DISSERTATION

A SEMI-DYNAMIC RESOURCE MANAGEMENT FRAMEWORK FOR MULTICORE EMBEDDED SYSTEMS WITH ENERGY HARVESTING

Submitted by

Yi Xiang

Department of Electrical and Computing Engineering

In partial fulfillment of the requirements

For the Degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Fall 2015

Doctoral Committee:

Advisor: Sudeep Pasricha

Anura Jayasumana
H. J. Siegel
Michelle Mills Strout

ABSTRACT

A SEMI-DYNAMIC RESOURCE MANAGEMENT FRAMEWORK FOR MULTICORE EMBEDDED SYSTEMS WITH ENERGY HARVESTING

Semiconductor technology has been evolving rapidly over the past several decades, introducing a new breed of embedded systems that are tiny, efficient, and pervasive. These embedded systems are the backbone of the ubiquitous and pervasive computing revolution, embedded intelligence all around us. Often, such embedded intelligence for pervasive computing must be deployed at remote locations, for purposes of environment sensing, data processing, information transmission, etc. Compared to current mobile devices, which are mostly supported by rechargeable and exchangeable batteries, emerging embedded systems for pervasive computing favor a self-sustainable energy supply, as their remote and mass deployment makes it impractical to change or charge their batteries. The ability to sustain systems by scavenging energy from ambient sources is called energy harvesting, which is gaining monument for its potential to enable energy autonomy in the era of pervasive computing. Among various energy harvesting techniques, solar energy harvesting has attracted the most attention due to its high power density and availability.

Another impact of semiconductor technology scaling into the deep submicron level is the shifting of design focus from performance to energy efficiency as power dissipation on a chip cannot increase indefinitely. Due to unacceptable power consumption at high clock rate, it is desirable for computing systems to distribute workload on multiple cores with reduced execution frequencies so that overall system energy efficiency improves while meeting performance goals.

Thus it is necessary to adopt the design paradigm of multiprocessing for low-power embedded systems due to the ever-increasing demands for application performance and stringent limitations on power dissipation.

In this dissertation we focus on the problem of resource management for multicore embedded systems powered by solar energy harvesting. We have conducted a substantial amount of research on this topic, which has led to the design of a semi-dynamic resource management framework designed with emphasis on efficiency and flexibility that can be applied to energy harvesting-powered systems with a variety of functionality, performance, energy, and reliability goals. The capability and flexibility of the proposed semi-dynamic framework are verified by issues we have addressed with it, including: (i) minimizing miss rate/miss penalty of systems with energy harvesting, (ii) run-time thermal control, (iii) coping with process variation induced core-to-core heterogeneity, (iv) management of hybrid energy storage, (v) scheduling of task graphs with inter-node dependencies, (vi) addressing soft errors during execution, (vii) mitigating aging effects across the chip over time, and (vii) supporting mixed-criticality scheduling on heterogeneous processors.

ACKNOWLEDGEMENTS

I would like to thank all the individuals whose encouragement and support have made the completion of this dissertation possible.

First and foremost, I would like to express my sincere gratitude to my advisor, Dr. Sudeep Pasricha, who has patiently guided me through the entire process of graduate study step by step. In the last year of my bachelor program in semiconductor physics, I made up my mind to seek overseas study opportunities in another area to feed my curiosity about the interactions between computer hardware and software. Although the picture of snowcapped mountains on the ECE department website was impressive, it was Dr. Pasricha's description of research on multicore embedded systems that immediately caught my eye and enlightened me about the field. Since then I have never looked back as I was fortunate enough to join his research group and to receive his help, which changed my life. In the first year, the course and research work suggested by Dr. Pasricha helped prepare me with basic skills for research and reassured me that I had found my area of interests. After that, it was his vision and wisdom that stimulated me to look at research problems with more critical and creative thinking, which led to several publications in well-known conferences and journals. For countless times, I was impressed by his thoroughness and attention to detail despite his tight schedule, from which I got to know his passion and enthusiasm for research. On the other hand, he is the type of advisor that is caring enough to suggest his graduate students to slow down, get some rest, and recharge whenever he senses high pressure on them. Dr. Pasricha can also give good life advice when required, which helped me to overcome various difficulties and confusions in life and study during my graduate school years. I really appreciate

all the help, guidance, and inspiration I received from Dr. Pasricha, who made it possible for me to survive the trials of graduate school with unforgettable memories and broadened horizons.

I would like to take this opportunity to thank the respected members of my PhD committee, Dr. H. J. Siegel, Dr. Anura Jayasumana, and Dr. Michelle Mills Strout. Their feedback helped me to rediscover my research and refine my work from different perspectives. I am also thankful to my colleagues in Dr. Pasricha's MECS lab for their collaboration during my Ph.D. study: Yong Zou, Nishit Kapadia, and Brad Donohoo. Also this list cannot be complete without mentioning company and help from Srinivas Desai, Vipin Kumar Kukkala, Ishan Thakkar, Saideep Tiku, C Sai Vineel Reddy, Shirish Bahirat, Yuhang Li, Tejasi Pimpalkhute, Pramit Rajkrishna, Dalton Young, Daniel Dauwe, and Shoumik Maiti.

I would like to thank my family, especially my parents, for their support to pursue my Ph.D. on the other side of the planet. I cannot wait to share more good news with them in the future as I continue with my work and study. Their kindness shaped my view of this world and made me the person I am.

Thank you to Yixiao, for all her love and support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

LIST OF ACRONYMS

APU → **a**ccelerated **p**rocessing **u**nit

CMOS → **c**omplementary **m**etal-**o**xide **s**emiconductor

DTS → **d**igital **t**hermal **s**ensor

DoP → **d**egree **o**f **p**arallelism

DVFS → **d**ynamic **v**oltage and **f**requency **s**caling

EDF → **e**arliest **d**eadline **f**irst scheduling algorithm

EM → **e**lectro**m**igration

ILP → **i**nstruction **l**evel **p**arallelism

ISA → **i**nstruction **s**et **a**rchitecture

MILP → **m**ixed **i**nteger **l**inear **p**rogramming

MPPT → **m**aximum **p**ower **p**oint **t**racking

MTTF → **m**ean-**t**ime-**t**o-**f**ailure

NBTI → **n**egative **b**ias **t**emperature **i**nstability

NOC → **n**etwork-**o**n-**c**hip

NTC → **n**ear-**t**hreshold **c**omputing

RTOS → **r**eal-**t**ime **o**perating **s**ystem

SA → **s**imulated **a**nnealing

SMP → **s**ymmetric **m**ulti**p**rocessor

STC → **s**uper-**t**hreshold **c**omputing

TDDB → **t**ime **d**ependent **d**ielectric **b**reakdown

TDP → **t**hermal **d**esign **p**ower

LIST OF PUBLICATIONS

- Y. Xiang, S. Pasricha, "Mixed-Criticality Scheduling on Heterogeneous Multicore Systems Powered by Energy Harvesting", ACM Transaction on Embedded Computing (TECS), ***under review***.

- Y. Xiang, S. Pasricha, "Soft and Hard Reliability-Aware Scheduling for Multicore Embedded Systems with Energy Harvesting", IEEE Transactions on Multi-Scale Computing Systems (TMSCS), ***under review***.

- Y. Xiang, S. Pasricha, "Run-Time Management for Multi-Core Embedded Systems with Energy Harvesting", IEEE Transactions on Very Large Scale Integration Systems (TVLSI), March 2015.

- Y. Xiang, S. Pasricha, "Fault-Aware Application Scheduling in Low Power Embedded Systems with Energy Harvesting", ACM/IEEE International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), October 2014.

- Y. Xiang, S. Pasricha, "A Hybrid Framework for Application Allocation and Scheduling in Multicore Systems with Energy Harvesting", ACM Great Lakes Symposium on VLSI (GLSVLSI), May 2014.

- B. Donohoo, C. Ohlsen, S. Pasricha, C. Anderson, Y. Xiang, "Context-Aware Energy Enhancements for Smart Mobile Devices", IEEE Transactions on Mobile Computing, July 2013.

- Y. Xiang, S. Pasricha, "Harvesting-Aware Energy Management for Multicore Platforms with Hybrid Energy Storage", ACM Great Lakes Symposium on VLSI (GLSVLSI), May 2013.

- Y. Xiang, S. Pasricha, "Thermal-Aware Semi-Dynamic Power Management for Multicore Systems with Energy Harvesting", IEEE International Symposium on Quality Electronic Design (ISQED), March 2013.

- Y. Zou, Y. Xiang, S. Pasricha, "Characterizing Vulnerability of Network Interfaces in Embedded Chip Multiprocessors", IEEE Embedded System Letters, June 2012.

- Y. Zou, Y. Xiang, S. Pasricha, "Analysis of On-chip Interconnection Network Interface Reliability in Multicore Systems", IEEE International Conference on Computer Design (ICCD), October 2011.

1. INTRODUCTION

Energy constraints remain the major factor that limits the availability and versatility of embedded systems in the era of pervasive computing [1]. Despite tremendous efforts in academia and industry to improve the energy efficiency of current embedded devices, there is still need for an effective solution that can be applied to energy-constrained embedded systems deployed in remote locations around the world [2]. This chapter contains an introduction to the basic concepts of energy harvesting, which has emerged recently as an attractive alternative to supply energy for embedded systems when other energy sources are limited or unavailable. Also, we introduce real-time multicore embedded systems as the target platform type in this dissertation. Lastly, this chapter discusses the need for an intelligent resource management framework to exploit the full potential of multicore embedded systems powered by energy harvesting.

1.1. ENERGY HARVESTING

Energy harvesting, also known as power harvesting or energy scavenging, is the process of deriving energy from external sources, such as wind energy, thermal energy, kinetic energy, and solar energy [3]. With its history tracing back to the invention of windmills and waterwheels, in recent years energy harvesting has attracted ever-increasing interest and investments from the industrial sector, the research community, and individual prospectors due to its positive effects on both the environment and the economy, two of the major concerns for modern society. The energy harvesting technologies market was worth $131.4 million in 2012 and is projected to increase to $4.2 billion in 2019 [4]. Figure 1 shows change in web search frequency for the term “energy

harvesting" over time, which is broadly in line with the rising interest in this topic. In this dissertation, we focus on energy harvesting technologies used for electronic devices [5].

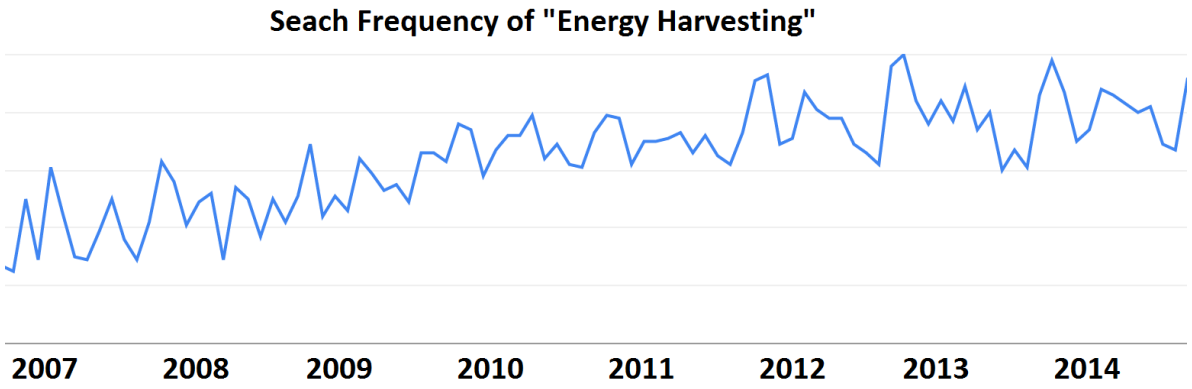

Figure 1 Normalized Search Frequency of "Energy Harvesting" over Time [6]

Although energy exists everywhere in the physical universe in multiple forms, only some forms can be effectively converted into electric energy to power electronic systems, including piezoelectric energy, thermal energy, wind energy, and solar energy. Listed below are some of the most common forms of energy available for energy harvesting:

- *Piezoelectric energy*: Piezoelectric effect is the phenomenon of accumulating electric charge in certain solid materials when mechanical stress is applied. This effect can be utilized to convert subtle energy sources, such as seismic vibration, acoustic noise, and ambient object motion, into electric energy, which becomes available in the form of a voltage difference between material surfaces [7]. As piezoelectric energy harvesting techniques usually generate electric power in the order of a microwatt, they are normally employed only in micro-scale electronic devices. A common example of piezoelectric energy harvesting is in step detection sensors deployed in sports shoes.

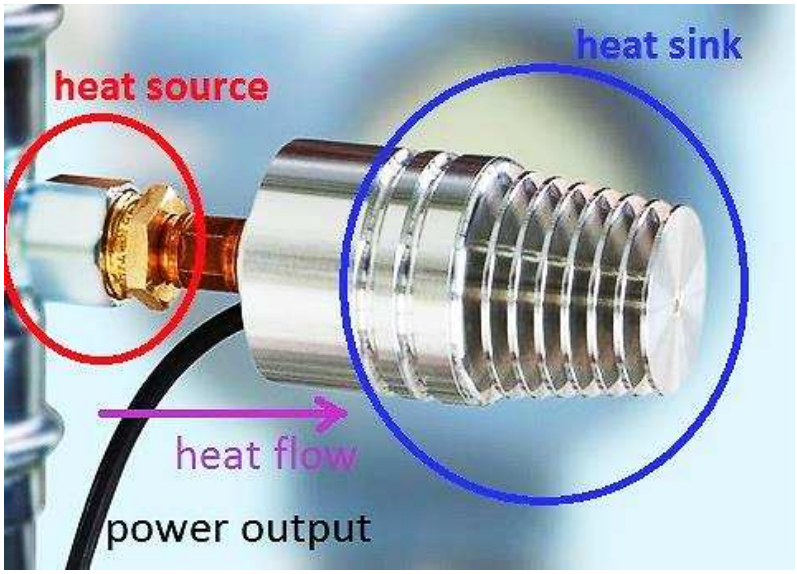

Figure 2 TE-Power PROBE Thermal Harvester by Micropelt [8]

- *Thermal energy*: Heat flow due to thermal gradient in a conducting material can also produce a voltage difference and thus provides the possibility of converting thermal energy into electric energy. [9] The major drawback of this form of energy harvesting is the fact that long-term stable thermal gradients are only available in particular places, limiting the location flexibility in deployment. Therefore, systems powered by thermal gradients are usually seen attached to the surface of other heating objects as parasitic devices. For example, Figure 2 shows the TE-Power PROBE thermal harvester manufactured by Micropelt, which can be attached to hot surfaces, such as a pipe with warm water flowing through it, to enable thermal harvesting by dissipating heat through its heat sink into ambient air [8].
- *Wind energy*: Wind energy has been demonstrated to be both technically and economically viable [10]. The most common exploitation of wind energy is with the help of large-scale wind turbines deployed at geographically windy locations around the world, which provide

an auxiliary clean energy source to power grids for utility providers. On the other hand, small-scale wind turbines also exist for specific applications such as auxiliary power supply for boats. The major issue facing the harvesting of wind energy lies in its strict location requirement and unstable wind conditions over time.

Figure 3 Photovoltaic Panels at Various Scales

- *Solar energy*: Solar energy [11], which is widely considered as a possible replacement for the more costly fossil energy in the future, is probably the most discussed source of renewable energy in recent years. Solar energy is derived from sunlight which is the most plentiful and widely distributed renewable energy source on earth. The most common method of harvesting solar energy is to convert solar radiation into electricity using photovoltaic panels (solar panels) [12]. As a result of technological advancements, there have been significant reductions in manufacturing cost and improvements in conversion efficiency of photovoltaic panels. In addition, photovoltaic panels are available at various scales, making it practical for applications in different areas ranging from industrial utility energy production to consumer level electronics, as examples shown in Figure 3. *Therefore*,

solar energy is widely recognized as the most promising source of energy harvesting for electronic systems.

In this dissertation, we consider solar energy as the source of energy harvesting to power real-time embedded multicore systems for best-effort execution due to its advantages in power density, availability, and scalability.

1.2. REAL-TIME MULTICORE EMBEDDED SYSTEMS

This section introduces and motivates the use of the primary category of platforms considered in this dissertation: real-time multicore embedded systems powered by solar energy harvesting.

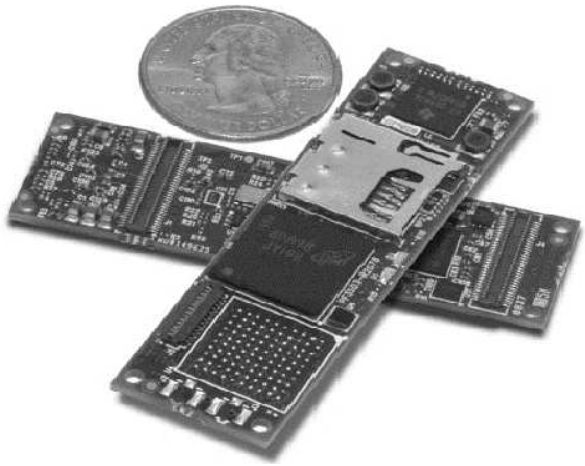

Figure 4 Example of Tiny Embedded Computer with Wi-Fi and Bluetooth

1.2.1. EMBEDDED SYSTEMS

An embedded system is a computer system with dedicated functions that is integrated within a larger mechanical or electrical system, often with real-time computing constraints [13].

Embedded systems are involved in a large portion of our daily life and have been ubiquitously deployed all over the world. We can find their existence for numerous applications, from space stations to microwave ovens, usually in small but powerful forms. Figure 4 shows an example of an embedded computing system with Wi-Fi and Bluetooth support. In the upcoming era of pervasive computing, embedded systems can play an even more important role with the help of energy harvesting technologies to achieve energy autonomy.

1.2.2. REAL-TIME SYSTEMS AND WORKLOAD MODELS

Computing systems with timing behavior as part of their performance or correctness criterion are called real-time computing systems [14]. While logical correctness is necessary for all types of computing systems, real-time systems are also subject to certain timing constraints usually characterized as *deadlines* to finish real-time jobs. These deadlines for the system and workload can further be classified into hard, soft, and firm deadlines:

- *Hard deadline*: Missing of a hard deadline is considered total system failure that in practice may lead to undesirable or even catastrophic consequences [15]. Therefore, hard real-time systems should have zero tolerance to a hard deadline miss. Such a strong guarantee in timing is only necessary for real-time systems where delayed response would actually cause great loss in profit, damage in physical surroundings, or even harm to human beings. For example, aircraft engine control systems must be designed to deal with hard deadlines in a robust manner as any delayed action may result in a dangerous flight state. Since hard-deadlines define very strict timing constraints, hard real-time embedded systems usually require abundant on-board resource to guarantee high robustness.

- *Soft deadline*: Unlike hard deadlines, soft deadlines can be missed without any immediate impact on system performance and functionality. Soft real-time systems are typically designed for non-critical, less timing-sensitive applications [16]. An example can be a data sensing hub trying to update a remote server with data samples stored in its on-board buffer queue, for which a single transmission task can miss its deadline without significant impact to the system's effectiveness, as another transmission can be scheduled in the next interval. However, the system may still face a performance impact if there are too many soft deadline misses in a short period of time because its on-board buffer queue will then fill up. In such a case, the sensing hub will drop less important data points to make room for new ones so that the system continues execution without total failure. Compared to hard deadlines, soft deadline constraints provide more flexibility in system design, enabling more effective trade-offs between the deadline miss rate and other criteria such as energy efficiency.
- *Firm deadline*: A deadline is firm if missing it results in immediate system performance degradation [17]. As with soft deadlines, missing a firm deadline does not lead to total system failure. However, missing any firm deadlines leads to immediate performance penalty and the task with the missed firm deadline is dropped as delayed output is considered invalid. A good example of firm real-time systems is a security camera system that always tries to provide the latest captured frames to its client. When a system fails to deliver a frame by its deadline, this missed frame should be dropped immediately in order to avoid accumulation of delay for the upcoming frames.

As solar energy harvesting is unable to guarantee a stable and continuous energy supply, hard real-time systems are not suitable candidates to work with energy harvesting, thus these

systems are beyond the scope of this dissertation. Our contribution focuses on embedded systems with firm deadlines, the miss rate of which is the main criteria to improve given a limited energy supply. Our work also considers soft deadlines in certain parts of this dissertation to form a more flexible workload model.

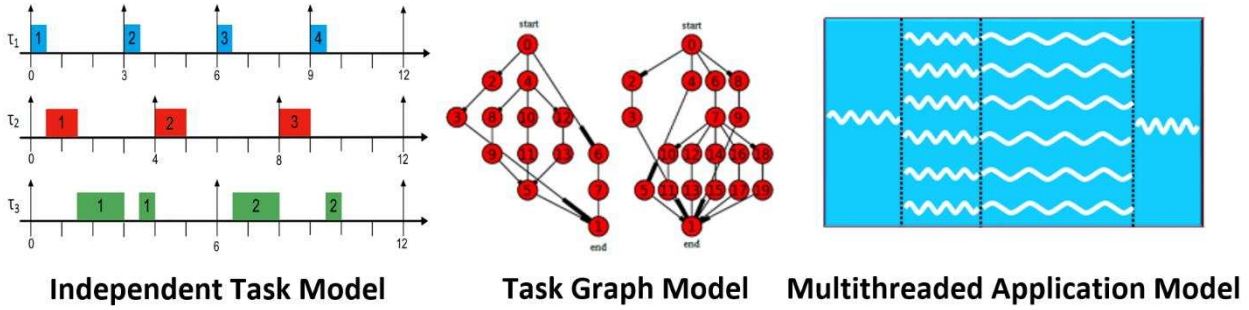

Figure 5 Workload Models Considered in this Dissertation

To model real-time applications with varying structures and requirements, this dissertation considers three types of workload models (see Figure 5): independent tasks, task graphs with dependencies, and multithread applications, which are described in more detail in Section 2.2, 3.3, and 4.3, respectively. Our work also mainly focuses on optimizations for periodic arrivals [18] of these different types of workloads.

1.2.3. MULTICORE PROCESSORS IN EMBEDDED SYSTEMS

Multicore processors are computing units with more than one processing core manufactured on a single chip, which are used across many application domains including supercomputing, mobile computing, and embedded processing. Although the concept of multiprocessing has for long been implemented using multiple discrete CPUs for supercomputers and servers, multicore processors have not been commonly used until recent years, when it became clear that it was no

longer viable to improve performance of processors by merely increasing their operating frequency or architectural complexity. The reasons for the paradigm shift towards multicore computing can be characterized by the "three walls" of computing [19]:

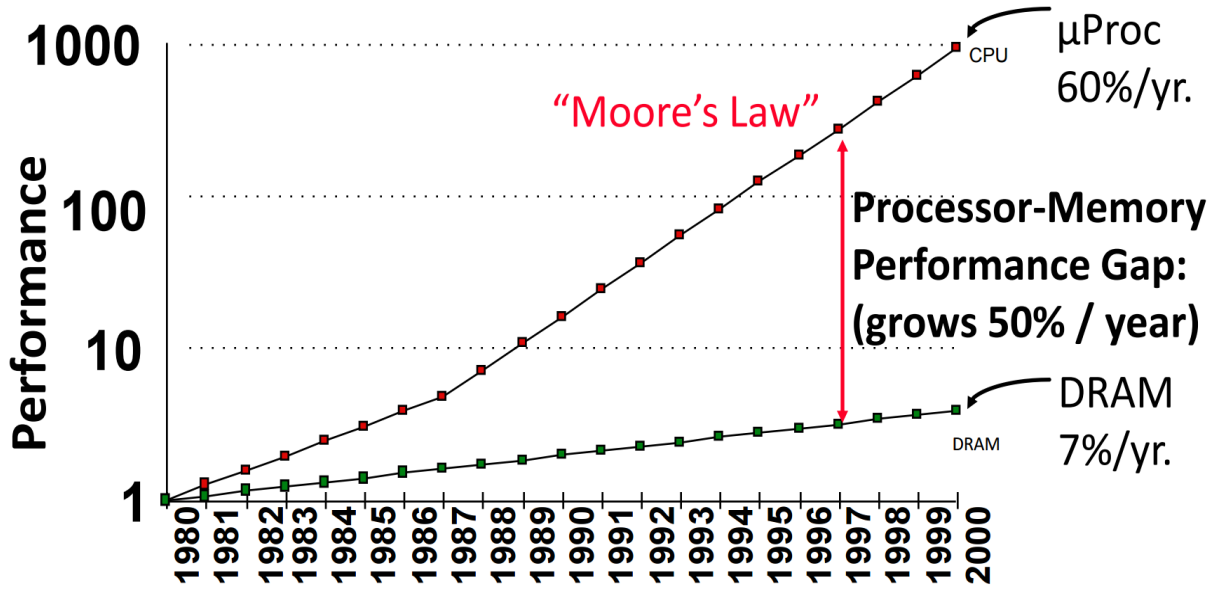

Figure 6 Increasing Processor-Memory Performance Gap [19]

- *Memory wall*: Due to the increasing performance gap between processors and memory, as shown in Figure 6, memory access delay has become a main obstacle hindering computing performance improvement. Thus merely increasing clock speed of emerging processors does not yield performance gain anymore because the processors spend significant amount of time waiting for data to arrive from memory. Even worse, higher frequency usually means much higher power consumption and reduced energy efficiency.
- *ILP wall*: It is hard to find enough ***i****nstruction* ***l****evel* ***p****arallelism* (**ILP**) in a single application to maintain high utilization of components on a high-performance single-core processor. Besides, attempts to extract high ILP from processors often results in low energy efficiency.

