*, pages 39–52, 2019.

[52] AM Coutinho Demetrios, Daniele De Sensi, Arthur Francisco Lorenzon, Kyriakos Georgiou, Jose Nunez-Yanez, Kerstin Eder, and Samuel Xavier-de Souza. Performance and energy trade-offs for parallel applications on heterogeneous multi-processing systems. *Energies*, 13(9):2409, 2020.

[53] Hergys Rexha, Simon Holmbacka, and Sébastien Lafond. Core level utilization for achieving energy efficiency in heterogeneous systems. In *2017 25th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*, pages 401–407. IEEE, 2017.

[54] Hom-Lay Wang and Khalaf Al-Shammari. Hvc ridge deficiency classification: a therapeutically oriented classification. *International Journal of Periodontics & Restorative Dentistry*, 22 (4), 2002.

[55] Venkatesh Pallipadi and Alexey Starikovskiy. The ondemand governor. In *Proceedings of the Linux Symposium*, volume 2, pages 215–230, 2006.

[56] Vincenzo Scoca, Atakan Aral, Ivona Brandic, Rocco De Nicola, and Rafael Brundo Uriarte. Scheduling latency-sensitive applications in edge computing. In *Closer*, pages 158–168, 2018.

[57] Jason Cong, Muhuan Huang, Bin Liu, Peng Zhang, and Yi Zou. Combining module selection and replication for throughput-driven streaming programs. In *2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1018–1023. IEEE, 2012.

[58] Elham Shamsa, Anil Kanduri, Amir M Rahmani, and Pasi Liljeberg. Energy-performance co-management of mixed-sensitivity workloads on heterogeneous multi-core systems. In *Proceedings of the 26th Asia and South Pacific Design Automation Conference*, pages 421–427, 2021.

[59] C. Bienia et al. The PARSEC benchmark suite: Characterization and architectural implications. In *Proc. of PACT*, 2008.

[60] Kaige Yan, Xingyao Zhang, Jingweijia Tan, and Xin Fu. Redefining qos and customizing the power management policy to satisfy individual mobile users. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1–12. IEEE, 2016.

[61] Liang He, Eugene Kim, Kang G Shin, Guozhu Meng, and Tian He. Battery state-of-health estimation for mobile devices. In *Proceedings of the 8th International Conference on Cyber-Physical Systems*, pages 51–60, 2017.

[62] Matthew B Pinson and Martin Z Bazant. Theory of sei formation in rechargeable batteries: capacity fade, accelerated aging and lifetime prediction. *Journal of the Electrochemical Society*, 160 (2):A243, 2012.

[63] Yukai Chen, Alberto Bocca, Alberto Macii, Enrico Macii, and Massimo Poncino. A li-ion battery charge protocol with optimal aging-quality of service trade-off. In *Proceedings of the 2016 International Symposium on Low Power Electronics and Design*, pages 40–45, 2016.

[64] Alan Millner. Modeling lithium ion battery degradation in electric vehicles. In *2010 IEEE Conference on Innovative Technologies for an Efficient and Reliable Electricity Supply*, pages 349–356. IEEE, 2010.

[65] Gang Ning and Branko N Popov. Cycle life modeling of lithium-ion batteries. *Journal of The Electrochemical Society*, 151(10):A1584, 2004.

[66] Alma Pröbstl, Philipp Kindt, Emanuel Regnath, and Samarjit Chakraborty. Smart2: Smart charging for smart phones. In *2015 IEEE 21st International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 41–50. IEEE, 2015.

[67] Alberto Bocca, Alessandro Sassone, Alberto Macii, Enrico Macii, and Massimo Poncino. An aging-aware battery charge scheme for mobile devices exploiting plug-in time patterns. In *2015 33rd IEEE International Conference on Computer Design (ICCD)*, pages 407–410. IEEE, 2015.