For example, an out-of-order processor design compared in [20] against an in-order design resulted in 2.4x performance improvement at the cost of 4.3x more power consumption, indicating a substantial energy efficiency reduction of 45%.

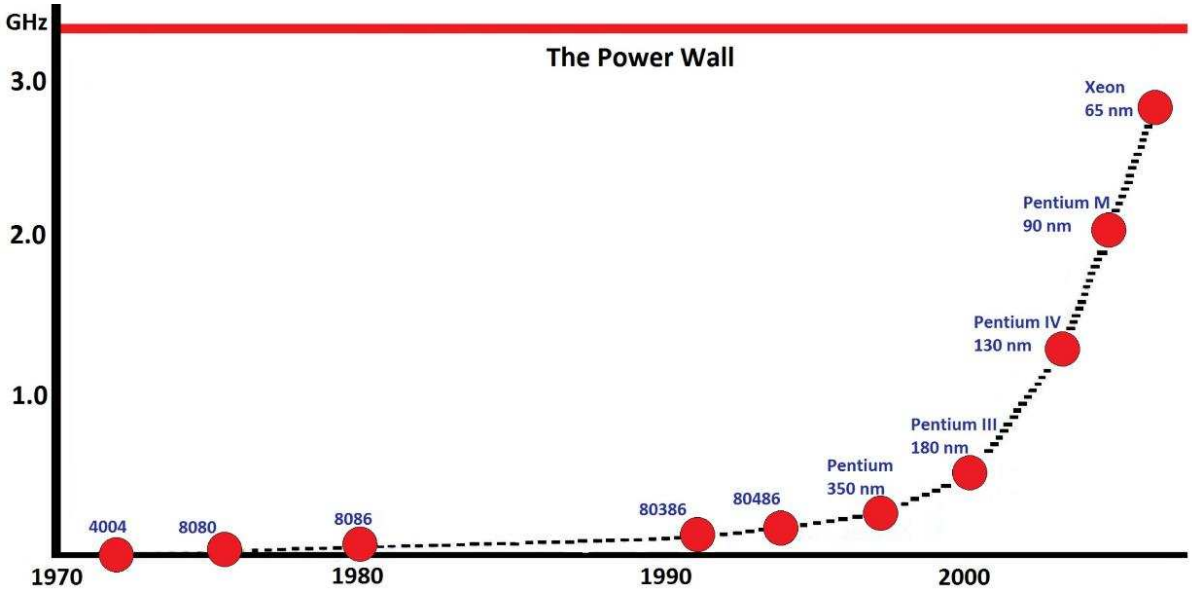

Figure 7 Approaching Power Wall with Dennard Scaling [21]

- *Power wall*: It is not possible to increase processor power dissipation indefinitely no matter how much performance we may gain. Figure 7 shows that the rising clock speed of processors had already approached the power wall in the early 2000s, which represents the limit on ***t**hermal **d**esign **p**ower* (**TDP**) for a single chip due to problems in technology scaling and thermal dissipation. Thus, the processing capability we can extract from a processor does not depend on its peak performance anymore. Maintaining execution at maximum clock speed leads to core overheating because of the exponential increase in power dissipation with factorial increase in operating frequency. Instead, the design focus

for today's processors has shifted to energy efficiency because performance per watt decides how much processing power can be utilized for a given power budget.

Due to stringent power/energy constraints, energy efficiency is even more crucial for embedded systems in terms of performance as well as service availability (% time that a system can be functioning). For this reason, recent years have led to increasing popularity of multicore processors in high-end embedded systems, especially for mobile devices [22] [23] [24].

The introduction of multicore processing has also ushered in a variety of processor architecture compositions. Based on the types of cores integrated, multicore processors can be classified into three categories:

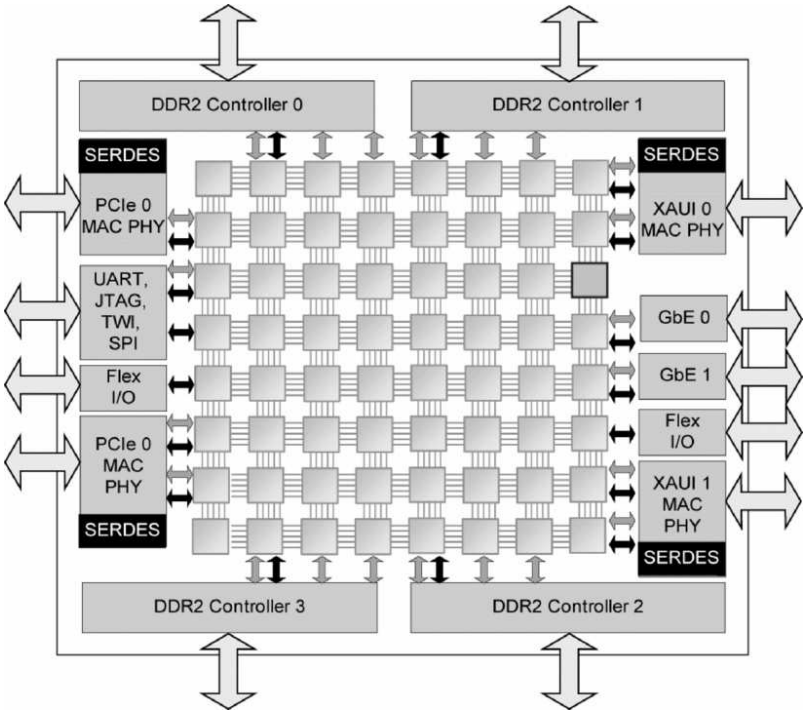

Figure 8 Diagram of Tile64 processor by Tilera [25]

- *Homogeneous*: By reusing the same design for all cores across the chip, homogeneous multicore processors provide a symmetric architecture that simplifies the programming model and on-board resource management. For example, Figure 8 shows the Tile64 processor designed by Tilera, a homogenous many core chip with 64 identical processors arranged in an 8x8 array and connected through a 2D mesh network [25]. However, this approach overlooks the opportunity to provide diversity in hardware to better support diverse execution patterns of different types of applications.

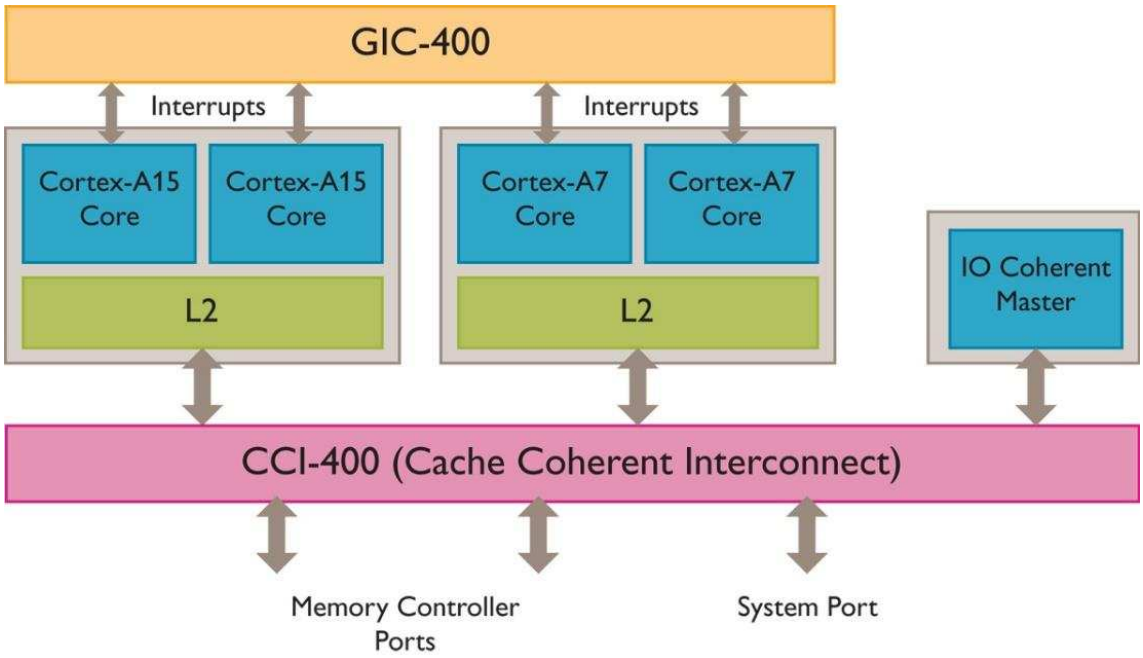

Figure 9 A Typical big.LITTLE System by ARM [23]

- *Single-ISA heterogeneous*: A single-ISA heterogeneous multicore processor [26] is composed of processing cores with the same instruction set architecture but diverse core implementations with respect to parameters such as clock speed, cache configuration, out-of-order execution support, etc. ARM's big.LITTLE architecture [23], an example of

typical single-ISA multicore system, can be seen in Figure 9. Compared to a homogeneous system, such heterogeneous design can provide a greater ability to adapt to specific demands of different applications/tasks for improvement in both performance and energy efficiency. Additionally, there is no need to rewrite software for specific core types and workload can be migrated freely among cores as all cores execute the same instruction set. However, such processors require an intelligent system resource management scheme to evaluate workload and choose the right execution strategy to attain its full capability.

- *Heterogeneous-ISA*: This is the most aggressive heterogeneous design pattern for multicore processors. Usually a heterogeneous-ISA multicore processor consists of one cluster of cores for general purpose processing, while the other cluster consists of application-specific processing units that provide hardware acceleration to heavy-weight tasks with improved speed and efficiency. This design paradigm is common in embedded systems with one or more cores for general-purpose computing and accelerator cores for data-intense computation. It also finds a place in personal computers, workstations, and data centers in the form of ***g****eneral-****p****urpose computing on* ***g****raphics* ***p****rocessing* ***u****nits* (**GPGPUs**) on a single chip. Figure 10 shows the ***a****ccelerated* ***p****rocessing* ***u****nit* (**APU**) processor developed at AMD, which is a typical example of a heterogeneous-ISA multicore processor with built-in general-purpose cores and graphics processing cores. While it possesses significant potential, the main obstacle to widespread implementation of this design paradigm is its demand for increased efforts in hardware/software co-design to best match a given workload to its highly customized architecture. Additionally, the workload must be partitioned to different core types at design-time, as it is almost impossible to migrate workload between cores with different ISAs on-the-fly.

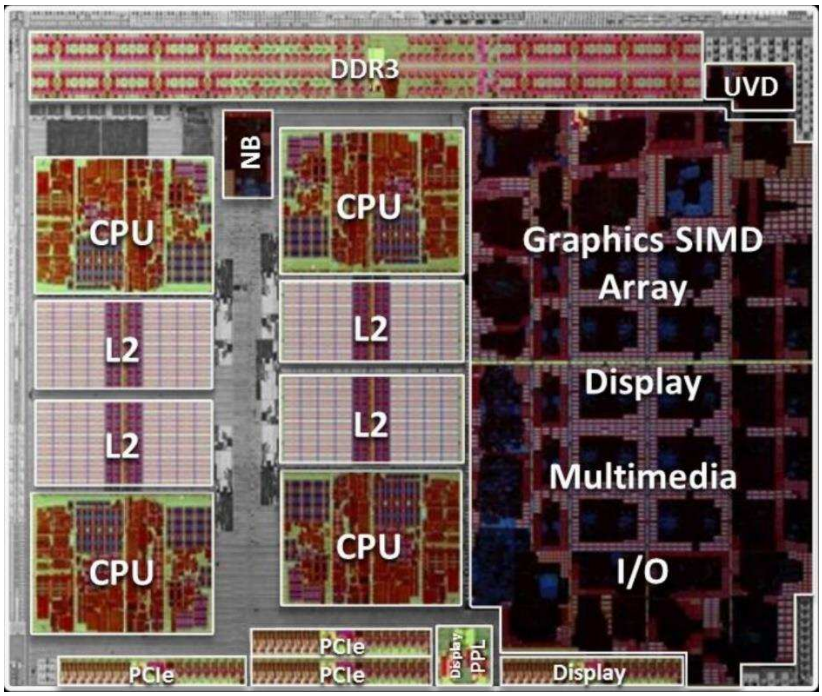

Figure 10 AMD Fusion APU: "LLANO" [27]

In this dissertation, we explore the problem of resource management for multicore embedded systems under energy constraints from solar energy harvesting. For most of this dissertation, we assume homogenous multicore processors as the target platform. However, we also consider single-ISA heterogeneous multicore processors to tackle the problem of mixed-criticality workload scheduling with energy harvesting. Even for homogenous multicore processors, our framework still considers core-heterogeneity caused by non-ideal factors such as process variations [28] and aging effects [29]. Principally, we consider multicore processors as the inevitable choice for systems powered by energy harvesting because of the benefits they provide in energy efficiency.

1.3. BACKGROUND AND RELATED WORK ON RESOURCE MANAGEMENT FOR LOW POWER REAL-TIME EMBEDDED SYSTEMS WITH ENERGY HARVESTING

Limitation in the energy budget is one of the major constraints facing embedded systems which can impact their availability, performance, or even correctness during execution. Traditionally, the operating duration of embedded systems with no external energy supply was limited by the energy budget provided by batteries. On the other hand, embedded systems powered by energy harvesting have a dynamically changing energy budget due to variations in the energy replenish rate from harvesting sources. For both cases, it is necessary to manage on-board resources intelligently to trade-off between timing performance and energy efficiency so that systems can operate more effectively. In this dissertation, we will focus on addressing this problem of energy optimization for multicore processors in real-time embedded systems.

***D**ynamic **v**oltage/**f**requency **s**caling* (**DVFS**) has been proven to be one of the most effective ways to make trade-offs between energy efficiency and computation performance for computing systems at run-time [30]. With this technique, processors can scale down supply voltage (V_{DD}) and operation frequency (f) on-the-fly to reduce dynamic power consumption [31]. The main reasons for its effectiveness are twofold:

- *Processors cannot or do not need to always execute at peak performance*: Processors can find slack in computation to slow down for energy savings whenever the system workload is not fully utilizing a processor. In most cases, processors are just not designed with the expectations to keep running at their full capability, especially with modern multicore processors hitting the power wall and facing thermal dissipation limits (See *power wall* discussion in Section 1.2.3). Additionally, embedded processors powered by energy harvesting may be forced to reduce energy consumption by lowering their voltage and/or

frequency at any time as there is no guarantee of a stable and sufficient energy budget to support a high operating voltage and frequency level at all times.

- *Processors are based on CMOS logic, which usually has much higher energy efficiency with lower clock speed*: at most operation (V_{DD}, f) points, the dominant power consumption for microprocessors is the dynamic component, which originates from the switching activity of CMOS logic gates. Dynamic power consumption of a processor is approximately proportional to its frequency, and to the square of its supply voltage, as shown in Equation (1), where C is the collective capacitive load of processor [32]:

$$P_{dynamic} = C \times {V_{DD}}^2 \times f \quad (1)$$

In addition, higher frequency requires higher supply voltage to avoid timing violations in synchronized CMOS logic. Thus, boosting execution frequency of a processor can lead to significant increase in power consumption (which typically increases energy consumption) and it is usually desirable to minimize execution frequency of processors whenever possible.

In this dissertation, we utilize the DVFS technique to control performance and energy consumption of real-time embedded systems powered by energy harvesting. Apart from DVFS, ***d**ynamic **p**ower **m**anagement* (**DPM**) is another approach for run-time energy optimization, which selectively turns off components or changes power states of electronic systems for energy saving [33]. In this dissertation, DPM is considered as a secondary mechanism for energy saving that is utilized under special scenarios, as turning off a component for some time with DPM can sub-optimally impact the over system, e.g., requiring higher execution frequency for other components or at later times to meet deadlines.

Many prior research projects have utilized DVFS techniques to optimize energy consumption of real-time processors dynamically. An early work [34] addressed the problem of power aware scheduling of periodic hard real-time tasks using DVFS. This study proved that an optimal execution frequency meant for energy minimization and meeting all task deadlines can be deduced for any periodic hard real-time policy that can fully utilize the processor (e.g. Earliest Deadline First, Least Laxity First). Another early work integrated DVFS scheduling algorithms with a **r**eal-**t**ime **o**perating **s**ystem (**RTOS**) to provide significant energy savings while maintaining real-time deadline guarantees [30]. In [35] algorithms were proposed to optimize energy consumption of homogeneous multiprocessors with DVFS support. They also considered co-optimization methods for the minimizing of energy consumption and task rejection penalty. *However, none of these papers consider the challenges arising from utilizing energy harvesting in real-time embedded systems.*

Solar energy harvesting is increasingly becoming an attractive solution in the quest to obtain clean sustainable energy for emerging embedded systems. Recently, a few papers have explored improvements in the efficiency and reliability of such systems ([36] [37] [38]). Some of these works focused on the implementation of energy harvesting systems and their energy conversion circuits (e.g., [38]). We are more concerned in this dissertation about related work on run-time management and scheduling for real-time embedded systems with energy harvesting. An early work [39] proposed the lazy scheduling algorithm (LSA) that executed tasks as late as possible, reducing deadline miss rates when compared to the classical ***e****arliest* ***d****eadline* ***f****irst* (**EDF**) algorithm. However, LSA does not consider DVFS and always executes tasks at full speed. Because a processor's dynamic power is generally a convex function of its operating frequency, running the processor at a frequency lower than the maximum frequency often results in higher

energy efficiency. In [40], the proposed energy-aware DVFS technique (EA-DVFS) takes processor DVFS into consideration for energy harvesting-aware scheduling. EA-DVFS utilized task slack to slow down execution speed, thereby achieving more energy savings than LSA, especially when total task utilization is low. Later the same authors proposed a more intelligent technique called harvesting-aware DVFS (HA-DVFS) [41], which improved energy efficiency by distributing multiple arriving tasks as evenly as possible over time and executing them with more uniform frequency. Recently, Chetto [42] proposed a semi-online EDF-based scheduling algorithm that is theoretically optimal. *However, these research efforts are only limited to uniprocessor systems and have not considered execution on multi-core platforms.*

There are a few notable research efforts that have considered multiprocessing with energy harvesting. In [36], a run-time framework is proposed for intelligently adjusting run-time system workload on multi-core platforms that use photovoltaic array for energy harvesting, so that the array works at its maximum operation points, producing more power for the computation system. However, the proposed work assumes grid utility as a backup energy source which may not be viable for many types of embedded systems. Also their approach is not applicable to real-time embedded systems with deadlines and operating constraints, which is the focus of this dissertation. A utilization-based technique (UTB) was proposed in [43] to better address periodic task scheduling in energy-harvesting embedded systems. UTB takes advantage of the predictability provided by the periodic task information for more efficient task allocation than in prior work. Moreover, UTB was extended to support multi-core platforms by allocating a subset of tasks to each core and executing the single-core UTB algorithm separately on each core. Zhang et al. [44] introduced a deadline-aware scheduling algorithm with energy migration strategies specifically designed to manage distributed supercapacitors in sensor networks.

In this dissertation, we propose a novel semi-dynamic approach for resource management of real-time multicore embedded systems that leads to significant improvement in energy efficiency while providing flexibility to simultaneously address other concerns such as thermal management, hybrid energy storage, allocation for heterogeneous multicore systems, task dependencies, transient faults, and processor aging effects.

1.4. DISSERTATION OUTLINE

In this dissertation, we propose a semi-dynamic resource management framework for multicore embedded systems powered by energy harvesting. A high level overview of the contributions we make is shown in Figure 11. The rest of this dissertation is organized as follows:

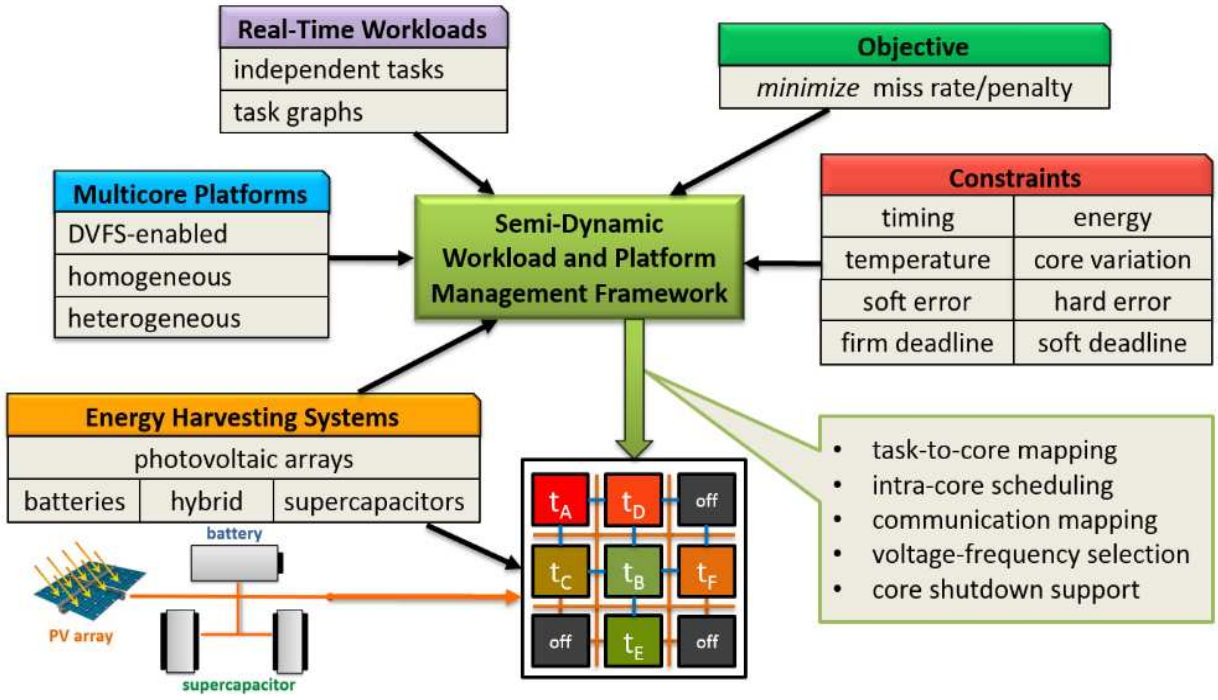

Figure 11 Preview of Contributions of this Dissertation

In Chapter 2, we present a ***s**emi-**d**ynamic scheduling **a**lgorithm* (**SDA**) for scheduling independent tasks on energy harvesting capable multicore embedded systems. The fundamental idea of our proposed semi-dynamic framework is to delay utilization of harvested energy by a certain amount of time, which is the length of all schedule windows, so that instantaneous harvesting power variations will not impact system execution immediately, allowing semi-dynamic adjustments of system strategies to utilize recently harvested energy intelligently with low scheduling overhead. We study the benefits of a semi-dynamic framework on stabilizing execution frequencies of processors even with power variations due to energy harvesting, which helps to reduce total energy consumption over time. Besides, the flexibility of the proposed semi-dynamic scheme allows further exploration and optimization for a number of related topics, such as hybrid energy storage system, core heterogeneity due to process variations, and overheating. Additionally, a dual-speed method is also introduced to overcome the performance impact of discrete frequency levels.

In Chapter 3, we apply our proposed semi-dynamic framework to the scheduling problem for task graphs with dependencies between tasks, resulting in a template-based scheduling algorithm. Compared to the previous contribution, here we address the even more difficult problem of scheduling task graphs with inter-node dependencies on systems that rely entirely on limited and fluctuating solar energy harvesting. As the limited energy supply prevents the deployment of complex scheduling algorithms at run-time, we propose a template-based algorithm in which scheduling complexity can be offloaded to design-time to pre-compute an execution strategy for task graphs. Note that our template-based algorithm still allows run-time execution adjustments so that a system can still address the problems of soft errors and aging effects on-the-fly. For design-time template generation, we propose two methods: one is a ***m**ixed **i**nteger **l**inear **p**rogramming*

(**MILP**) optimization method and the other one is a novel ***a****nalysis-based* ***t****emplate* ***g****eneration* (**ATG**) method.

In Chapter 4, we apply our semi-dynamic framework and template-based scheduling method to the problem of mixed-criticality scheduling on single-ISA heterogeneous multicore processors powered by energy harvesting. We considered a mixed-criticality workload set characterized by varying parallelism models, miss penalties, and deadline constraint types for tasks. A novel *timing intensity-aware penalty density* metric is introduced to estimate the importance of each task instance. With this metric, our proposed algorithm can find a balanced resource allocation dynamically for different mixed-criticality workload types so as to maximize overall system performance.

Lastly, Chapter 5 summarizes our research contributions and concludes this dissertation, with a discussion on future research directions.

2. SEMI-DYNAMIC SCHEDULING ALGORITHM FOR INDEPENDENT TASKS

In this chapter, we propose a novel framework for real-time energy and workload management in multi-core embedded systems with solar energy harvesting and a period real-time independent task set as the workload. Compared to prior work, our framework makes several novel contributions and possesses several advantages, including (i) a semi-dynamic scheduling heuristic that dynamically adapts to run-time harvested power variations without losing the consistency of periodic tasks, (ii) a battery-supercapacitor hybrid energy storage module for more efficient system energy management, (iii) a coarse-grained core shutdown heuristic for additional energy saving, (iv) energy budget planning and task allocation heuristics with process variation tolerance, (v) a novel dual-speed method specifically designed for periodic tasks to address discrete frequency levels and DVFS switching overhead at the core level, and (vi) an extension to prepare the system for thermal issues arising at run-time during extreme environmental conditions.

2.1. BACKGROUND AND CONTRIBUTION

Power and energy constraints have led to significant changes in the design of contemporary computing systems. In the last decade, ***t**hread-**l**evel **p**arallelism* (**TLP**) to improve performance within a power budget has seen widespread adoption across various computing platforms, ranging from high-end servers to desktops, as well as embedded devices. Recent years have also witnessed an increase in the use of multi-core processors in low-power embedded devices. With advances in parallel programming and power management techniques, embedded devices with multi-core processors and TLP support are outperforming single-core platforms in performance and energy efficiency [24].

As core counts continue to increase to keep up with rising application complexity, techniques for run-time workload distribution and energy management are the key to achieving energy savings in emerging multi-core embedded systems. Moreover, advances in parallel programming and increasing performance demands from embedded computing have forced implementations of high-end embedded processors composed of many cores running at the GHz level. Unfortunately, such increased performance levels in multi-core processors result in much higher power density than ever before, creating the risk of overheating when core utilization is high. Moreover, as CMOS technology scales down to integrate more cores on the same die area, process variations have become prominent, significantly impacting the system-level design and management of multi-core chips [28]. As the impact of time-varying power density and variations is hard to predict at design-time, it becomes critical to employ intelligent run-time techniques in emerging multi-core platforms that can adapt to these challenging system requirements.

For some embedded applications, we may require energy autonomous devices that utilize ambient energy to perform computations without relying entirely on an external power supply or frequent battery charges. Because it is the most widely available energy source, solar energy and its harvesting for embedded systems has attracted a lot of attention in recent years [36] [37] [45]. Due to the variable nature of solar energy harvesting, deployment of an intelligent run-time energy management scheme is not only beneficial but also essential for meeting system performance, robustness, and energy goals. To exploit the capabilities of energy harvesting systems, several prior efforts have explored workload scheduling for embedded systems with real-time tasks [39] [40] [41] [43]. An early work [39] proposed the ***l**azy **s**cheduling **a**lgorithm* (**LSA**) that executed tasks as late as possible, reducing deadline miss rates when compared to the EDF algorithm. However, LSA does not consider DVFS and always executes tasks at full speed. Because a

processor's dynamic power is generally a convex function of frequency, operating the processor at a frequency lower than the maximum frequency often results in higher energy efficiency. A ***ut**ilization-**b**ased **t**echnique* (**UTB**) was proposed in [43] to better address periodic task scheduling in energy-harvesting embedded systems. UTB takes advantage of the predictability provided by the periodic task information for more efficient task allocation than in prior work. Moreover, UTB was extended to support multi-core platforms by allocating a subset of tasks to each core and executing the single-core UTB algorithm separately on each core. More discussion on related work in the field of scheduling with solar energy harvesting can be seen in Section 1.3. Besides, there are many relevant research projects in the field of energy optimization for embedded systems that do not consider energy harvesting. A Li-Ion battery-supercapacitor hybrid storage system that supports a long lifetime, wireless sensor network was described in [46], presenting a good example of hybrid energy system design, from which we derives a customized hybrid storage system in this chapter. In [47] the HypoEnergy framework was proposed to extend power supply life-time of hybrid battery-supercapacitor systems. An algorithm for application scheduling and power management of chip multiprocessors with awareness of within-die processor variations was proposed in [48]. In [49], a thermal-aware task allocation and scheduling algorithm was proposed which was used as a subroutine for hardware/software co-synthesis.

In this chapter, we propose a novel ***s**emi-**d**ynamic **a**lgorithm* (**SDA**) based framework with energy budgeting that manages energy and workload allocation at run-time for multi-core embedded systems with solar energy harvesting capability. Our framework aims to minimize deadline miss rate and penalty of periodic tasks in the presence of variant and insufficient energy harvesting conditions. In addition, our framework possesses the flexibility to be able to

accommodate other goals, such as run-time thermal management and process variation aware workload distribution. The novelty and main contributions of this work are summarized as follows:

- Unlike prior work, SDA reacts to run-time energy shortages and fluctuations proactively to find significantly greater scope for energy savings, especially in multi-core platforms.
- A hybrid energy storage system is designed to decouple the run-time management scheme from variations in energy harvesting, as well as to enhance charging/discharging efficiency.
- The energy and task distribution heuristics in SDA take system heterogeneity into consideration by assigning workloads with awareness of variations due to within-die process variations.
- At the core level, a novel dual-speed frequency selection method is deployed to combine two neighboring discrete frequency levels for superior energy efficiency with awareness of dynamic voltage/frequency switching overhead.
- Our framework cooperates with basic throttling mechanisms to tackle processor overheating. Additionally, it dynamically re-allocates workload or shuts down cores for more proactive multi-level throttling to reduce the occurrences and overhead of system overheating.

Our experimental studies show that our framework is able to outperform the best known prior work (UTB [43]) on run-time management of periodic tasks for real-time systems with energy harvesting, achieving superior task drop penalty/rate reduction and energy efficiency. Additionally, our framework also provides the flexibility to adapt to run-time thermal variations and supports core heterogeneity-aware workload distribution.

2.2. PROBLEM FORMULATION

Our focus of this chapter is on the problem of effective workload and energy management for real-time multi-core embedded systems running periodic tasks, and powered by solar energy, as shown in Figure 12. The following sections describe the key components of our system model.

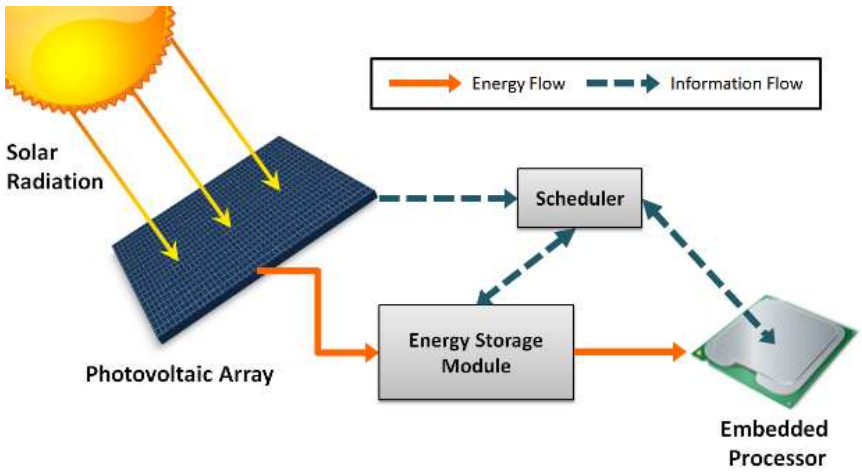

Figure 12 Real-Time Embedded Processing with Solar Energy Harvesting

2.2.1. ENERGY HARVESTING AND ENERGY STORAGE MODULE

A ***p**hotovoltaic* (**PV**) array is used as a power source for our embedded system, converting ambient solar energy into electric power. Naturally, the amount of harvested power varies over time due to changing environmental conditions, like angle of sunlight incidence, cloud density, temperature, humidity, etc. To cope with the unstable nature of the solar energy source, rechargeable batteries and supercapacitors can be used to buffer solar energy collected by photovoltaic cells. In our study, the converted solar power at time t is denoted as $P_H(t)$. The energy E_H charged into the energy storage system between time instances t_1 and t_2 is given by:

$$E_H(t_1 \sim t_2) = \eta_{\text{chrg}} \int_{t1}^{t2} P_H(t)\, dt \quad (2)$$

where η_{chrg} is a coefficient between 0 and 1 to represent charging efficiency of the energy storage system. The capacity of the energy storage device is limited and clearly harvested energy will be wasted if the energy storage device is already fully charged. We assume that task execution must be halted when the remaining energy in the system goes below a specified threshold. This step is essential to maintaining the system state and ensure graceful shutdown.

2.2.2. PERIODIC REAL-TIME WORKLOAD WITH INDEPENDENT TASKS

In many real-world applications, an energy autonomous embedded system powered by solar energy harvesting is deployed to execute certain types of repetitive lightweight real-time tasks, such as sensing, controlling, and data preprocessing. We assume a task set of N independent periodic real-time tasks $\psi: \{\tau_1, \ldots, \tau_N\}$ for such use cases, in which each periodic task τ_i has a characteristic triplet (C_i, D_i, T_i), $i \in \{1, \ldots, N\}$. C_i is the maximum number of CPU clock cycles needed to finish a job instance of task τ_i, referred to as the ***w**orst-**c**ase **e**xecution **c**ycles* (**WCEC**). The relative deadline of the task, D_i, is the time interval between a job's arrival time and its firm deadline (see Section 1.2.2). A job instance is missed if it is not finished before its deadline. T_i is the period of the task. At the beginning of each period, a new job instance of that task will be dispatched to the system. Like most recent works on periodic task scheduling (e.g., [43]) we assume that D_i equals T_i, with all jobs expected to finish before the arrival of the next job instance of the same task. We also define an attribute X_i, which is the miss penalty associated with each task. Each time that a task's job misses its deadline, the job will be aborted and the penalty applied to the system. Thus, we can refine the triplet for task τ_i as (C_i, T_i, X_i). The relative importance of a

task can be characterized by a *penalty density* parameter, defined as the ratio of the task miss penalty and WCEC (X_i/C_i) [35]. In this chapter, we assume the system is designed to execute one set of periodic real-time tasks consistently and information of tasks such as execution time and miss penalty is profiled at design-time and thus is available to the run-time scheduler.

Table 1 Xscale Processor Power and Frequency Levels [43]

Level	0	1	2	3	4	5
Voltage(V)	-	0.75	1.0	1.3	1.6	1.8
Power(mW)	40	80	170	400	900	1600
Frequency(MHz)	idle	150	400	600	800	1000
Energy Efficiency	0	1.875	2.353	1.5	0.889	0.625

2.2.3. DPM AND DVFS-ENABLED MULTI-CORE PROCESSOR

We consider an embedded system with a low power multi-core processor that has support for task preemption. We assume that the frequency of each core can be adjusted individually (i.e., the processor possesses per-core DVFS capability) as observed in recent implementations with this capability enabled in industry and academia [50] [51]. Each core has *M* discrete voltage and frequency levels: $\varphi: \{L_0, \ldots, L_M\}$. Each level is characterized by $L_j: (v_j, p_j, f_j), j \in \{1, \ldots, M\}$, which represents voltage, average power, and frequency respectively. We consider power-frequency levels of the Xscale processor as shown in Table 1. Here, level 0 represents the idle power of the processor when no task is executed while the system stays in active state. Typically, the dynamic power-frequency function is convex. Thus, a processor running at lower frequency can be expected to execute the same number of cycles with lower energy consumption. However, this is not always the case due to the increasing prominence of leakage power in recent CMOS technologies. To find an energy optimal frequency, we represent energy efficiency of a *v-f* level L_i by δ_i = *cycles executed/energy consumed* = f_i/p_i. From Table 1 we can conclude that level 2 is the most energy

efficient because executing at this level consumes the least energy for a given number of cycles. The most energy efficient level is often called **critical level** in the literature and thus $f_{crt} = f_2$ [52]. Although it is desirable to execute tasks at this critical frequency level for energy-efficiency, executing tasks at f_{crt} may end up being insufficient to finish all task instances by their deadlines, due to the unique timing constraints of each task. As we also consider inter-core heterogeneity caused by within-die process variations, some cores have lower maximum frequency and higher static power values than for the ideal case. For each core, unsupported *v-f* levels are blocked to ensure system stability.

The utilization of a periodic task (U) is defined with respect to the full speed (maximum frequency) provided by the processor. A task's utilization is its execution time under the maximum frequency divided by its period:

$$U_i = \frac{C_i / f_{max}}{T_i} \tag{3}$$

The utilization for an entire task set is simply the accumulation of the utilization for all the tasks in the set. In preemptive real-time systems, a task set is schedulable by the ***e**arliest **d**eadline **f**irst* (**EDF**) algorithm for a frequency j if it meets the following condition:

$$U_{total} \leq \frac{f_j}{f_{max}} \tag{4}$$

When total task set utilization is known, the most energy efficient frequency can be deduced from this equation, assuming $f_j \geq f_{crt}$ [34].

Also, unlike any prior work, we consider thermal management in an energy harvesting multi-core processing environment. We assume that each core in the multi-core processor has a ***d**igital **t**hermal **s**ensor* (**DTS**) implemented to monitor run-time temperature independently [53]. We set 85°C as the thermal *setpoint* at which throttling is initiated to halt all processor execution (i.e.,

throttling threshold = 85°C) [54]. When throttling is triggered, a core must halt execution and shift to idle state until its temperature drops to 80°C.

2.2.4. RUN-TIME SCHEDULER

This module is an important component of the system for information gathering and execution control. The scheduler dynamically gathers information by monitoring the energy storage medium and multi-core processor state (Figure 12). The gathered data, together with offline-profiled information about task execution times and energy consumption on cores informs a management algorithm in our scheduler that coordinates operation of the multi-core platform at run-time. Each core is eventually assigned a strategy by the scheduler to guide intra-core task execution.

2.2.5. SCHEDULING PROBLEM OBJECTIVE

Our primary optimization objective is to perform task allocation and scheduling at run-time such that total task miss rate (or penalty) is minimized. Our technique must react to changing harvested energy dynamics to complete as much (critical) work as possible, thus maximizing overall system utility and cost effectiveness. Further, our task allocation should be cognizant of processor thermal behavior and frequency limits of each core (due to process variations) to ensure system stability.

2.3. MOTIVATION

2.3.1. MOTIVATION FOR SEMI-DYNAMIC ALGORITHM

In this section, we present the motivation for applying our semi-dynamic algorithm to the problem of workload and energy management in energy harvesting multi-core systems.

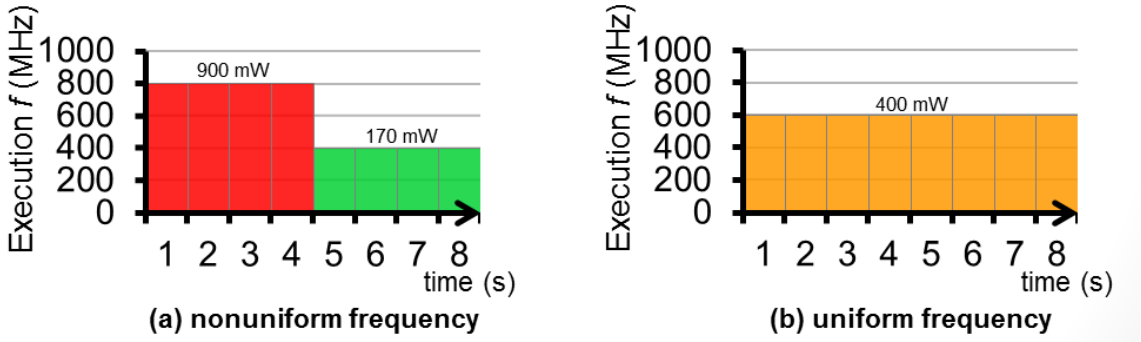

Figure 13 Real-Time Scheduling with Energy Harvesting

2.3.1.1. IMPORTANCE OF BALANCED WORKLOAD EXECUTION

As dynamic power consumption in processors is typically a convex function of frequency, increasing the processor frequency level can lead to significantly higher power consumption and much lower energy efficiency, as shown in Table 1. Imbalances in workload allocation require sub-optimally changing voltage-frequency levels that can result in higher power consumption than for a balanced workload allocation case. To illustrate this point, we compare average power consumption for two different schedules in Figure 13, both of which execute a workload for 4.8 billion cycles within 8 seconds. The schedule in Figure 13(a) executes with non-uniform speeds (800MHz and 400MHz) while the one in Figure 13(b) has uniform execution speed fixed at 600MHz. A simple analysis based on Table 1 shows that the schedule in Figure 13(b) is more energy efficient with average power consumption of 400 mW compared to 535 mW for the

schedule in Figure 13(a). This example highlights how maintaining a uniform execution speed is critical for energy efficiency, which in turn motivates the need for an intelligent run-time management approach that minimizes instances of workload imbalance across cores over time.

2.3.1.2. SDA FRAMEWORK FOR RUN-TIME WORKLOAD DISTRIBUTION

In this section, we provide a motivational example to illustrate the benefits of our SDA framework that integrates energy budgeting to achieve better workload distribution at run-time than in existing approaches, under varying solar energy harvesting scenarios.

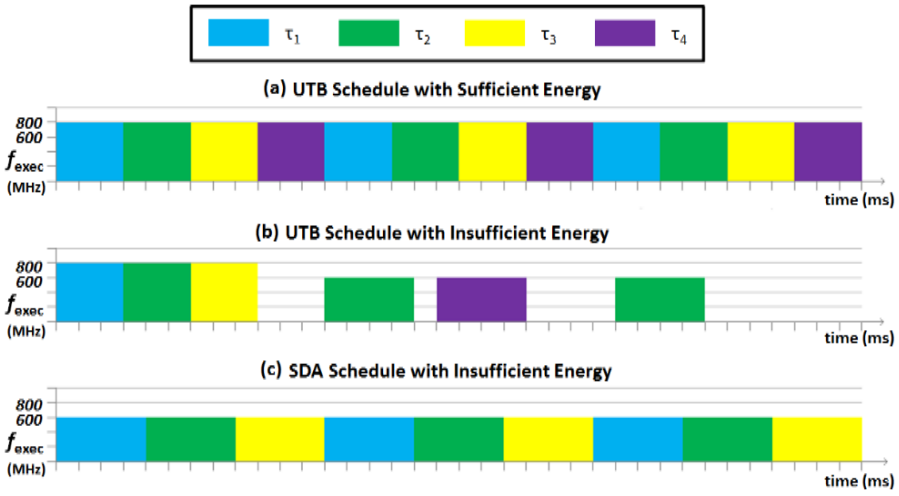

Figure 14 Motivation for Proposed Semi-Dynamic Approach

Most prior work deals with dynamic solar energy variations by halting, dropping, or speeding up the execution of a current task, changing instantly from an initial schedule deduced offline. For energy harvesting aware periodic task set scheduling, the best known prior work, UTB [43], also follows this strategy. Although UTB deduces an optimal initial schedule offline assuming sufficient energy, it does not cope well with run-time energy variations, and there is scope for notable improvements as discussed below:

- The task dropping mechanism in UTB reacts to run-time energy shortages *passively*, only when the current task lacks sufficient energy to finish in time. In the motivational example shown in Figure 14, we assume a task set with four periodic tasks ($\tau_1 \sim \tau_4$), where each task has WCEC of 2.4 million CPU cycles and a task period of 12ms. According to Table 1, Equation (3) and Equation (4), UTB initially sets execution frequency to 800MHz so that all tasks can finish with the best efficiency if energy is sufficient, as shown in Figure 14(a). However, the real challenge arises when the run-time energy budget is insufficient. Let us assume that the remaining energy in the energy storage is 7200μJ and harvested power in the next 36ms (3 periods) is 200mW, i.e., 200μJ of incoming energy per microsecond. After finishing three jobs, the energy storage is depleted, and UTB has to drop jobs due to insufficient energy, as shown in Figure 14(b). Only 6 out of 12 job instances are finished with UTB, resulting in a high 50% miss rate. With the same energy budget, our proposed SDA technique copes with energy shortage by *proactively* dropping tasks. It drops one task, τ_4, based on the energy budget which helps to execute the remaining tasks steadily at a lower frequency of 600 MHz. According to Table 1, executing at 600MHz corresponds to a power consumption of 400mW, which is dramatically lower than 900mW at 800MHz due to the nonlinear relation between frequency and power consumption. As can be seen in Figure 14(c), all accepted job instances for $\tau_1 \sim \tau_3$ are finished and the overall miss rate is 25%, which is significantly lower than the 50% miss rate achieved by UTB.
- UTB encourages dropping tasks with longer execution time, because finishing them requires more energy than other tasks. This biased dropping may be undesirable for real-time applications, as tasks with longer execution time may represent complex applications of high priority. Moreover, it is nontrivial to add priority awareness into UTB due to its

passive task dropping scheme mentioned above. Our SDA framework allocates tasks and performs task dropping with the awareness of the miss penalty corresponding to each task.

- On multi-core platforms, UTB partitions tasks into separate sets and then executes each set on a core using a single-core scheduling algorithm. However, as all cores are dependent on the same energy source, such isolated run-time adjustment is not amenable to learning upcoming energy requirements of other cores, leading to sub-optimal schedules. SDA avoids inter-core energy resource contention by allocating tasks based on energy budgets assigned to each core. In addition, static task partitioning in UTB wastes the flexibility provided by a multi-core platform. In contrast, SDA triggers task reallocation dynamically for improved results.

In summary, we found several limitations with the best known prior work on energy harvesting-aware energy and workload management. Our SDA scheme is designed to address these limitations and improve upon prior work. In the following sections, we discuss other issues related to multi-core embedded systems powered by solar energy harvesting. To cope with these issues, we exploit the flexibility of SDA to integrate hybrid energy storage, heterogeneity-aware task allocation, and run-time thermal management, forming a cross-layer design that improves performance, stability, and adaptivity of target systems.

2.3.2. MOTIVATION FOR HYBRID ENERGY STORAGE

Most prior efforts on harvesting-aware task scheduling assume a near-ideal battery as the energy storage medium that is limited merely by its capacity, ignoring other factors such as nonlinear efficiency, slow charge rate, and limited lifetime in terms of recharge cycles [55]. When applied to real-world platforms, overlooking these factors can result in suboptimal or even

unrealistic design and scheduling techniques that diminish system efficiency, stability, and lifespan. For example, the rate capacity effect leads to decreasing battery capacity when discharging current increases [47]. Supercapacitors present an interesting alternative to batteries for energy storage with benefits over electro-chemical batteries, such as orders of magnitude higher recharge cycles, ease of charging, and significantly higher energy efficiency. However, high capacity supercapacitors are not practical for small-package low-power embedded systems due to their significantly lower energy density and higher leakage overhead than an electro-chemical battery, even with the state-of-art supercapacitor technology [56]. Recent work has shown that a battery-supercapacitor hybrid system can overcome the limitations of both types of energy storage mediums [47] [46]. Therefore we employ a hybrid energy storage system for our work.

2.3.3. MOTIVATION FOR HETEROGENEITY-AWARE ALLOCATION

As CMOS feature sizes continue to scale, process variations in manufacturing are becoming more and more prevalent, causing performance asymmetry within a chip. For multi-core processors, within-die process variations differentiate critical path delays among cores such that the maximum frequencies supported by cores may diverge from their nominal specification [28]. Without awareness of this undesirable inter-core heterogeneity, a run-time management scheme may distribute excessive workload to slower cores. Even worse, faulty schedules that try to finish these excessive workloads will be deployed, ending up with a high miss rate due to energy and CPU time being wasted on tasks that cannot be finished in time. Overclocking slower cores is a possibility, but is often not a viable option due its high likelihood of causing timing violations on the critical path. Thus an appropriate run-time energy management framework must consider inter-core frequency variations; otherwise it may lower system performance by causing task overloading

on certain cores, which can create workload imbalances that also additionally reduce the energy efficiency of the entire system.