[68] Elham Shamsa, Alma Pröbstl, Nima TaheriNejad, Anil Kanduri, Samarjit Chakraborty, Amir M Rahmani, and Pasi Liljeberg. Ubar: User-and battery-aware resource management for smartphones. *ACM Transactions on Embedded Computing Systems (TECS)*, 20(3):1–25, 2021.

[69] Donghwa Shin, Kitae Kim, Naehyuck Chang, Woojoo Lee, Yanzhi Wang, Qing Xie, and Massoud Pedram. Online estimation of the remaining energy capacity in mobile systems considering system-wide power consumption and battery characteristics. In *2013 18th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 59–64. IEEE, 2013.

[70] Qing Xie, Jaemin Kim, Yanzhi Wang, Donghwa Shin, Naehyuck Chang, and Massoud Pedram. Dynamic thermal management in mobile devices considering the thermal coupling between battery and application processor. In *2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 242–247. IEEE, 2013.

[71] Aaron Carroll and Gernot Heiser. An analysis of power consumption in a smartphone. In *USENIX annual technical conference*, volume 14, pages 21–21. Boston, MA, 2010.

[72] Tae-Rok Hwang. Battery log, version 2.0.3. `https://play.google.com`, 2013.

[73] Eibe Frank, Mark Hall, and Bernhard Pfahringer. Locally weighted naive bayes. *arXiv preprint arXiv:1212.2487*, 2012.

[74] Elham Shamsa, Anil Kanduri, Nima TaheriNejad, Alma Pröbstl, Samarjit Chakraborty, Amir M Rahmani, and Pasi Liljeberg. User-centric resource management for embedded multi-core processors. In *2020 33rd International Conference on VLSI Design and 2020 19th International Conference on Embedded Systems (VLSID)*, pages 43–48. IEEE, 2020.

[75] Matthew R Guthaus, Jeffrey S Ringenberg, Dan Ernst, Todd M Austin, Trevor Mudge, and Richard B Brown. Mibench: A free, commercially representative embedded benchmark suite. In *Proceedings of the fourth annual IEEE international workshop on workload characterization. WWC-4 (Cat. No. 01EX538)*, pages 3–14. IEEE, 2001.

[76] Wooseok Lee, Reena Panda, Dam Sunwoo, Jose Joao, Andreas Gerstlauer, and Lizy K John. Buqs: battery-and user-aware qos scaling for interactive mobile devices. In *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 64–69. IEEE, 2018.

[77] Anil Kanduri, Mohammad-Hashem Haghbayan, Amir M Rahmani, Pasi Liljeberg, Axel Jantsch, Nikil Dutt, and Hannu Tenhunen. Approximation knob: Power capping meets energy efficiency. In *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8. IEEE, 2016.

[78] Matthew Klenk, Matt Molineaux, and David W Aha. Goal-driven autonomy for responding to unexpected events in strategy simulations. *Computational Intelligence*, 29(2):187–206, 2013.

[79] Ulit Jaidee, Héctor Munoz-Avila, and David W Aha. Integrated learning for goal-driven autonomy. In *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

[80] Ben Weber, Michael Mateas, and Arnav Jhala. Learning from demonstration for goal-driven autonomy. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 26, 2012.

[81] Dana S Nau. Current trends in automated planning. *AI magazine*, 28(4):43–43, 2007.

[82] Jay Powell, Matthew Molineaux, and David W Aha. Active and interactive learning of goal selection knowledge. In *Proceedings of the Twenty-Fourth Florida Artificial Intelligence Research Society Conference*, 2011.

[83] Dongkyu Choi. Reactive goal management in a cognitive architecture. *Cognitive Systems Research*, 12(3-4):293–308, 2011.

[84] Stuart Russell and Peter Norvig. Artificial intelligence: a modern approach. 2002.

TURUN
YLIOPISTO
UNIVERSITY
OF TURKU

Painosalama, Turku, Finland 2022

ISBN 978-951-29-8907-2 (PRINT)
ISBN 978-951-29-8908-9 (PDF)
ISSN 2736-9390 (PRINT)
ISSN 2736-9684 (ONLINE)