2.3.4. MOTIVATION FOR RUN-TIME THERMAL MANAGEMENT

The motivations for considering run-time thermal management for energy harvesting based multi-core embedded systems are:

- Limited power budgets and form factors of embedded systems make it uneconomical, if not inapplicable, to apply aggressive cooling techniques used on desktop and server systems, such as cooling fans and large heat sinks. With increasing power density and absence of active cooling, high performance multi-core embedded processors can easily end up causing thermal emergencies during their long operation periods. Such overheating of processors is known to harm system reliability and stability. A throughput-focused run-time management scheme that ignores this risk may fail to maintain system stability and end up with thermal runaway. Perhaps most importantly, frequent thermal throttling that is initiated in processors to cope with thermal emergencies may end up disrupting balanced scheduling strategies, reducing system performance and overall energy efficiency.
- Due to the inherent nature of solar energy, solar energy harvesting systems tend to receive abundant energy to run at full speed around the middle of the day. However, continuously executing at full-speed creates excessive heat in the processor package and can lead to overheating issues. Around the same time, the ambient temperature is also usually the highest in the day (Figure 15), making it even more difficult for the processor to cool down around those hours without intervention.

- Thus there is a critical need to consider run-time thermal management strategies for energy harvesting based embedded systems as thermal issues can have a notable impact on the performance, energy efficiency, and reliability of such systems.

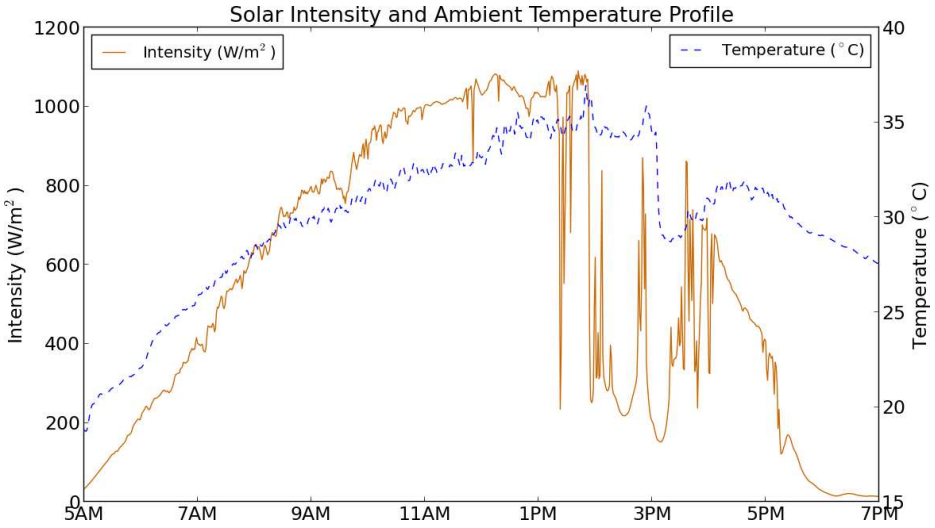

Figure 15 An Example of Solar Intensity vs. Ambient Temperature

2.4. PROPOSED RUN-TIME ENERGY AND WORKLOAD MANAGEMENT FRAMEWORK

2.4.1. SEMI-DYNAMIC ALGORITHM OVERVIEW

In this section, we present a holistic overview of our novel energy and workload management framework based on a ***s**emi-**d**ynamic **a**lgorithm* (**SDA**). Subsequent sections present more details of each major component in our SDA-based framework.

One of the underlying ideas behind SDA is to exploit time-segmentation during energy management, as illustrated in Figure 16. At each specified time interval (epoch), there is a reschedule point, where the execution strategy can be adjusted based on the energy budget provided by the energy storage system. A time frame between two reschedule points is called a

schedule window, within which the strategy specified at the prior reschedule point is in effect until the next reschedule point. Thus reschedule points provide dynamic adaptivity needed by the energy harvesting aware system to adjust the task execution strategy, while the schedule window enables stable execution that utilizes periodic task information for better energy efficiency, as illustrated in Figure 14(c). For example, from schedule window 1 to 4 in Figure 16, it can be seen that under low energy conditions, SDA maintains execution at optimal low (critical) frequency with different number of cores activated. Cores only execute at higher frequency when the energy harvested is abundant as in schedule windows 6 and 7. In this manner, SDA can provide better execution efficiency to improve performance under variable solar radiance conditions.

At each reschedule point, we update the execution strategy for the upcoming schedule window with a rescheduling scheme composed of three stages:

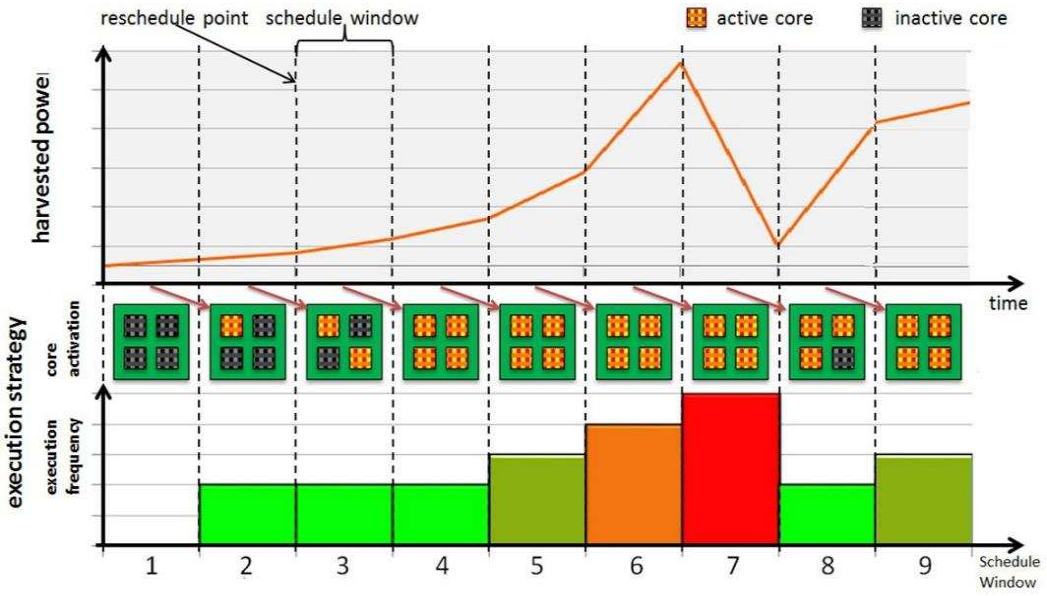

Figure 16 Illustration of Semi-Dynamic Algorithm

- *Energy budgeting:* This stage estimates the energy budget available for the upcoming schedule window based on the status of the hybrid energy storage system. Estimating the energy budget decouples run-time system management from energy variations in the environment, making it possible to deduce a stable balanced execution strategy that maximizes energy efficiency.
- *Workload estimation:* This second stage evaluates the amount of workload that can be supported by the energy budget, and forks into two separate paths. When energy budget is below a threshold, E_{th}, the first path is chosen with a focus on active-core selection to improve energy efficiency under a low energy budget. When energy budget is above E_{th}, the second path is chosen with a focus on variation-aware workload assignment to ensure that no core is required to run at a frequency higher than its maximum limit. Note that there is no need to consider active core selection and variation-aware assignment at the same time, as maximum frequency variation only matters when the energy budget is high and active core selection only helps when energy budget is very low (Section *2.4.3.1* and *2.4.3.2*). Additionally, this stage can proactively reduce workload when thermal issues arise at run-time.
- *Task rejection and allocation:* Based on the amount of workload estimated by the previous stage, this stage takes the periodic task set and filters out the subset of tasks that are less important. The remaining tasks are accepted for execution and are allocated to cores with awareness of core heterogeneity.

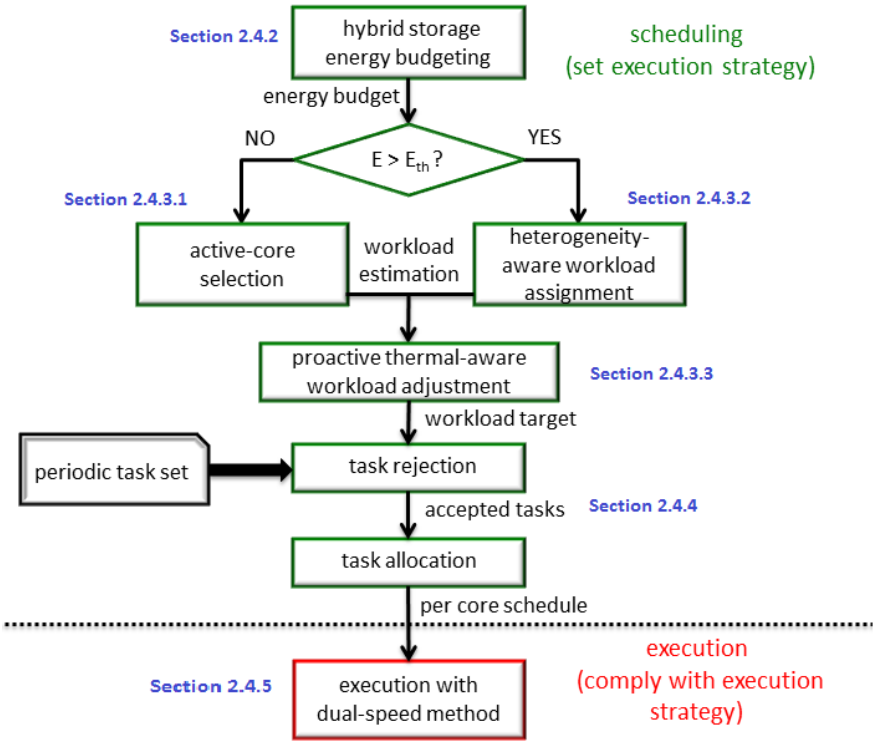

Figure 17 Design Flow of Our Proposed SDA-Based Framework

These three stages are organized in an order such that successor stages make use of efforts made by previous stages, rather than diminishing them, and are described in the following sections (Sections 2.4.2, 2.4.3, and 2.4.4). After the execution strategy is fixed for a schedule window, cores apply a dual-speed switching method to improve energy efficiency in the presence of discrete frequency levels, which is discussed in Section 2.4.5. The complete design flow of our proposed SDA framework is shown in Figure 17.

2.4.2. HYBRID ENERGY STORAGE SYSTEM AND ENERGY BUDGETING

In this section, we describe our hybrid energy storage system and its management policy that determines the energy budget for the upcoming schedule window, thereby isolating run-time task scheduling from fluctuations in solar energy harvesting.

2.4.2.1. BATTERY-SUPERCAPACITOR HYBRID ENERGY STORAGE

Inspired by [46], we propose a hybrid energy storage system with one Li-Ion battery and two separate supercapacitors connected by a dc bus, as shown in Figure 18. During each schedule window, one capacitor is used to collect energy extracted from the PV array, while the other one is used as a power source for system operation or battery charging. At each reschedule point, the two supercapacitors switch their roles. Supercapacitors charge the battery only when their saved energy exceeds peak requirements of processors running at full speed. The PV array, battery, and supercapacitors are coupled with bidirectional dc-dc converters to serve the purpose of voltage conversions between components with ***m**aximum **p**ower **p**oint **t**racking* (**MPPT**) [38] and voltage level compatibility. This hybrid battery and dual-supercapacitor design has several advantages over a non-hybrid system:

- The supercapacitors can support embedded processors directly, taking advantage of a much lower charging/discharging overhead compared to a battery.
- The electro-chemical battery offers high capacity to preserve energy especially in scenarios with excessive harvested energy. On the other hand, the capacity requirement of supercapacitors is much smaller.
- The supercapacitor with energy buffered during the last schedule window acts as a known stable energy source for the system in the upcoming schedule window. Thus our energy budgeting does not require energy harvesting power predication. Besides, the stable energy source makes it possible to charge the battery with a steady constant current for more effective charging [55].

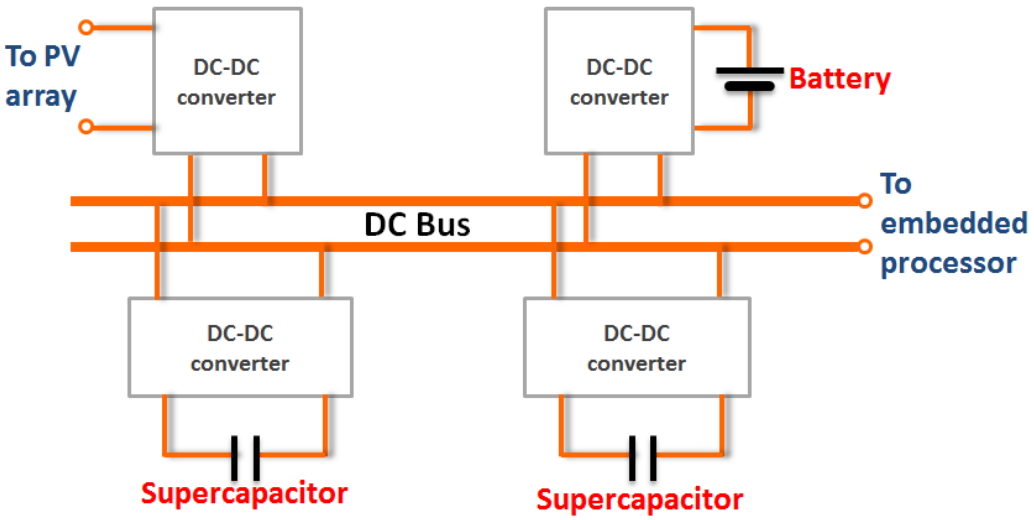

Figure 18 Proposed Hybrid Energy Storage System

2.4.2.2. HYBRID ENERGY STORAGE BASED ENERGY BUDGET

We propose an energy budgeting heuristic that selects among energy sources (supercapacitors and battery), sets the amount of energy to charge the battery for (E_{chrg}), and assigns the energy budget for system execution in the upcoming schedule window (E_{budget}), as shown in Algorithm 1. The heuristic is based on storage levels of the battery (LV_B) and supercapacitor (LV_C) with range 1, 2, and 3, representing respectively charge level of low, medium, and high. LV_C is classified into three levels (lines 1-3) based on two thresholds: i) energy budget to execute a single core at critical frequency (E_{crt}) and ii) energy budget to execute all cores at maximum frequency ($E_{max} \times NUM\_CORE$). As we want to avoid battery charging/ discharging overhead, there are only two scenarios where the battery is selected as a power source: i) when energy harvested in the supercapacitor is below a critical level ($LV_C = 1$); and ii) when battery storage level is high ($LV_B = 3$) such that battery overflow becomes a possibility (line 4). The battery is charged only when energy in the supercapacitor exceeds peak requirements of the processor (lines 12-14). This hybrid storage management and energy budgeting policy is shown in Figure 19.

Algorithm 1 Energy Budgeting with Hybrid Energy Storage

Input:
(i) E_{cap}, harvested energy in charged capacitor
(ii) LV_B, battery energy storage level
(iii) E_{crt}, energy budget to execute one core at critical frequency
(iv) E_{max}, energy budget to execute one core at maximum frequency
(v) *NUM_CORE*, number of cores in embedded processor
Output: E_{budget}, assigned energy budget for next schedule window

1. **if** $E_{cap} < E_{crt}$: $LV_C \leftarrow 1$
2. **else if** $E_{cap} > E_{max} \times NUM\_CORE$: $LV_C \leftarrow 3$
3. **else** : $LV_C \leftarrow 2$
4. **if** $LV_B > LV_C$:
5. set to discharge battery
6. **if** $LV_B = 2$: $E_{budget} \leftarrow E_{crt} \times NUM\_CORE$
7. **if** $LV_B = 3$: $E_{budget} \leftarrow E_{max} \times NUM\_CORE$
8. **else** :
9. set to discharge supercapacitor
10. **if** $LV_C = 1$: $E_{budget} \leftarrow 0$
11. **if** $LV_C = 2$: $E_{budget} \leftarrow E_{cap}$
12. **if** $LV_C = 3$:
13. $E_{budget} \leftarrow E_{max} \times NUM\_CORE$
14. $E_{chrg} \leftarrow E_{cap} - E_{budget}$

The resulting energy budget, E_{budget}, reflects the amount of energy dynamically collected from the energy harvesting system at run-time and can be considered as a stable energy supply for the next schedule window so that a uniform execution strategy can be enabled for energy efficiency.

discharging battery battery stays idle charging battery

status/policy	LV_B = low	LV_B = medium	LV_B = high
LV_C = low	budget = 0	budget = E_{crt}	budget = Emax
LV_C = medium	budget = E_{cap}	budget = E_{cap}	budget = E_{max}
LV_C = high	budget = E_{max}	budget = E_{max}	budget = Emax

Figure 19 Hybrid Storage Management Policy

2.4.3. CRITICAL FREQUENCY, CORE HETEROGENEITY AND THERMAL AWARE WORKLOAD ESTIMATION

This section describes our approach for energy budget-based workload estimation at the beginning of each schedule window, which intelligently estimates the optimal workload to be allocated for each core while considering energy efficiency, core heterogeneity, and temperature distributions.

At each reschedule point, our scheme first estimates the amount of workload that can be supported in the upcoming schedule window using the energy budget provided by the hybrid energy storage system. As shown in Figure 17 earlier, this stage forks into two paths based on the energy budget threshold, E_{th}. As discussed in Section 2.2.3, multi-core processors may have cores that have a lower maximum frequency due to within-die process variations. We assume that within-die variations are measured after manufacturing by variation acquisition methods, such as vMeter, proposed in [57], and maximum frequency of each core is considered as known to the run-time manager. The energy budget threshold, $E_{th,}$ is defined as the energy budget required for the slowest core to run at its maximum frequency. As we assume even the slowest core is able to run above critical level, it is always true that $E_{th} > E_{crt}$. When the average budget per core is below E_{th}, uniform workload distribution is sufficient to ensure that every core runs below its maximum frequency and the run-time manager focuses on active core count selection for energy savings. On the other hand, when the average energy budget for each core is higher than E_{th}, the core heterogeneity cannot be ignored and the run-time manager switches to a heuristic that activates all cores and estimates workload based on each core's achievable frequency. Apart from workload estimation, this stage also takes core temperatures into consideration for proactive run-time thermal management. The final outputs of this stage are the cores to activate and the workload to

support in the upcoming schedule window. The following subsections describe the three main components of this stage.

2.4.3.1. CRITICAL FREQUENCY-AWARE ACTIVE CORE SELECTION

We propose a heuristic that selects the number of cores to activate and workload to allocate on each core, assuming uniform workload distribution among activated cores. The motivation for this active core selection heuristic (that is executed only for low energy budget scenarios) is that running a processor below its critical frequency decreases energy efficiency, as can be seen from Table 1. This situation can occur when the energy budget is so low that only a small subset of tasks can be accepted for execution, i.e., after evenly distributing these tasks to all cores, utilization on each core is smaller than maximum utilization supported by the critical frequency. With our active core selection heuristic, we can shut down some cores at each reschedule point based on the estimated energy budget. The power dissipated by inactive cores is negligible and the remaining cores can then receive enough workload to run at critical frequency. Also the associated power state switching overhead is minimal as we only trigger core shutdown at reschedule points. However, arbitrarily shutting down cores to reach a frequency higher than critical is not always optimal. Figure 20 shows the maximum energy-efficiency for different frequencies on the XScale processor. Suppose cores execute at point *A* without shutdown. After shutdown of one core, the extra power budget allows us to run the remaining core(s) at higher frequencies such as *B, C,* or *D*. But not every higher frequency is viable, e.g., frequency *D* leads to even lower energy efficiency than A, before shutdown! Thus it is important to compare resulting energy efficiencies before making a core shutdown decision.

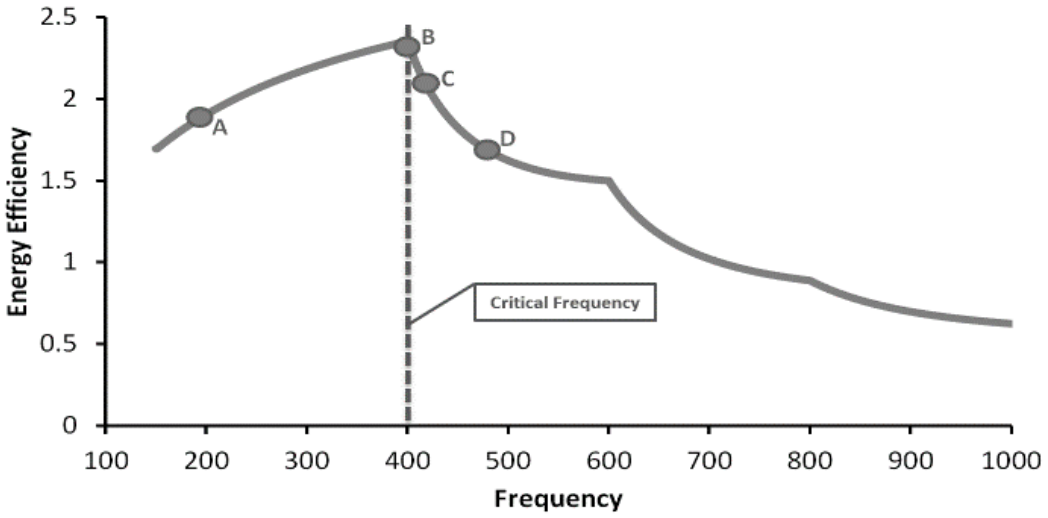

Figure 20 Energy Efficiency of XScale Processor

The pseudo code of the active core selection heuristic is given in Algorithm 2a. The core shutdown procedure is triggered when the energy budget is unable to support all active cores to execute at their critical frequency (line 2). Subsequently (lines 3-10) if one less active core results in a better efficiency, $\delta(U_{num\_core-1}) > \delta(U_{num\_core})$, then the scheduler shuts down one core. If the energy budget for the current schedule window is extremely low, eventually all cores in the system will be shut down to save harvested energy for future execution. Recursively, these steps set the number of cores to keep active. Finally, the objective task-set utilization (i.e., the amount of workload that the system can support) is obtained by aggregating the supported utilization of each core (line 11). As a result of this selection heuristic, the number of cores activated is tightly related to the energy budget available.

Algorithm 2a Active Core Selection and Workload Estimation

Input:
(i) E_{budget}, energy budget for coming schedule window
(ii) $\delta(U)$, dual-speed method energy efficiency profile (see Section 2.4.5) for task utilizations from 0 to 1
Output: U_{obj}, objective utilization for next schedule window

1. num_active ← NUM_CORE, $E_{per\_core} = E_{budget}$/num_active
2. **while** $E_{per\_core} < E_{crt}$ **and** num_active > 0 :
3. $E_{num\_core} \leftarrow E_{budget}$ / num_active
4. $E_{num\_core-1} \leftarrow E_{budget}$ / (num_active-1)
5. calculate f_{num_core-1} and f_{num_core}, maximum frequencies supported by E_{num_core-1} and E_{num_core}
6. based on Inequation (4), calculate U_{num_core-1} and U_{num_core}, maximum utilization supported by f_{num_core-1} and f_{num_core}
7. look up profile for δ (U_{num_core}) and δ (U_{num_core-1})
8. **if** δ(U_{num_core-1}) > δ(U_{num_core}) :
9. num_active ← num_active – 1
10. update E_{per_core} , U_{per_core}
11. $U_{obj} \leftarrow U_{per\_core} \times$ num_active

2.4.3.2. CORE HETEROGENEITY-AWARE WORKLOAD ESTIMATION

When per-core average energy budget for the next schedule window, E_{budget}/NUM_CORE, is above the energy threshold, E_{th}, we have sufficient energy budget to activate all cores and the main concern shifts to assigning workload in a heterogeneity-aware manner (Algorithm 2b). The key idea is to recursively assign workload and energy budget to the slowest unassigned core based on its frequency limit until energy budget per core is below a threshold for the remaining unassigned cores. The inputs of this heuristic are the energy budget for the upcoming schedule window, E_{budget}, number of cores on the chip, *NUM_CORE*, and, peak frequency supported by each cores, $f_{peak}(core\_id)$. Initially, all cores will be activated for the next schedule window (line 1) as the energy budget is capable of executing all cores above critical level, i.e., $E_{budget}/NUM\_CORE > E_{th} > E_{crt}$. In the main loop, we first find the slowest core and calculate U_{low} which is the maximum workload utilization that the core can support (lines 4-5). This utilization is accumulated into the objective workload utilization of the system, U_{obj}, and the corresponding energy consumption, E_{low}, is deduced from the energy budget (line 6). Then the heuristic updates

(line 7, 8) and compares (line 3) per-core average budget and threshold energy again for the rest of cores. After the main loop, the remaining energy budget will be evenly distributed to the unassigned cores and the final utilization is calculated (lines 9-11).

Algorithm 2b Heterogeneity-Aware Workload Estimation

Input: $f_{peak}(core\_id)$, peak frequency supported by each cores
Output: U_{obj}, objective workload utilization of system for next window

1. num_active ← NUM_CORE
2. num_unassigned ← NUM_CORE
3. **while** $E_{per\_core} > E_{th}$ **and** num_unassigned > 0 :
4. low_id ← core_id of unassigned core with lowest peak frequency
5. $U_{low} \leftarrow f_{peak}(cur\_id)/f_{max}$
6. $U_{obj} \leftarrow U_{obj} + U_{low}$, $E_{budget} \leftarrow E_{budget} - E_{low}$
7. num_unassigned ← num_unassigned – 1
8. update E_{per_core}, E_{th} for unassigned cores
9. calculate f_{per_core}, maximum frequencies supported by E_{per_core}
10. based on Inequation (4), calculate U_{per_core}, maximum utilization supported by f_{per_core}
11. $U_{obj} \leftarrow U_{obj} + U_{per\_core} \times$ num_unassigned

2.4.3.3. PROACTIVE RUN-TIME THERMAL MANAGEMENT

As discussed in Section 2.2.3, processors typically enforce throttling mechanisms to avoid thermal run-away. However, when a throttling decision is enforced, a processor has to drop all executing tasks and halt the system until temperature drops below a certain value. A system that encounters throttling often has frequent and dramatic changes in execution speed, which will hamper system energy efficiency. For this reason, in addition to the baseline enforced throttling mechanisms in processors, we propose to integrate a proactive reaction threshold, T_{pro}, at a slightly lower temperature than the baseline throttling threshold, T_{th}, to trigger measures that reduce system workload proactively with the goal of minimizing overheating and balancing workload over time. The details of our proposed scheme are summarized as follows:

- Cores with higher temperature than others are always given priority when there is a chance of core shutdown in Algorithm 2a;
- Cores with temperature above a proactive reaction threshold, T_{pro}, only run at critical frequency, so as to finish their limited workload with the highest energy efficiency and low power dissipation;
- When system peak temperature exceeds T_{pro}, the core which is operating at the peak temperature will be shut down to address the thermal hotspot in the system.

Thus, our run-time thermal management approach proactively manages workload to limit processor overheating so that occurrences of enforced throttling can be reduced for more stable execution, compared to traditionally used reactive throttling solutions.

2.4.4. TASK PENALTY AND CORE HETEROGENEITY AWARE TASK REJECTION AND ALLOCATION

This section describes how periodic tasks are allocated to cores or dropped, based on the awareness of individual task penalties and available core heterogeneity.

Algorithm 3 Heterogeneity Aware Task Rejection and Assignment

Input:
(i) U_{obj}, objective utilization from Algorithm 2 (a or b)
(ii) ψ, full task set assigned to system for scheduling
(iii) U_{ψ}, total utilization of task set ψ
Output: $f_{opt}(core\_id)$, optimal execution frequency of each core for upcoming schedule window

1. sort task set ψ in non-decreasing order of tasks' penalty densities
2. $\psi_{accepted} \leftarrow \psi$, Uaccepted $\leftarrow U_{\psi}$
3. **for** n = 1 to N :
4. **if** $U_{accepted} > U_{obj}$:
5. reject n^{th} task
6. $U_{accepted} \leftarrow U_{accepted} - U(n^{th}\ task)$
7. **else**

8. done with task rejection, **break**
9. sort accepted task set $\psi_{accepted}$ in non-increasing order of task utilization
10. **for** n = 1 to $N_{accepted}$:
11. filter out cores that has $U_n + U_{core} > U_{core\_max}$
12. assign n^{th} task to active core with the lowest utilization
13. get assigned task utilization for each active core, $U(_{core\_id})$
14. based on Inequation (4), calculate f_{opt}(core_id)
15. execute assigned tasks on each core with dual-speed heuristic

To add task priority control in SDA, we distinguish a task's importance by assigning a miss penalty to each task [35]. In this stage, our framework rejects tasks with lower penalty density (Section *2.2.2*) first, rather than simply drop tasks with longer execution time, to allocate the limited energy budget to more important tasks for miss penalty reduction. In particular, for the case when all tasks are assigned an identical miss penalty, this scheme reduces miss penalty equivalent to miss rate. We describe our task rejection heuristic below in Algorithm 3.

In lines 1-8, we sort all tasks in non-decreasing order of the tasks' penalty densities so that we can then reject tasks iteratively until the remaining tasks' total utilization is lower than the objective utilization given by Algorithm 2 (described earlier). The remaining tasks form the accepted task set and are assigned to all active cores using a simple but effective approach in lines 9-12. This approach not only enables priority control among tasks, but also distributes workload to each core as evenly as possible for balanced execution under a stable frequency. Also it ensures that the assigned workload will not exceed a core's maximum capability. After all accepted tasks are assigned, we obtain the actual utilization and optimal frequency of each core for the next schedule window.

2.4.5. DVFS SWITCHING-AWARE DUAL-SPEED METHOD

The previous section showed how we distribute accepted tasks among cores and deduce the theoretical optimum execution frequency for each core, which, however, is unlikely to be supported directly by processors with discrete frequency levels. To address this problem this section introduces a dual-speed method, which approximates the objective optimal frequency by switching between its two adjacent discrete frequencies [58]. For convenience, we denote the adjacent higher frequency as f_{high}, the lower one as f_{low}, and the objective optimal frequency as f_{obj}.

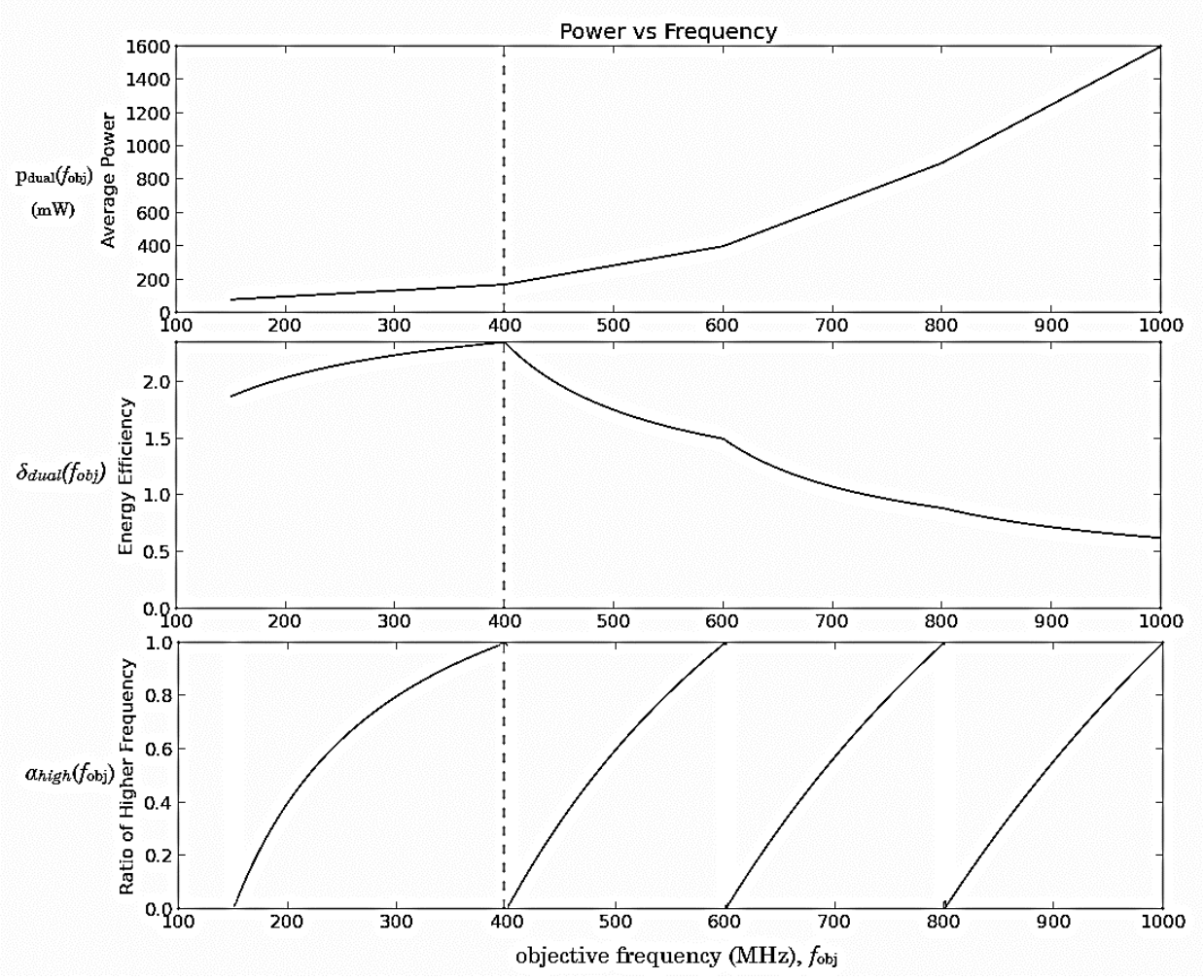

Figure 21 Energy Efficiency and Switching Proportion for the XScale Processor

Firstly, to guide the switching between two adjacent discrete frequencies, we need to calculate the proportion of cycles to execute with f_{high}, denoted as α_{high}. Assume that the total number of cycles to be executed is C. Emulating f_{obj} with a combination of f_{low} and f_{high} implies finishing C within the same amount of time, which is:

$$\frac{C}{f_{obj}} = \frac{\alpha_{high} C}{f_{high}} + \frac{(1-\alpha_{high})C}{f_{low}} \quad (5)$$

From this equation, we can deduce the proportion α_{high} for each objective frequency, f_{obj}, as

$$\alpha_{high}(f_{\text{obj}}) = \frac{1/f_{\text{obj}} - 1/f_{\text{low}}}{1/f_{\text{high}} - 1/f_{\text{low}}} \quad (6)$$

As f_{low} and f_{high} are determined by f_{obj}, there is a one-to-one correspondence between α_{high} and f_{obj}, and the values of $\alpha_{high}(f_{obj})$ can be calculated offline for a given task set. Based on the definition of energy efficiency in Section *2.2.3*, the theoretical efficiency of the dual-speed method $\delta_{dual}(f_{obj})$ can also be obtained offline as the objective frequency divide by the average power consumption. Based on the processor power-frequency model given in Table 1, the calculated $\alpha_{high}(f_{obj})$ and $\delta_{dual}(f_{obj})$ is shown in Figure 21. We can see that even with dual-speed switching, the efficiency will decrease when f_{obj} drop below critical frequency, $f_{crt} = 400\ MHz$. Thus for $f_{obj} < f_{crt}$, we should disable dual-speed switching and fix execution speed at f_{crt}.

However, it is non-trivial to get close to theoretical efficiency in a dual-speed method implementation, due to the following difficulties:

- Excessive DVFS switching results in massive switching overhead that considerably reduces energy efficiency [59];
- Executing at f_{low} for too long can cause task timing violations;
- Executing at f_{high} for too long results in timing slack before the arrival of new job instances of periodic tasks, which is wasted as idle cycles, thus reducing energy efficiency.

To address these issues, we implement a simple and intuitive dual-speed mechanism with inter-task switching, which aims to execute as many tasks as possible before switching to another frequency. This mechanism is described below:

- In order to prevent unnecessary DVFS switching, we denote number of cycles continuously executed at f_{high} to be C_{high} and set a threshold C_{thresh}. When $C_{high} = C_{thresh}$, whether switching to f_{low} or staying at f_{high} brings about the same energy consumption, i.e.,

$$p_{high} \times \frac{C_{thresh}/\alpha_{high}}{f_{high}} = p_{opt} \times \frac{C_{thresh}/\alpha_{high}}{f_{opt}} + 2 \times E_{switch} \quad (7)$$

  where p_{opt} is the average power consumption of executing with two frequencies to emulate f_{opt} and E_{switch} is DVFS switching overhead. The value of $C_{thresh}(f_{obj})$, can be easily calculated offline. From Equation (7), when $C_{high} < C_{thresh}$, the system forbids switching to f_{low}, as it leads to higher energy cost.
- To avoid task timing violation at f_{low}, our dual-speed method always sets execution speed to f_{high} initially. After finding a proper chance to switch to f_{low}, execution frequency jumps back to f_{high} as soon as a certain number of cycles have been executed at f_{low} such that $C_{high}/(C_{high}+C_{low}) = \alpha_{high}$, according to the specified proportion.
- To avoid undesirable idle cycles at f_{high}, our dual-speed method switches to f_{low} if number of unfinished job instances is not greater than 1, indicating possible shortage of workload. On the other hand, this step also helps to reduce number of switches as it halts switching to f_{low} when job instances in the queue are sufficient.

The steps above are summarized in Algorithm 4. Note that in line 3, frequency switching will not be triggered if $f_{obj} < f_{crt}$, as executing below critical frequency must be avoided.

Algorithm 4 Dual-Speed Method with Inter-Task Switching

Input:
(i) f_{obj}, objective optimal frequency
(ii) $\alpha_{high}(f_{obj})$ and $C_{thresh}(f_{obj})$, switching proportion and threshold profile for f_{obj} from 400 to 1000 MHz

```
1. f_cur ← f_high
2. while true :
3.     if f_obj > f_crt :
4.         if f_cur = f_high :
5.             C_high ← C_high + 1
6.             if jobpool.size ≤ 1 and C_high > C_thresh :
7.                 f_cur ← f_low
8.         if f_cur = f_low :
9.             C_low ← C_low + 1
10.            if C_low > C_high × (1- α)/ α :
11.                f_cur ← f_high
12.                C_low←0, C_high←0
13.    if at reschedule point :
14.        update f_obj based on Inequation (3)
15.        find adjacent frequencies such that f_low < f_opt < f_high
16.        fetch α_high(f_obj) and C_thresh(f_obj) from profile
```

To illustrate the advantage of our dual-speed method with inter-task switching, we compare it to three alternatives: (i) *single-speed method* which finds a higher than optimal frequency directly supported by the processor and does not switch frequency at all; (ii) *intra-task method* that aggressively toggles speed during executions for every task while considering switching overhead; and (iii) *ideal case* where intra-task method is applied, with switching overhead set to 0 to achieve theoretical best case efficiency. The comparison study calculates task miss rate and sets per core utilization to 100%. As can be seen in Figure 22, the *single-speed* scheme shows the worst result as it always executes at f_{high} when energy is available. The *intra-task* method works better by switching between two DVFS levels. However, its miss rate is still significantly higher than the ideal case due to excessive DVFS switching overhead. By presetting switching threshold and monitoring available workloads in the job pool, our *inter-task* switching scheme finds appropriate switching points and results in a miss rate that is close to the ideal case.

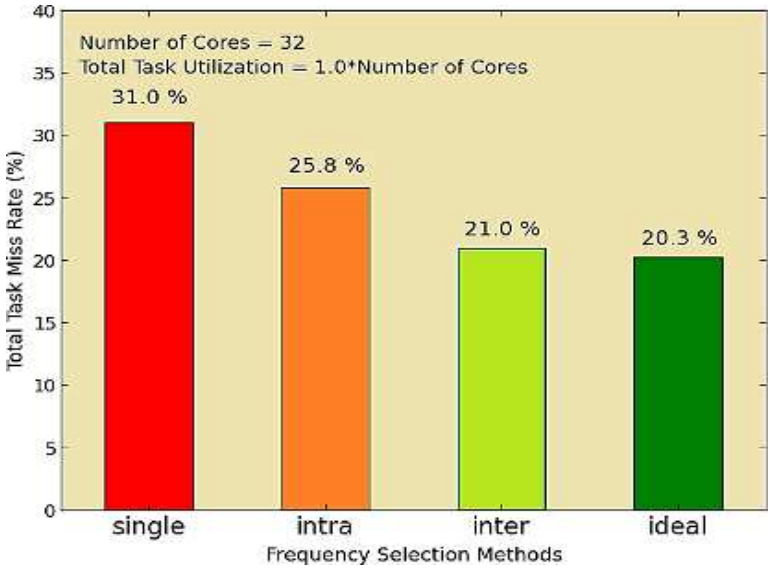

Figure 22 Comparison of Frequency Selection Methods

2.5. EXPERIMENTAL RESULTS

2.5.1. EXPERIMENT SETUP

We developed a simulator in C++ to implement and evaluate the effectiveness of our proposed SDA framework for run-time energy and workload management. The processor's power model was described in Table 1. Additionally, we ignore the timing delay to wake up cores from sleep state (~order of milliseconds) once per schedule window as it has a negligible impact on overall performance due to the much larger window size (~order of minutes). The energy harvesting profile is obtained from historical weather data from Golden, Colorado, USA, provided by the Measurement and Instrumentation Data Center (MIDC) at the National Renewable Energy Laboratory (NREL) [60]. Our harvesting-based embedded system only executes during daytime over a span of 750 minutes, from 6:00 AM to 6:30 PM and shuts down when solar radiation is unavailable.

In most experiments, we use synthetic task sets so than we can explore corner cases and have control over the spectrum of workload characteristics during testing. We generated 50 random

tasks for each test set configuration in our experiments. Each task set has an average task execution time randomly selected from 5 to 10 seconds. We vary the periods of all tasks in a task set based on the desired level of utilization required from the entire task set. We also ran experiments with the MiBench benchmark suite of embedded applications [61].

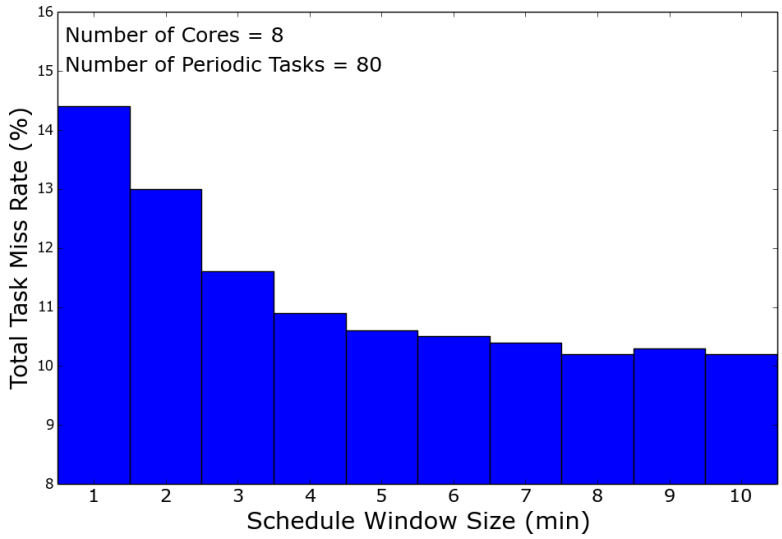

Figure 23 Miss Rates for Different Schedule Window Sizes

To determine the appropriate schedule window size for the SDA algorithm we ran several experiments with different window sizes. Figure 23 shows a set of results (miss rates) for our random task with 100% utilization on a core. We found that when window size increases from 1 to 5 minutes, there is a notable decrease in task miss rate. The reason behind this trend is that smaller schedule window sizes cause more task instances to span across the boundary of two different schedule windows, disrupting the newly assigned execution schedules of the next window. On the other hand, when we continue increasing window size beyond 5 minutes, the performance benefits become negligible while the demand on supercapacitor capacity to buffer energy harvested during a schedule window increases linearly. We found this trend to be consistent

for simulations with multiple cores as well. Thus we set 5 minutes as the size of schedule window in SDA for our experiments, to balance system performance and supercapacitor capacity requirements.

2.5.2. COMPARISON BETWEEN SDA AND PRIOR WORK

In this first set of experiments, we compare overall miss rates between *HA-DVFS* [41], *UTB* [43] and our proposed *SDA* framework for different number of homogeneous cores ranging from 1 to 32 with insufficient energy harvesting, for which the energy storage system is not stressed with surplus energy, so that the comparison in this subsection is focused on scheduling performance of SDA compared to prior work, without considering the advantages from our improved energy storage system design. We modeled the state-of-the-art ***ut****ilization-****b****ased algorithm* (**UTB**) in our environment. In addition, we also extended the ***e****nergy* ***h****arvesting-****a****ware* ***DVFS*** *technique* (**HA-DVFS**) for multi-core systems with balanced task partitioning across multiple cores, to enable another comparison point. With increasing number of cores, we scale harvesting power, number of tasks, and total task utilization linearly so as to keep a consistent and reasonable per core workload and energy budget.

First, we experiment on a workload with per core utilization set to 40%, which has moderate energy requirements such that the system can execute at critical frequency for highest efficiency when energy is sufficient. The results are shown in Figure 24. HA-DVFS can be seen to have a much higher miss rate as it does not make use of periodic task information and thus underestimates future workload. For the other two techniques, the advantage of SDA over UTB is small for the single-core setup because task utilization is not very high. However, with increasing number of cores, SDA's advantage expands considerably even though per-core workload and energy budget

stays the same. One reason for this trend is that UTB uses an isolated task dropping scheme on each core, which is based on energy availability prediction for one upcoming task, ignoring workload on other cores that compete for the same energy source. In contrast, SDA performs task rejection before assigning accepted tasks to different cores; thus the workload is adapted to a system-wide energy budget that has been predicted. Furthermore, SDA actually benefits from increasing number of cores as it exploits the flexibility to shut down some cores for higher efficiency.

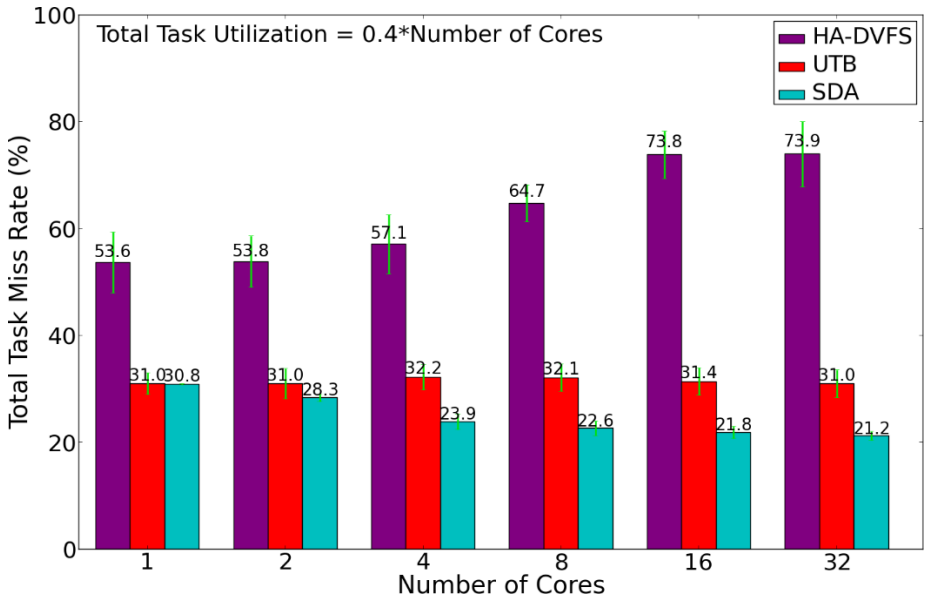

Figure 24 Miss Rate Comparison with Light Workload

We also compare UTB and SDA under a much heavier workload, with per core utilization set to 100%, results for which are shown in Figure 25. Not surprisingly, the heavier workload expands the performance gap between UTB and SDA. The reason for the increasing performance gap is that the higher workload implies more stringent timing and energy constraints, under which SDA's balanced run-time adjustment becomes more effective, as discussed in Section 2.3.1.2. As

a result, the most significant difference between these two techniques can be seen for the 32-core platform scenario, where SDA achieves approximately 70% miss rate reduction compared to UTB. Additionally, the results of SDA have less variation on multiple task sets compared to UTB, which indicates that task set randomness has less impact on SDA as its dynamic adjustment is based on the scope of the entire task set, and not just individual tasks.

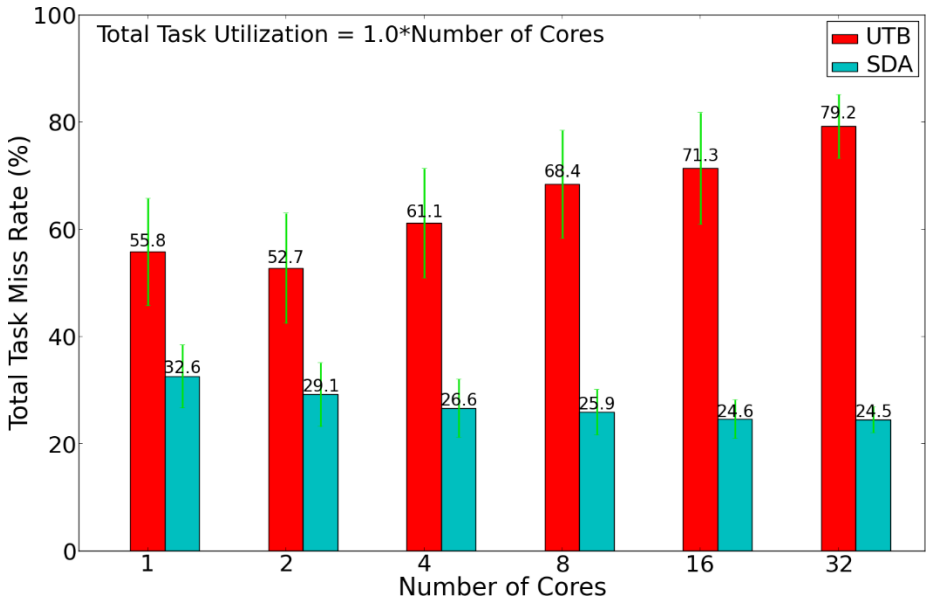

Figure 25 Miss Rate Comparison with Heavy Workload

Additionally, we compare performance of HA-DVFS, UTB, and SDA by scheduling a 4-core system running a set of applications (*jpeg, qsort, dijkstra, patricia, blowfish, susan, tiff*) extracted from MiBench, a benchmark suite of embedded applications [61], with total utilization of 160%, where every application executes recursively based on its assigned period with each application execution request considered as an independent task instance. The result in Table 2 shows higher miss rates compared to average values with a similar 4-core configuration in Figure 24. The reason lies in the application set's higher average length of task instances compared to

most of the randomly generated tasks, making it harder to balance workload among cores and leading to higher overhead when a task instance's life cycle spans across two schedule windows, as discussed earlier in Section 2.5.1.

Table 2 Miss Rate Comparison on MiBench

Scheduling Technique	HA-DVFS	UTB	SDA
Total Miss Rate	58.5%	33.8%	25.8%

2.5.3. ANALYSIS OF SDA WITH HYBRID ENERGY STORAGE

This set of experiments explores the performance benefits of our proposed SDA algorithm together with the proposed hybrid energy storage system. Compared to the previous section that focuses on scenarios with insufficient energy harvesting, experiments in this section assume per-core nominal harvested energy scaled up by a factor of two, so that the system receives more than sufficient energy occasionally and surplus energy needs to be stored to support execution when harvesting power drops. We use the approach from [47] to model rate capacity effect of batteries by scaling efficiency based on discharge current. Also we implemented four variants of SDA, namely (i) *BA-SDA*: SDA for battery-only system with doubled battery capacity; (ii) *CA-SDA*: SDA for supercapacitor-only system with doubled supercapacitor capacity; (iii) *MISS-SDA*: SDA with hybrid storage and focus on miss rate reduction; (iv) *HY-SDA*: SDA with hybrid storage and focus on miss penalty reduction. These variants of our approach were compared against UTB. Additionally, UTB, BA-SDA and CA-SDA rely on a *moving average* algorithm for energy harvesting prediction [41] as they do not have dual-supercapacitor design to buffer harvested energy for upcoming schedule windows. All task sets have utilization of 100% for this set of experiments. Additionally, tasks are assigned a miss penalty ranging from 1 to 100 with a uniform distribution. We compared average overall miss penalty and miss rate for these various techniques,

with increasing multi-core platform complexity (from 1 to 16 cores). Capacities of batteries and supercapacitors, and nominal harvested energy for the entire system scale linearly with number of cores in the processors.

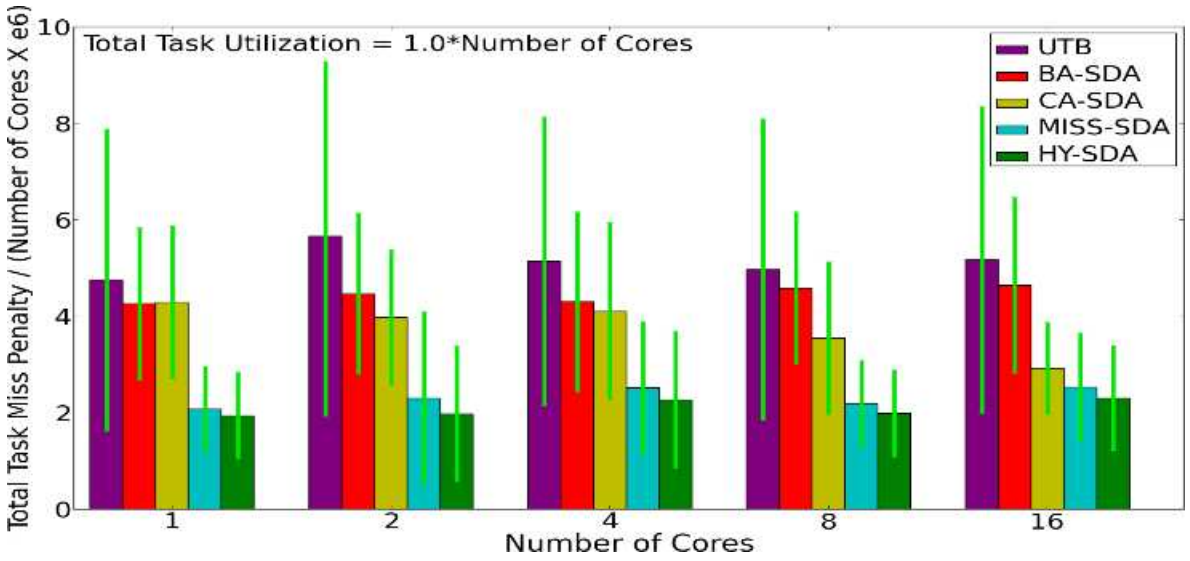

Figure 26 Overall Miss Penalty Comparison

The results for this experiment are shown in Figure 26 and Figure 27. Similar to the conclusion in the previous section, both BA-SDA and CA-SDA have lower miss penalty (Figure 26) and miss rate (Figure 27) than UTB, and their advantage expands with increasing number of cores. However, their advantage over UTB is less significant compared to what we see in the previous section (Figure 25). The reason is two-fold: firstly, with doubling of per-core nominal harvested energy in this set of experiments, the stringent energy constraint, which highlights the difference between UTB and SDA, is relaxed significantly; secondly, with more than sufficient energy harvesting, management of surplus energy becomes the new bottleneck that partially diminishes the advantage of SDA. Respectively, the performances of BA-SDA and CA-SDA mainly suffer from lower charging/discharging efficiency of the battery and limited capacity of the

supercapacitor. Also, CA-SDA has advantage over BA-SDA with increasing number of cores in the system as systems with more cores have higher demands on discharging current; and supercapacitors, with their high power density, serve high current load more efficiently than batteries [47].

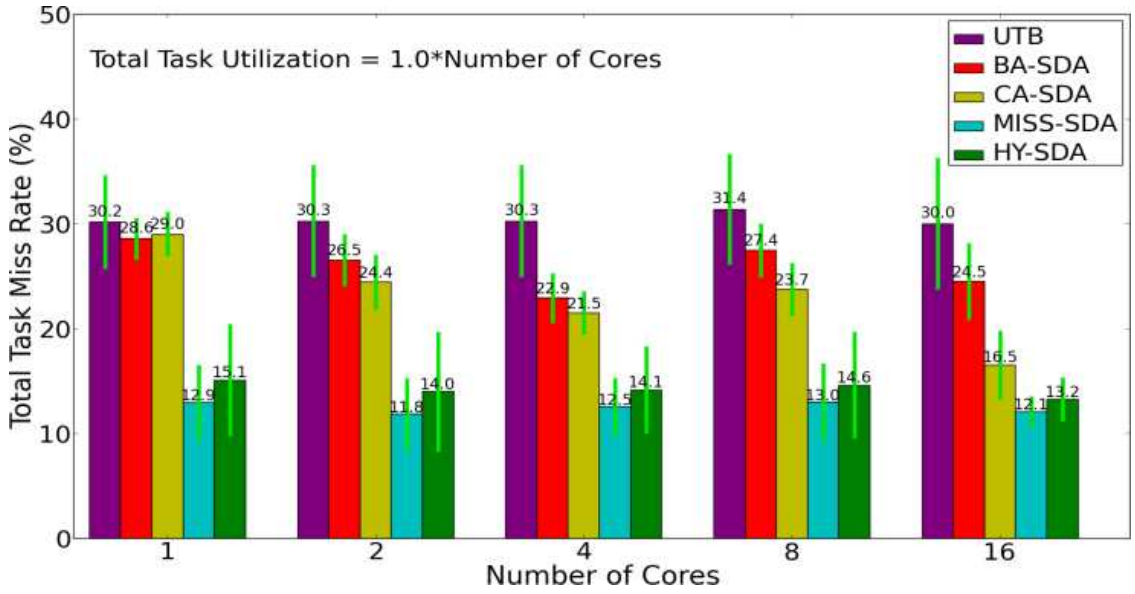

Figure 27 Overall Miss Rate Comparison

For MISS-SDA and HY-SDA, integration with our hybrid storage system managed by the SDA-based policy results in a much lower miss penalty and miss rate compared to UTB, BA-SDA and CA-SDA. Even though this significant performance improvement is due to the introduction of a hybrid storage system in MISS-SDA and HY-SDA, the efficient management of such a hybrid storage system is made possible by the semi-dynamic scheme of SDA, which offers flexibility at reschedule points to select the appropriate energy source and deduce optimal energy budgets at the start of each schedule window. Additionally, the difference between MISS-SDA and HY-SDA is in how they prioritize minimization of miss rate and miss penalty. The HY-SDA scheme leads to

the lowest task miss penalty, with up to 65% reduction compared to UTB, while MISS-SDA results in a slightly higher miss penalty than HY-SDA as it focuses on miss rate reduction. As expected, the miss rate for MISS-SDA is the lowest and has less variation compared to HY-SDA (Figure 27).

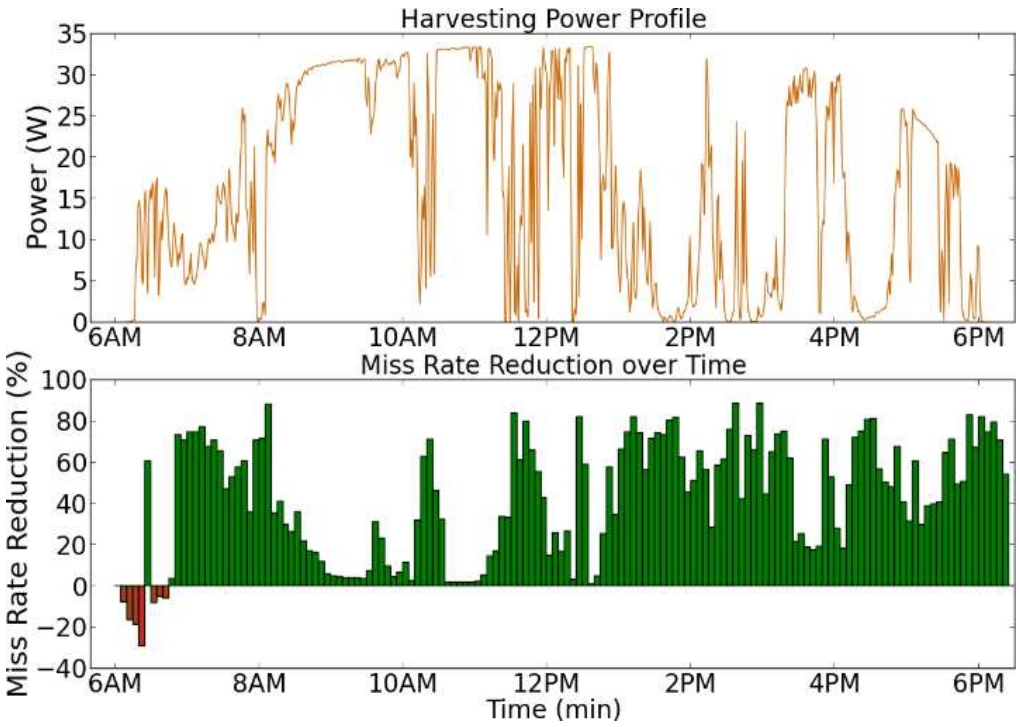

Figure 28 Miss Rate Reduction for HY-SDA Compared to UTB

We also further explored the results for miss rate reduction obtained with HY-SDA compared to UTB for each schedule window. These results on a 16-core system are shown in Figure 28. We can see that the HY-SDA results in a higher miss rate than UTB initially, because it shuts down all cores until the supercapacitor is sufficiently charged to avoid executing with inefficient frequencies under the critical level. Subsequently, higher miss rate reduction for HY-SDA is achieved when harvesting power is low or changes dramatically, reflecting the advantage that HY-SDA has over UTB to cope with stringent energy budgets and its ability to filter out solar

harvesting variations. Moreover, HY-SDA results in a more significant miss rate reduction after 12 PM. The reason for this is that HY-SDA's high energy efficiency leads to more energy savings in the battery, which enables more tasks to be executed and meet their deadlines.

2.5.4. ANALYSIS OF CORE HETEROGENEITY-AWARE MANAGEMENT

Next, we study the performance impact of core heterogeneity caused by within-die process variations. Based on results from [48] we set core frequency variation within a die as 33% and static power variation as 50% with normal distribution. When a frequency level cannot be reached by a core, the system always conservatively sets frequency to the next lower discrete frequency level. We tested three different setups, namely (i) *Variation-Unaware*: SDA with core heterogeneity-aware techniques disabled. Also we assume that the system will force cores to execute at frequencies no higher than their maximum capability to ensure stability; (ii) *Variation-Aware*: SDA with our core heterogeneity-aware techniques; and (iii) *Homogeneous*: an ideal case assuming no heterogeneity.

The results for this study are shown in Figure 29. It can be seen that without awareness of within-die process variation, the system suffers from a very high miss rate, as the assigned workload exceeds the actual execution capabilities of slower cores on the die, resulting in a faulty schedule which wastes energy and CPU time on tasks that cannot be finished in time. In comparison, with core heterogeneity-aware workload distribution, the system avoids faulty scheduling and alleviates the impact of process variation. However, as expected, the results are inferior to that obtained for the ideal case which has homogeneous cores unaffected by process variation, because of the degradation in maximum throughput supported and non-uniform workload distribution forced by inter-core heterogeneity.

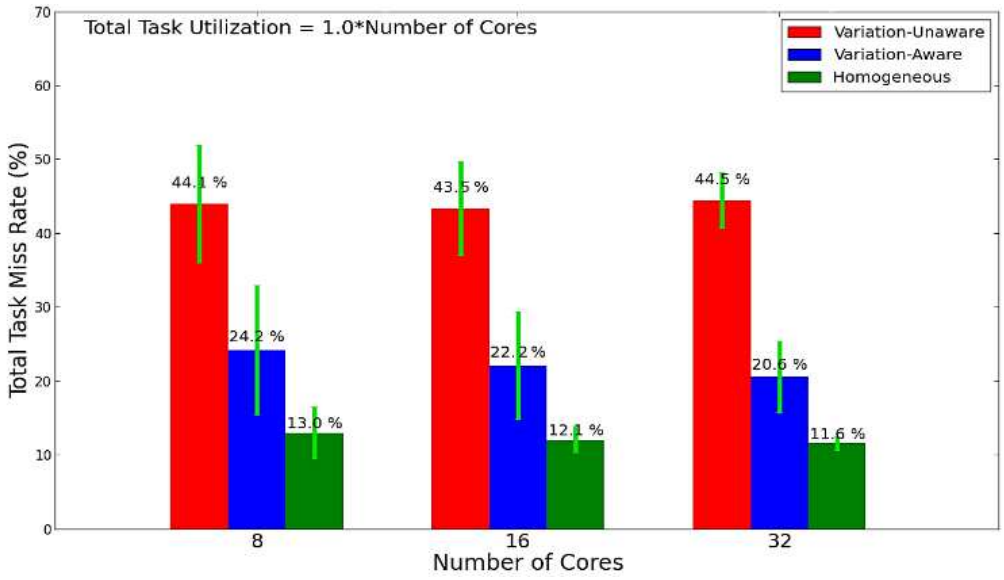

Figure 29 Overall Miss Rate Comparison with Core Heterogeneity

2.5.5. ANALYSIS OF RUN-TIME THERMAL MANAGEMENT

In this section, we explore the impact of run-time thermal management in an energy harvesting environment. While prior work [36] has considered the effect of temperature on maximum power point tracking in energy harvesting systems, it has not considered the impact of thermal-induced overheating on task execution throttling and slowdown in energy harvesting embedded systems. To simulate a scenario with high overheating risk (as discussed in Section 2.3.4), we evaluate our approach for a very heavy workload with per core utilization set to 100%. Our environmental profile considers high solar intensity and ambient temperatures from 9AM to 3PM. For thermal analysis, we integrated our simulator with HotSpot, a thermal modeling and analysis tool [62]. We set package parameters of the Hotspot tool to model a 16-core processor with no power-hungry cooling system (only a heat spreader and heat sink is assumed). We assume die area of our chip to be a 16mm × 16mm, with cores placed in a mesh topology. Then we set processor package size as 60mm × 60mm, which is also the size of heat spreader and heat sink. In

our tests, we compare the performance of three schemes: (i) *Non-Throttling*: A basic SDA scheme with no run-time thermal management scheme. This is representative of current state of the art scheduling techniques for energy harvesting systems that ignore thermal issues; (ii) *Throttling*: We again consider our SDA scheme without thermal-awareness, but here system hardware can measure temperature and reactively enforce throttling when temperature exceeds the throttling threshold; (iii) *Proactive*: This is our SDA approach that integrates proactive core slowdown and task redistribution from Section 2.4.3.3 to proactively address hotspots in the systems.

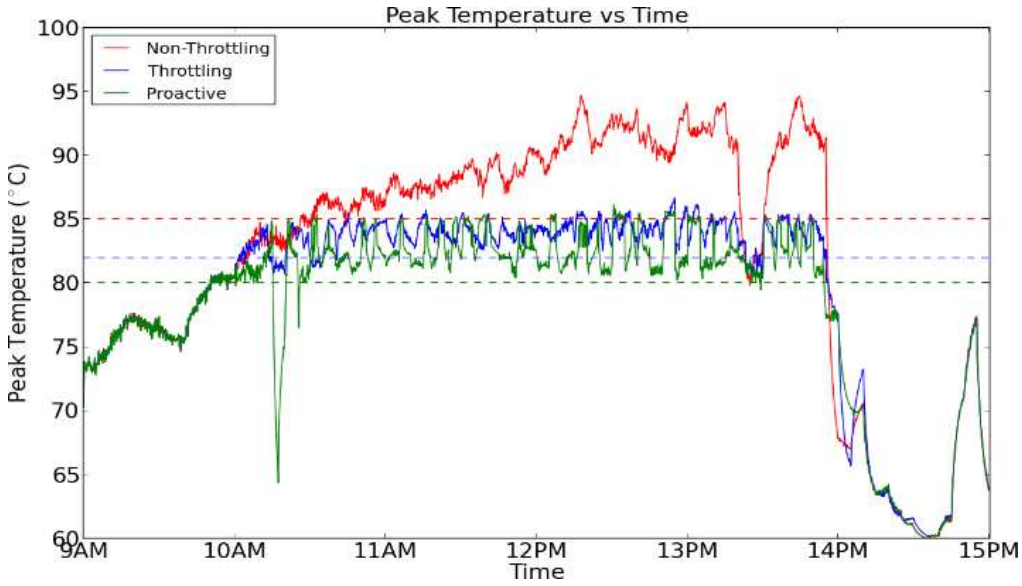

Figure 30 Peak Temperature of Various Thermal Management Techniques

The results for the three schemes are shown in Figure 30. It can be seen that the *Non-Throttling* scheme suffers from high peak temperatures for extended periods of time. Such high temperatures will significantly impact the system stability and reliability. In contrast, the reactive *Throttling* scheme is able to control temperature to stay below the throttling threshold for a majority of the time. In Figure 30 the red dashed line indicates the throttling threshold at 85°C and

the green dashed line shows the threshold at 80°C at which throttling terminates. Note that peak temperature seldom drops to 80°C in simulation. This is due to the fact that other un-throttled cores take over the role of thermal hotspots in the system from the throttled cores. Our TA-SDA *Proactive* scheme proactively performs core slowdown when temperature exceeds a proactive reaction threshold (set to 82°C, and shown with the blue dashed line shown in Figure 30). This scheme helps to increase energy efficiency by avoiding unbalanced frequencies created by thermal throttling. Table 3 shows how our proactive approach not only reduces peak temperature, but also reduces the number of throttling instances, which allows more efficient scheduling management, culminating in an overall improved task miss rate. The results highlight the benefits of proactive run-time thermal management.

Table 3 Comparison between Throttling and Proactive Schemes

Thermal management scheme	average peak temperature	number of throttlings	overall task miss rate
Throttling	79.60°C	94	35.92%
Proactive	78.53°C	74	35.33%

2.5.6. ANALYSIS OF SCHEDULING OVERHEAD

To compare scheduling overhead between UTB, HA-DVFS and our proposed SDA framework, we executed the scheduling procedures of these schemes on the gem5 simulator [63] with a single thread at 1GHz to observe average execution time overhead averaged over all task instances when managing a 16-core system running 160 periodic tasks with a scheduling granularity of 1ms. The results of this study are shown in Figure 31, in which we can see that SDA+DUAL has less scheduling overhead (with respect to performance and energy) compared to UTB while providing more features such as hybrid storage-based energy budgeting, thermal

management, and dual-speed switching. The main reason for the lower overhead with SDA+DUAL is that it is designed to reuse intermediate information computed at the beginning of each schedule window, avoiding frequent on-the-fly scheduling procedure invocations during task execution, with dual-speed method as the exception. The HA-DVFS also has much lower overhead than UTB's, as well as SDA+DUAL, as most of its features are triggered only when a new task instance is available. We were also interested in quantifying the overhead of our dual-speed method, which is perhaps the most complex run-time component in our scheduling framework. We therefore also present the scheduling overhead for SDA-DUAL, which disabled the dual-speed feature. It can be seen that without the dual-speed method, our scheduler execution time and energy overheads become lower than overheads for UTB and HA-DVFS.

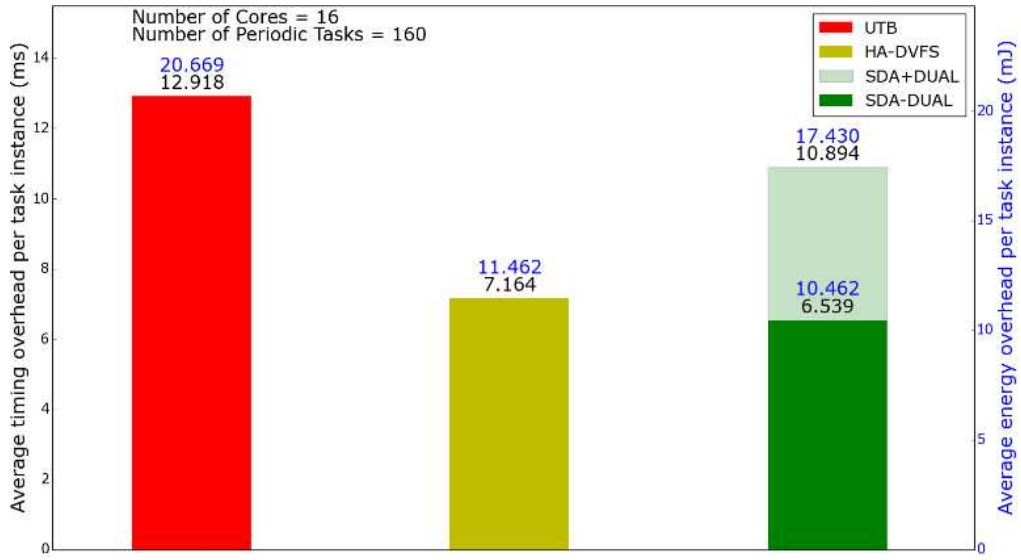

Figure 31 Comparison of Scheduling Overhead

2.6. CHAPTER SUMMARY

In this chapter, we proposed a novel framework for run-time energy and workload management based on a ***s**emi-**d**ynamic **a**lgorithm* (**SDA**), for real-time multi-core embedded

systems with solar energy harvesting. Compared to the best known previous work, our approach is promising for energy-harvesting based multicore embedded systems: 1) up to 70 % miss rate reduction and 65% miss penalty reduction for SDA compared to the best know prior work, UTB; 2) Analysis with system overheating considerations establishes the need for combing proactive thermal management during scheduling, as done in our SDA approach, to reduce both miss rate and average peak temperature among cores; 3) SDA with core-heterogeneity awareness presents miss rate reduction of 49% compared to SDA without such awareness when process variation effects on maximum frequency and power are considered. Overall, SDA provides a holistic solution with many novel components, integrating a new hybrid energy storage system, task drop penalty awareness, run-time thermal management, and core heterogeneity awareness. Moreover, the design methodology of a semi-dynamic framework for resource management is the core idea of our research, which will be applied to address further issues in the rest of this dissertation.

3. TEMPLATE-BASED SCHEDULING ALGORITHM FOR TASK GRAPHS

The problem of scheduling weighted ***d****irected* ***a****cyclic* ***g****raphs* (**DAG**s) on a set of homogeneous cores under optimization goals and constraints is known to be NP-complete [64]. In this chapter, we address the even more difficult problem of scheduling on systems that rely entirely on limited and fluctuating solar energy harvesting. The limited energy supply prevents the deployment of complex scheduling algorithms at run-time. Moreover, execution of applications that will not have enough energy or computation resources to complete due to shortages in harvested solar energy can lead to significant wasted energy with no beneficial outcome. Fortunately our concept of semi-dynamic scheduling proposed in last chapter can be applied to address these challenges. Thus in this chapter, we propose a ***hy****brid* ***w****orkload* ***m****anagement framework* (**HyWM**) that combines template-based hybrid scheduling with our energy budget window-shifting strategy derived from semi-dynamic framework in last chapter to decouple run-time application execution from the complexity of DAG scheduling in the presence of fluctuations in energy harvesting. Basically, our framework generates schedule templates at design-time with an emphasis on energy efficiency and uses lightweight online management schemes to react to run-time system dynamics. Moreover, our framework also considers varying aspect of issues like stochastic task execution time, random transient faults, and progressive aging effects.

3.1. BACKGROUND AND CONTRIBUTION

Due to the variable nature of solar radiation intensity, the most suitable role of embedded systems with limited-scale solar energy harvesting as the only energy source is to host non-critical applications that allow for imperfect operation. Thus it may not be desirable to consider such

systems for real-time applications with hard deadlines, such as life-support mechanisms and powertrain controllers, for which any deadline miss is a critical system failure that may have catastrophic consequences. Instead, it is more practical to deploy such systems without energy guarantees for best-effort execution of soft or firm real-time applications where a deadline miss is not considered a failure of the entire system but a degradation of performance.

Consider an example of such a best-effort embedded system powered by energy harvesting that is deployed for continuous structural integrity sensing at a remote location on a bridge. For each operation interval, a usable raw data point can be collected from sensor modules by executing certain real-time control tasks such as data accessing, data post-processing and data-transmission. In the event of an energy shortage, the system stays operational with certain data collection intervals ignored such that overall sensing quality is sacrificed in favor of ensuring system continuity.

To achieve best-effort operation with limited resources, the deployment of an intelligent run-time resource management strategy is not only beneficial but also essential. Such a strategy must possess low overhead, so as to not stress the limited energy resources at run-time. As shown in section 1.3, several prior efforts have explored workload scheduling for such real-time embedded systems with energy harvesting. However, all of these efforts are aimed at independent task execution models, and cannot be easily extended to more complex application sets that possess inter-node data dependencies, such as workloads represented by ***d**irect **a**cyclic **g**raphs* (**DAG**s).

Due to aggressive scaling in CMOS technology, emerging multicore processors are also facing ever-increasing likelihoods of transient faults (i.e., soft errors) and permanent faults (i.e., hard errors). Co-optimization of reliability and energy-efficiency have thus become a critical design concern in recent work on task scheduling [65] [66] [67] [68] [69] [70] [71] [72] [73].

However none of these efforts focus on energy harvesting based systems. For low-power embedded systems that scale down voltage and frequency for energy savings, the rate of transient fault occurrences, caused by a variety of factors, e.g., high-energy cosmic neutron or alpha particle strikes, and capacitive and inductive crosstalk [74], is more severe as lower supply voltage leads to drastically increased susceptibility to transient faults [75]. Additionally, embedded systems with energy harvesting must also consider the impact of hard errors because a major incentive of deploying such systems is long-term system autonomy, which requires an extended system lifetime. For these reasons, we believe it is necessary to study workload management schemes that consider both transient errors and aging effects to enhance system reliability and lifetime for low-power systems with energy harvesting.

In this chapter, we propose a low-overhead soft and hard reliability-aware ***hy**brid **w**orkload **m**anagement framework* (**HyWM**) to address the problem of allocating and scheduling multiple applications on multicore embedded systems powered by energy harvesting, and in the presence of transient and aging faults. Compared to prior work, the novelty of our work can be summarized as follows:

- A hybrid application mapping and scheduling framework is proposed that integrates a rigorous design-time analysis methodology with lightweight run-time components for low-overhead energy management in solar energy harvesting based multicore embedded systems for the first time.
- We propose two different approaches to solve the DAG scheduling problem at design-time, generating schedule templates composed of energy-efficient application execution schedules for various energy budgets that can be encountered at run-time.

- Our allocation scheme for workload partitions considers different wear-out profiles of cores and adjusts workload distribution accordingly to maximize lifetime of the entire system.
- Our run-time scheduler utilizes a novel lightweight run-time heuristic that co-manages run-time *slack reclamation* and *soft/hard error handling* in a multicore computing environment without diminishing the benefits of schedule templates generated at design-time.

3.2. RELATED WORK

Many prior research projects have focused on the problem of run-time management and scheduling for embedded systems with energy harvesting, as we discussed in section 1.3. However, none of them take inter-task dependency into consideration.

Several other efforts have explored mapping and scheduling for task-graph based workloads. Luo et al. proposed a hybrid technique to find a static schedule for known periodic task graphs at design-time with the flexibility to accommodate aperiodic tasks dynamically at run-time [76]. Sakellariou et al. proposed hybrid heuristics for DAG scheduling on heterogeneous processor platforms [77]. Coskun et al. proposed a hybrid scheduling framework that adjusts the task execution schedule dynamically to reduce thermal hotspots and gradients for MPSoCs [54]. *However, all of these prior efforts cannot maintain performance when applied to energy harvesting systems that possess a fluctuating energy supply at run-time. Some of these efforts also do not focus on energy as a design constraint*. Our work specifically targets the problem of energy-aware scheduling of multiple co-executing task graphs in energy harvesting based multicore platforms.

A few efforts have addressed the problem of reliability and energy co-optimization during scheduling. For soft-error reliability, Zhu et al. proposed an approach to insert a recovery task during slack time obtained from executing multiple tasks [65]. To address the conservative nature of individual-recovery based approaches, Zhao et al. proposed a shared recovery technique that shares a small number of recovery nodes among all nodes executing tasks, to meet a system wide reliability target [66]. This SHR technique also has been applied to address reliability during scheduling of DAG-based workloads [67]. For hard failures, prior work has studied aging effects that lead to permanent system failure, such as ***e****lectro* ***m****igration* (**EM**), ***n****egative* ***b****ias* ***t****emperature* ***i****nstability* (**NBTI**), and ***t****ime* ***d****ependent* ***d****ielectric* ***b****reakdown* (**TDDB**). Coskun et al. proposed a framework to evaluate architecture-level effects of task scheduling and power management on lifetime of multi-processors [71]. An analytical model to estimate lifetime reliability of multi-processors with a periodic workload was proposed in [72]. Basoglu et al. quantitatively evaluated the long-term impact of NBTI-aware task-to-core mapping for multi-processors [73]. *None of these works target systems with unstable supply from energy harvesting.* In our work, unlike prior efforts on integrating reliability during scheduling, we do not aim to satisfy a target reliability. Instead, our focus is on alleviating the impact of soft and hard errors to finish as many applications correctly as possible and extending expected lifetime for a system with a time varying and stringent energy budget from energy harvesting.

3.3. PROBLEM FORMULATION

This section focuses on hybrid allocation and scheduling of multiple task-graph applications with real-time deadlines on multicore embedded systems with solar energy harvesting, in the presence of soft and hard errors, as shown in Figure 32. Although key components and assumptions

of system platform, like energy harvesting system and processor model, are similar to those in Chapter 2, problem formulated in this chapter is more complex with emphasis on several new design considerations such as task dependencies, soft errors, and system lifetime.

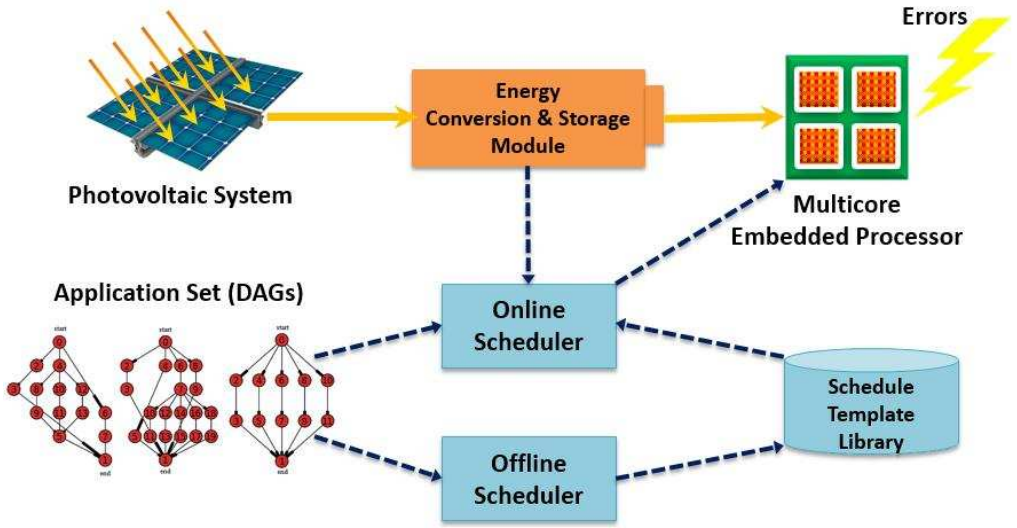

Figure 32 DAG Scheduling on Multicore Embedded System Platform with Solar Energy

3.3.1. PERIODIC REAL-TIME WORKLOAD WITH TASK GRAPHS

The main change in problem formulation of this chapter is the introduction of workload model with dependencies. We consider multicore systems hosting multiple recursive real-time applications modeled as periodic task graphs, $\psi: \{G_1, \ldots, G_{Ng}\}$, such as the examples shown in Figure 33. Each of the N_g applications is represented by a weighted ***d**irected **a**cyclic **g**raph* (**DAG**), denoted as $G_i: (t_i, e_i, T_i, D_{i,j})$, $i \in \{1, \ldots, N_g\}$, which contains a set of task nodes, $t_i: \{\tau_1, \ldots, \tau_j\}$ with worst-case execution cycles, $WCEC_i$, (number of CPU clock cycles needed to finish a task *i* in the worst case); and a set of directed edges, $e_i: \{\varepsilon_1, \ldots, \varepsilon_j\}$, used to represent inter-task dependences with communication (inter-core data transfer) delay from source to destination nodes represented as $COMM_{src,dst}$. A task node can have multiple dependences to/from other nodes, forking/rejoining

execution paths in the task graph. We assume that every task graph's execution paths rejoin at its last task node, which accumulates results and concludes execution.

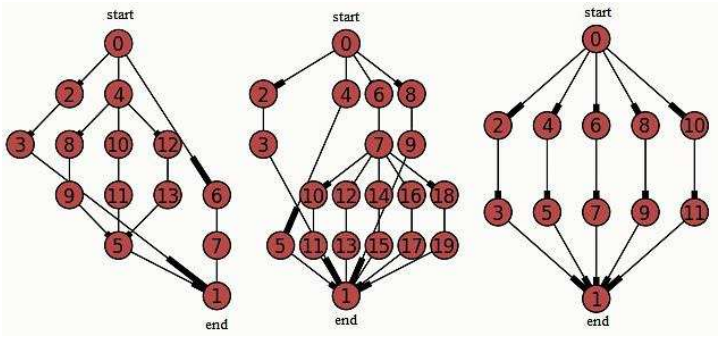

Figure 33 Example of Applications Modeled as DAGs

Every periodic task graph has a unique period, T_i and nodes in the graph are assigned relative deadlines, $D_{i,j}$. At the beginning of each period, a new instance of a task graph will be dispatched to the system for execution. A task node's relative deadline, $D_{i,j}$, is the time interval between the task graph instance's arrival time and node firm deadline (see Section 1.2.2). A task graph instance misses its deadline if it cannot finish executing any nodes before their deadlines. In this work, we assume that the deadline of each task graph's last node $D_{i,-1}$ equals T_i, i.e., for a periodic task graph, its instance has to finish execution or be dropped before the arrival of the next instance.

In this chapter, we assume the actual time (clock cycles) required to execute a task node may vary at run-time due to variations in memory system behavior and randomness in application procedures. We therefore use probability distributions to model variations in task node execution time [78] and assume that clock cycles consumed by a task node never exceed its WCEC.

Similarly, to assess the computation intensity of an application relative to a processor's full capability, the computation utilization of a periodic task graph (U_{comp}) is defined as the sum of execution times of all its task nodes for the highest processor clock frequency divided by its period:

$$U_{comp\,i} = \frac{\sum_j WCEC_{i,j}/f_{max}}{T_i}, i \in \{1, \ldots, N_g\} \quad (8)$$

Also we define communication utilization of a periodic task graph (U_{comm}) as the sum of the communication times for all of its edges divided by the task graph's period:

$$U_{comm\,i} = \frac{\sum_k \mathrm{COMM}^k{}_i}{T_i}, i \in \{1, \ldots, N_g\} \quad (9)$$

The computation/communication utilization of the entire multi-application workload is simply the accumulation of utilizations for all task graphs, which provides an indication of the overall workload intensity of a DAG application set.

3.3.2. SOFT ERROR MODEL

A system is said to be real-time if the total correctness of an operation depends not only upon its logical correctness, but also upon the time in which it is performed [14]. In most part of this dissertation, we focused on timing constraints of task instances by counting miss rate in regard to firm deadlines. In this chapter, we also look into logical correctness when counting miss rate and assume that task nodes can produce incorrect output due to soft errors occurring during execution and such incorrect outputs can be detected by verification logic executed at the end of regular task execution. To recover from a soft error, the task node with a faulty output must be re-executed, otherwise the output of the entire task graph will become invalid, which is counted as a task graph miss. We apply the exponential model proposed in [75] to simulate soft error rates, as shown in Equation (10):

$$\lambda(f) = \lambda_0 10^{\frac{d(1-f)}{1-f_{min}}} \quad (10)$$

where λ_0 is the average error rate corresponding to the maximum frequency, d is a constant that indicates the sensitivity of error rate to voltage scaling, f_{min} is the normalized minimum core

frequency, and *f* is the normalized core frequency. It can be observed that lower power execution at lower supply voltage (and thus frequency) to save energy can result in an exponential increase in soft error rate [67].

3.3.3. HARD ERROR MODEL

In addition to soft errors, we also consider aging effects that eventually lead to hard errors (permanent failure) in electronic systems. We adapt an analytical method to capture system-level lifetime reliability in embedded systems with multiple cores. In the rest of this section we first introduce how aging effects are modeled in our work and then describe a method to calculate reliability of a multicore embedded system according to a specified level of failure tolerance.

Many prior research efforts model hard reliability characteristics of systems using ***m**ean-**t**ime-**t**o-**f**ailure* (**MTTF**) prediction [71]. However, for aging effects, it is more intuitive to model the changing of reliability over time due to progressive wear-out [79]. In our work, we estimate instantaneous hard reliability of a core, which reflects the possibility of core's avoidance of permanent failure within a time epoch. We utilize a Weibull distribution, which is one of the most widely used and versatile lifetime distributions in reliability engineering, to characterize per-core wear-out over time [80]. The instantaneous hard reliability of a single core at time *t*, *R(t)*, can be expressed as:

$$R(t) = e^{-(\frac{t}{\alpha})^{\beta}} \tag{11}$$

where α and β represent the scale parameter and slope parameter in the Weibull distribution, respectively. While β is a constant that reflects architectural characteristics of core, α is highly dependent on the operating history of the core. Thus in our reliability model we need to deduce

the relationship between the scale parameter α and operating history of the processing core. Firstly, by the definition of a Weibull distribution, MTTF of a core can be calculated as

$$MTTF = \alpha \times \Gamma(1 + \frac{1}{\beta}) \tag{12}$$

Then we can represent the scale parameter α as:

$$\alpha = \frac{MTTF}{\Gamma(1 + \frac{1}{\beta})} \tag{13}$$

This representation makes it possible to calculate the scale parameter for a core's instantaneous hard reliability model by adapting various MTTF-based hard error models, such as ***e**lectro **m**igration* (**EM**), ***t**ime **d**ependent **d**ielectric **b**reakdown* (**TDDB**), and ***n**egative **b**ias **t**emperature **i**nstability* (**NBTI**) [71] [72] [73]. In this work we focus on EM-based aging, the MTTF model for which can be expressed as:

$$MTTF = A_0 (J - J_{crit})^{-n} e^{\frac{E_a}{kT}} \tag{14}$$

where A_0 is a material-related constant, $J = V_{dd} \times f \times p_i$ [71], and J_{crit} is the critical current density. Then we have

$$\alpha = \frac{A_0 (V_{dd} \times f \times p_i - J_{crit})^{-n} e^{\frac{E_a}{kT}}}{\Gamma(1 + \frac{1}{\beta})} \tag{15}$$

where V_{dd}, f, and T can be controlled by our workload management framework.

To approximate aging effects over time, we use a fixed time epoch of length Δt as the basic time unit, for which averaged core frequency, supply voltage and temperature are applied to the above model for hard reliability calculation. According to [72], the reliability of a core at time epoch t_w, as the result of accumulated wear-out effects in previous time epochs from t_0 to t_{w-1}, can be approximately calculated as:

$$R(t_w) = e^{-(\sum_{i=0}^{w-1} \frac{\Delta t}{\alpha(t_i)})^{\beta}} \quad (16)$$

Also, MTTF of a core can then be represented as

$$MTTF = \sum_{i=0}^{\infty} \Delta t \times R(t_i) \quad (17)$$

For multicore systems, it is essential to consider not just reliability of each core individually, but rather the impact of aging on the entire system. We define a system-level failure threshold (*h*) as the maximum number of core failures allowed before the entire system is considered to have failed. For example, if *h*=0, the system fails as soon as one core fails, i.e., all cores must maintain their functionality to keep system up. The hard reliability of a system for this case is:

$$R_{sys}(t_w)_{h=0} = \prod_{k=1}^{N} R_k(t_w) \quad (18)$$

where *N* is number of cores in a system. For general cases, where failure threshold *h* has a non-zero value, the hard reliability of system can be calculated as shown below:

$$R_{sys}(t_w)_h = R_{sys}(t_w)_{h-1} + \sum_{\substack{F \subset \{1,\dots,n\} \\ |F|=h}} \left(\prod_{k \in F} (1 - R_k(t_w)) \times \prod_{k \in \{1,\dots,n\} \setminus F} R_k(t_w) \right) \quad (19)$$
$$h \in [1, N-1]$$

In the above equation, hard reliability of the system is calculated recursively, such that reliability of the system with failure threshold *h* equals reliability of the system with threshold of *h-1* plus the probability of the system to have exactly *h* cores failed. Different cores usually have different hard reliabilities due to uneven workload distribution among them, therefore when calculating probability of a certain number of cores failed, it is essential to enumerate all cases in combination and sum up their probabilities.

3.3.4. RUN-TIME SCHEDULER

This module is an important component of the system for run-time information gathering and dynamic application execution control. The online scheduler gathers information by monitoring the energy storage medium and the multicore processor (Figure 32). The gathered information, together with preloaded schedule template library generated by the offline scheduler for the given workload (discussed further in Section 3.5), allows the run-time scheduler to coordinate operation of the multicore platform at run-time.

3.3.5. PROBLEM OBJECTIVE

The primary objective of our workload management framework is to allocate and schedule the execution of a workload composed of multiple application task graphs (DAGs) arriving periodically and running in parallel simultaneously at run-time, such that *total task graph miss rate is minimized*. Our framework must react to changing run-time scenarios, such as varying harvested energy budgets, variations in task execution time, and random transient faults, to schedule as many of the task graph instances as possible without overloading the system with complex re-scheduling calculations at run-time. The framework must also consider slack reclamation to aggressively save energy and support soft-error handling to avoid finishing task graphs with incorrect output (which is counted as a task graph miss). As a secondary objective, the framework must take aging effects into consideration to *maximize overall system lifetime*.

3.4. HYBRID SCHEDULING FRAMEWORK: MOTIVATION AND OVERVIEW

The problem of scheduling *weighted **d**irected **a**cyclic **g**raphs* (**DAG**s) on a set of homogeneous cores under some optimization goals and constraints is known to be NP-complete

[64]. This paper addresses the even more difficult problem of scheduling on systems that rely entirely on limited and fluctuating harvested energy. *The limited energy supply prevents the deployment of complex scheduling algorithms at run-time.* Moreover, *execution of applications that will not have enough energy or computation resources to complete due to shortages in harvested solar energy can lead to significant wasted energy with no beneficial outcome.*

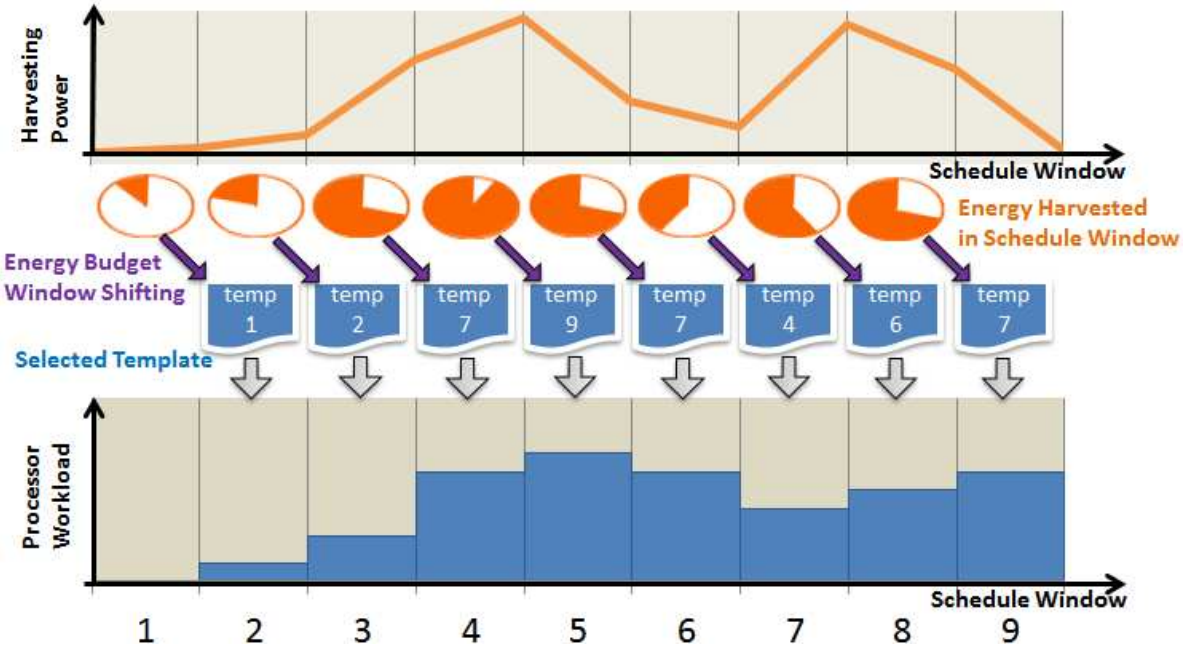

Figure 34 Overview of Hybrid Workload Management Framework

To address these challenges, we propose a ***hy**brid **w**orkload **m**anagement framework* (**HyWM**) that combines *template-based hybrid scheduling* with an *energy budget window-shifting* strategy derived from semi-dynamic framework proposed in Chapter 2 to decouple run-time application execution from the complexity of DAG scheduling in the presence of fluctuations in solar energy harvesting. An important underlying idea in this framework, as shown in Figure 34, is time-segmentation during run-time workload control that creates an independent stable energy

environment for run-time scheduling within each segment. The time of system execution is partitioned into *schedule windows* of identical length, which is referred to as the *hyper-period* of the DAG workload. An energy budget is assigned to a schedule window at its beginning, based on the amount of harvested and unused energy from the previous window. This conservative budget assignment scheme, called *energy budget window-shifting*, can delay utilization of harvested energy slightly to ensure that dynamic variations in energy harvesting do not halt the execution of applications in subsequent windows. The run-time scheduler knows the amount of energy that is available at the beginning of each window, and selects the best-fit schedule template generated at design-time based on this energy budget.

In the following sections, we describe our proposed framework in detail. Section 3.5 describes two design-time scheduling template generation approaches. Section 3.6 presents a run-time scheduler with aging-aware allocation of workload partitions, lightweight slack reclamation, and integrated soft error handling heuristics. Experimental results to validate our framework are presented in Section 3.7.

3.5. OFFLINE TEMPLATE GENERATION

In this section, we propose and discuss two different approaches to solve the DAG scheduling problem at design-time. Both approaches generate schedule templates composed of energy-efficient execution schedules for various energy budgets. The first approach is based on ***m**ixed **i**nteger **l**inear **p**rogramming* (**MILP**) that ensures schedule optimality for maximum performance. The second approach is an ***a**nalysis-based **t**emplate **g**eneration* (**ATG**) heuristic that is faster and more scalable than MILP, to accommodate larger problem sizes with acceptable compromise in schedule optimality.

3.5.1. MILP-BASED OFFLINE TEMPLATE GENERATION

We formulated an MILP problem to aid with the generation of optimal task scheduling templates at design-time. The MILP formulation aims to minimize miss rate for DAG instances in a schedule window under a given energy budget constraint. The constructed formulation is solved multiple times offline with different energy budget constraints to generate a set of schedule templates for the run-time scheduler to select. As our formulation focuses on workload management within an independent schedule window, in this section we assume periodic task graphs in set ψ are unrolled into a set of all task graph instances that arrive within a schedule window, ψ^{+}: $\{GI_{1,\ldots,}GI_{Ni}\}$. Our target processor has N_c cores, each with N_l discrete frequency levels.

3.5.1.1. INPUTS AND DECISION VARIABLES

For our MILP formulation, we provide several inputs that represent the energy budget and characteristics of the target workload and platform, as shown in Table 4. The energy budget parameter (*ENGY_BGT*) allows different schedule template outcomes, such that each of them can best match the available energy budget. The $WCET_{j,l}$ and $ENGY_{j,l}$ parameters are calculated based on worst case execution cycles (*WCEC*) of every task node for every frequency level supported by the processing cores (see Table 1).

Table 4 Inputs for MILP Formulation

Inputs	Description
EGY_BGT	energy budget of the schedule template to generate
$ARRIVAL_i$	arrival time of task graph instance *i*
$DDLINE_{i,j}$	deadline of task graph instance *i node j*
$WCET_{j,l}$	worst-cast execution time of task node *j* at frequency level *l* , $l \neq 0$
$ENGY_{j,l}$	energy consumption of task node *j* at frequency level *l*, when $l = 0$, $ENGY_{j,0} = 0$

$COMM_{src,dst}$	communication delay when preceding node *src* and descendent node *dst* are allocated to separate cores
Ni, Nt, Nl, and Nc	number of task graph instances, number of task nodes, number of frequency levels, and number of cores

† In our formulation, task nodes can be indexed in two different ways:
1) Local ID: tuple *(i, j)* for task node *j* of task graph *i*
2) Global ID: single variable *j* for task node *j* in the entire set

Table 5 Decision Variables in MILP Formulation

Variables	Description
$miss_i$	binary variable to indicate if task graph instance *i* is missed
$start_{(i,j)}$	Execution start time of task graph *i* on node *j*. Note that we also use variable $end_{i,j}$ as the end time of execution. Our schedule does not consider task preemption so that $end_{i,j} = start_{i,j} + WCET_{i,j}$
$freq_{j,l}$	binary variable which indicates if task node *j* is assigned with frequency level *l*
$alloc_{j,k}$	binary variable which indicates if task node *j* is mapped to core *k*, $k \neq 0$
$dec_{j,j'}$	binary variable which indicates if task nodes *j* and *j'* are NOT mapped to the same core (decoupled)
$bef_{j,j'}$	binary variable which indicates if task node *j* is scheduled before *j'*

There are two major requirements for decision variables in our MILP problem: *(i)* they must form a complete representation of a feasible execution schedule; and *(ii)* they should make it possible to represent all constraints and objectives as linear formulations. Table 5 shows decision variables used in our formulation. The binary indicators of task graph miss, $miss_i$, are used to construct the major part of the objective function. For $freq_{j,l}$, when $l = 0$, it indicates that task node *j* is not scheduled for execution and is thus to be dropped. The indicators $dec_{j,j'}$ and $bef_{j,j'}$ are used to construct constraints that arrange timings of task nodes without direct dependencies.

3.5.1.2. OPTIMIZATION OBJECTIVE

The major objective of the MILP formulation is to minimize the number of misses of task graph instances in a schedule window. Additionally, we include an auxiliary objective: the percentage of energy budget used, so that the MILP optimization also searches for a schedule with the least energy consumption possible. Note that this auxiliary objective does not sacrifice minimization of number of task graph misses for less energy consumption, as the energy usage percentage, with value no greater than 1, always has less impact on the objective function value than any single task graph instance miss. The objective formulation is shown below:

$$\text{Min:} \quad \sum_{i=1}^{Ni} miss_i + \left. \sum_{j=1}^{Nt} \sum_{l=0}^{Nl} (ENGY_{j,l} \times freq_{j,l}) \right/ EGY\_BGT \tag{20}$$

3.5.1.3. CONSTRAINTS

The constraints in our formulation guarantee the satisfaction of the energy budget and correctness of the execution schedule for the target workload and platform. The key constraints are described as follows:

- *Energy constraint for a schedule window:* Total energy consumption of all task nodes at their assigned frequency levels should be less or equal to the energy budget:

$$\sum_{j=1}^{Nt} \sum_{l=0}^{Nl} (ENGY_{j,l} \times freq_{j,l}) \leq EGY\_BGT \tag{21}$$

- *Timing constraints for task graph scheduling:* We formulate multiple constraints, which when combined together form a complete timing constraint for all task graph instances and their task nodes, as illustrated in Figure 35.

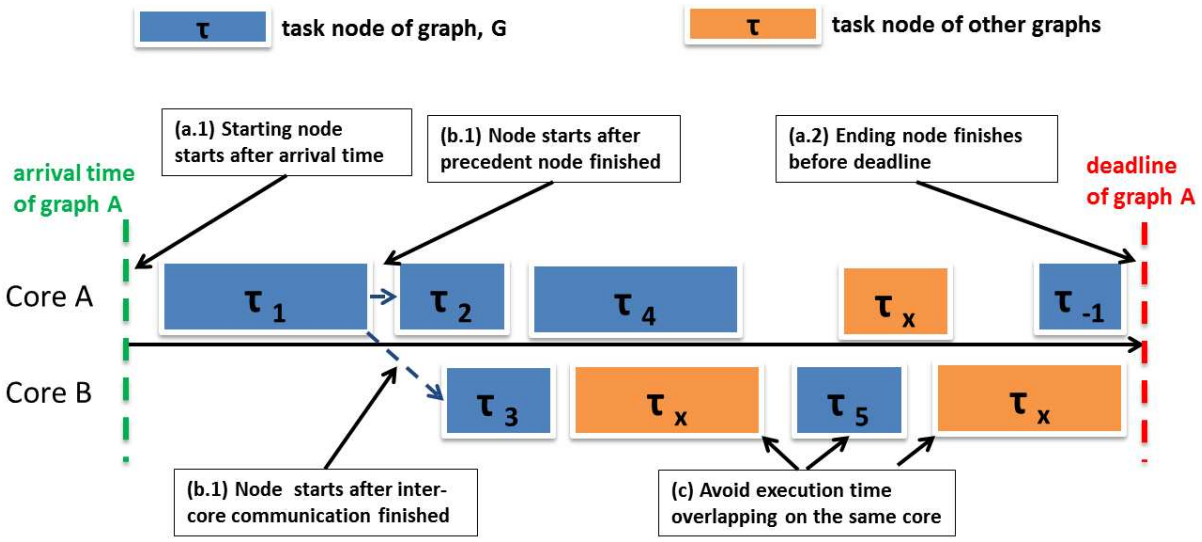

Figure 35 Timing Constraints for Periodic Task Graph Set

- *Timing constraints for graph instances:* The two constraints below confine start time of the first task node and end times of task nodes with deadlines to ensure that timing requirements of their corresponding task graph instances are satisfied, as illustrated in Figure 35 (a.1, a.2).

$$start_{(i,1)} \geq ARRIVAL_i - M \times miss_i \qquad i \in [1, N_i] \tag{22}$$

$$end_{(i,j)} = start_{(i,j)} + \sum_{l=1}^{Nl} (WCET_{(i,j),l} \times freq_{(i,j),l}) \tag{23}$$

$$end_{(i,j)} \leq DDLINE_{i,j} + M \times miss_i \qquad i \in [1, N_i], j \in [1, Nt] \tag{24}$$

We use a sufficiently large constant, M, in the formulation to equivalently represent "if" statements that cancel out constraints when $miss_i = 1$ (graph instance dropped). The constraints can be canceled out when $miss_i = 1$ because large values of M ensure that the inequality is satisfied for any variable values in range. In the rest of this paper, we use the same approach for "if" statements. However, for the purpose of intuitive representation, the following sections show "if" statements explicitly.

- *Timing constraints for task nodes with dependencies:* The type of constraints shown below model dependencies by forcing destination task nodes to start only after their predecessor nodes have finished. Also the constraints take communication cost into consideration when two dependent nodes are decoupled (not allocated to the same core), as illustrated in Figure 35 (b.1, b.2):

$\boldsymbol{if}\ miss_i = 0$:

$$end_{(i,src)} + COMM_{src,dst} \times dec_{src,dst} \leq start_{(i,dst)} \quad (25)$$

$i \in [1, N_i]$, $(src, \text{dst}) \in\ edges\ of\ G_i$, $G_i \in \psi^+$

- *Timing constraints for task nodes without dependencies:* The type of constraints shown below address the fact that task nodes allocated to the same core cannot overlap their execution times, as each core executes only one task at a time without preemption, as shown in Figure 35 (c).

$$dec_{j,j'} \leq 2 - allocs_{j,k} - allocs_{j',k} \quad (26)$$
$$j \in [1, Nt], j' \in [1, Nt], j \neq j', k\ \in [1, Nc]$$

$$dec_{j,j'} \geq allocs_{j,k} + allocs_{j',k'} - 1 \quad (27)$$
$$j \in [1, Nt], j' \in [1, Nt], j \neq j'$$
$$k \in [0, Nc], k \in [1, Nc], k \neq k'$$

These constraints represent relations between task node allocation variables, $alloc_{i,k}$, and node pair decoupling variables, $dec_{j,j'}$. The constraint in (26) ensures that the pair decoupling variable is equal to 0 when task nodes are on the same core. The constraint in (27) forces the decoupling variable to be 1 when two task nodes are found to be allocated to different cores.

With the value of $dec_{j,j'}$ available, the following constraints are used to avoid timing conflicts for every pair of task nodes:

$$bef_{j,j'} + bef_{j',j} - dec_{j,j'} = 1 \quad (28)$$

$$\textbf{if } bef_{j,j'} = 0: \quad end_{j'} < start_j \quad (29)$$

$$\textbf{if } bef_{j',j} = 0: \quad end_j < start_{j'} \quad (30)$$

$$j \in [1, Nt], j' \in [1, Nt], j \neq j' \quad \text{for (28-30)}$$

The constraint in (28) implies that the task node *j* should be scheduled either before or after task node *j'* when they are allocated on the same core. Based on the scheduled order of these two tasks, the constraint in (29 and 30) ensures that the task node only starts when earlier scheduled task nodes are finished. When two task nodes are decoupled to two different cores, the constraints in (29 and 30) cancel out [81].

- *Constraints for target platform:* The type of constraints shown below guarantee that only one frequency level and at most one core are selected for execution of each task node:

$$\sum_{l=0}^{Nl} freq_{j,l} = 1, \quad j \in [1, N_t] \quad (31)$$

$$\sum_{k=1}^{Nc} alloc_{j,k} \leq 1, \quad j \in [1, N_t] \quad (32)$$

$$\textbf{if } freq_{j,0} = 0: \quad \sum_{k=1}^{Nc} alloc_{j,k} = 1, \quad j \in [1, N_t] \quad (33)$$

A task is indicated as dropped in the generated schedule when its frequency level is set to 0. The constraint in (33) ensures that all tasks that are not dropped will be allocated to a core; otherwise they may end up being executed on a "ghost core" to escape timing constraints with other tasks.

All of the above constraints are necessary to create a correct, feasible and optimal set of schedule templates, for a set of chosen energy budgets. We also establish additional constraints (not shown for brevity) to eliminate obviously sub-optimal solutions and reduce the search space for the MILP solver.

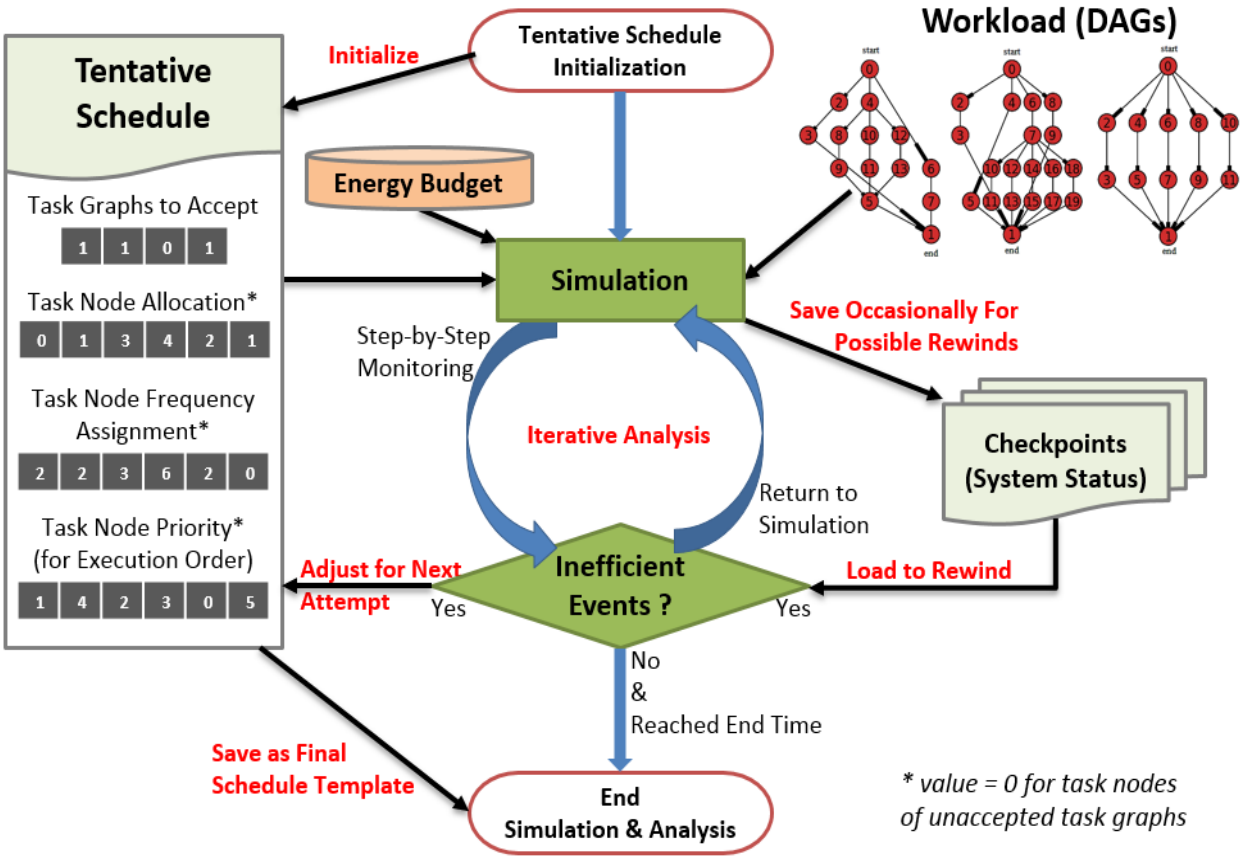

Figure 36 Analysis-Based Schedule Template Generation Heuristic

3.5.2. FAST HEURISTIC-BASED OFFLINE TEMPLATE GENERATION

The MILP optimization approach can provide optimal static schedule templates when online performance is the primary goal and the complexity of the workload is not excessive. For problems with larger sizes, however, the complexity of MILP optimization will increase dramatically such that the execution time of the MILP solver becomes impractical, even for design-time exploration.

Thus we propose another novel ***a**nalysis-based **t**emplate **g**eneration* (**ATG**) heuristic that emphasizes scalability and fast solution generation with an acceptable compromise on the optimality of generated schedule templates.

The outline of our proposed ATG method is illustrated in Figure 36. The main idea in ATG is to iteratively analyze and improve performance of tentative execution schedules based on feedback from step-by-step simulation, which detects energy inefficient events to help make informed updates to the tentative schedule that is evaluated in another round of analysis. ATG also has an in-built checkpoint mechanism to save system status so that a new round of analysis after a rewind event (discussed later) saves time before a modification on a tentative schedule takes effect. The three main components of ATG are outlined below:

Firstly, Algorithm 5 shows the steps to generate an initial tentative schedule for ATG based on a specified energy budget level. The algorithm starts out by finding the workload utilization that can be supported by a given energy budget level (step 1~3). Then the schedule accepts a subset of task graphs for execution and drops the remaining task graphs (step 4~11), while ensuring that task graphs with lower WCECs are more likely to be accepted and the total utilization of the task graphs satisfies the supportable workload utilization for the given energy budget. The generated initial schedule conservatively rules out some obviously sub-optimal portions of the solution space during scheduling and reserves enough headroom for upcoming iterative analysis and scheduling. The resulting initial schedule does not include core allocation and priority assignment of task nodes yet, which will be decided by the list scheduling algorithm used in a later stage.

Algorithm 5 Initializing of Tentative Schedule Template

Input:

(i) ψ, task graph set to be scheduled
(ii) *EGY_BGT*, specified energy budget for one schedule window
(iii) T_{win}, duration of a schedule window
(iv) *num_cores*, number of cores in system

(v) f_{max}, maximum frequency of processors
(vi) U_{Gi}, utilization of periodic task graph Gi
Output:
(i) $miss_i$, binary variables to indicate is task graph Gi is missed/dropped in schedule
(ii) $freq_j$, assigned frequency level of task node τ_j, value range [0, N_l]

1. avg_power ← (EGY_BGT/T_{win})/num_cores
2. find f_{ref}, the highest frequency that can be supported by avg_power
3. $U_{ref} \leftarrow f_{ref} / f_{max}$
4. $U_{accepted} \leftarrow 0$
5. sort ψ according to WCEC of each task graph
6. **while** $U_{accepted} < U_{ref}$:
7. find the task graph with lowest WCEC, G_i
8. $miss_i \leftarrow$ FALSE †
9. **for** τ_j in all task nodes of G_i :
10. $freq_j \leftarrow f_{ref}$
11. $U_{accepted} \leftarrow U_{accepted} + U_{Gi}$

† Default values of all elements in $miss_i$ for all task graphs is *TRUE*

Secondly, a list scheduling based algorithm is adapted to our problem and applied during iterative analysis, as shown in Algorithm 6. The algorithm is divided into two parts: Part I is concerned with task priority assignment, while Part II deals with allocation and execution order scheduling of task nodes.

First, we discuss the priority assignment in Part I. In our application model, not all task nodes in a task graph will have deadlines assigned to represent timing requirements of the corresponding real-time application (see section 3.3.1). For task nodes with deadlines assigned, we refer to their associated deadlines as *explicit deadlines*. On the other hand, for tasks nodes without explicitly assigned deadlines, there still exists a latest-time-to-finish for each of them to allow all remaining task nodes with explicit deadlines to finish. Thus tasks without explicitly assigned deadlines can be said to have *implicit deadlines*. We use implicit or explicit deadline to represent priority of a task node, as the earlier the deadline is, the more urgent it is to finish the task node to avoid a deadline miss for the entire task graph.

Algorithm 6 shows the heuristic in Part I that calculates implicit deadlines of all task nodes by using a nested function to traverse the entire task graph starting from task nodes with explicit deadlines assigned (step 1~4). Then in step 5~9, the nested function is called to back-traverse from nodes with explicit deadlines to predecessor nodes, calculating implicit deadlines of other task nodes in a depth-first manner. As a task node can have multiple successor nodes in a task graph, multiple values of implicit deadline can be derived from different calculation paths or different explicit deadlines of nodes. To address this issue, steps 7 and 8 ensure that only the earliest value among all derived ones is kept as a task node's implicit deadline. An illustrative example of this priority (implicit deadline) assignment heuristic is shown in Figure 37.

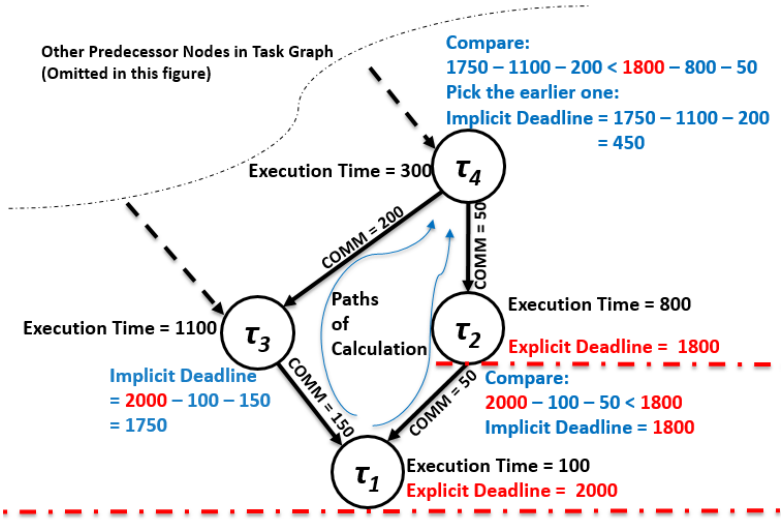

Figure 37 An Illustration Example of Implicit Deadline Calculation

Part II of Algorithm 6 shows the steps for allocating and scheduling task nodes during each simulation step. For task node allocation, a task pool is used to collect task nodes that are ready to be allocated and each core has a record of WCET required to finish all task nodes already assigned to it. A good allocation scheme should distribute task nodes to cores so that their workloads are as

evenly balanced as possible. In steps 10~15, we use a heuristic that is similar to a first-fit decreasing algorithm for the bin-packing problem [82], which sorts task nodes in decreasing order based on their WCETs and then iteratively allocates the task node with highest WCETs to cores with lowest WCETs accumulated for execution. The scheduling of task nodes on each core is performed based on the earliest implicit dead line first (EiDF) algorithm (steps 16~18), which is essentially EDF that uses implicit deadlines generated in part I. With multiple task graphs to be scheduled at the same time, EiDF gives priority to task nodes in the critical path of different task graphs, after comparing their implicit deadlines.

Algorithm 6 List Scheduling Based Approach for Task Scheduling

Part I Task node priority (implicit deadline) assignment
(Called every time tentative schedule is changed)

Input:
(i) ψ, task graph set to be scheduled
(ii) $DD\_LINE_{i,j}$, deadline of task graph instance *i* node *j*
(iii) $WCET_j$, worst cast execution time of each task node in task graph
(iv) $COMM_{src,dst}$, communication delay between node *src* and node *dst*
Output: $implicit\_priority_j$, implicit deadlines as priority indicators of task node τ_j

priority_assign():
1. **for** G_i in ψ :
2. **for** τ_j in task nodes of G_i with deadline constraints :
3. $dead\_priority_j \leftarrow DD\_LINE_{i,j}$
4. call nested_priority(τ_j)

nested_priority(τ_j):
5. **for** $\tau_{j'}$ in all predecessor nodes of τ_j :
6. $implicit\_deadline \leftarrow implicit\_priority_j - WCET_j - COMM_{j',j}$
7. **if** $implicit\_priority_{j'} > implicit\_deadline$:
8. $implicit\_priority_{j'} \leftarrow implicit\_deadline$
9. call nested_priority($\tau_{j'}$)

Part II List scheduling method
(Called in every simulation step)

Input:
(i) *sys_pool*, system task pool, containing task nodes that are ready to allocate
(ii) $core\_pool_k$, task pool for core *k*, containing allocated task nodes that are ready to execute
(iii) $CORE\_WCET_k$, remaining WCET of all task nodes assigned to core *k*

(iv) *implicit_priority*$_j$, implicit deadlines as priority indicators of task node τ_j

Output:

(i) *alloc*$_j$, allocation results of task node τ_j, value range [0, num_cores]

(ii) selected task node to execute in current simulation step

list_schedule():

10. sort sys_pool according to WCET of each node
11. **for** all task nodes in sys_pool :
12. find τ_j in sys_pool with highest $WCET_j$
13. find core *k*, with lowest $CORE\_WCET_k$
14. allocate τ_j to core *k*, $alloc_j \leftarrow k$ †
15. $CORE\_WCET_k \leftarrow CORE\_WCET_k + WCET_{\tau j}$
16. **for** all cores in system :
17. sort core_poolk according to implicit deadline of each tasks
18. select task node with earliest implicit deadline to execute

† Allocated task is not ready to execute until preceding dependencies are resolved

Lastly, at the core of the ATG heuristic is a checkpoint-based iterative analysis method, as shown in Algorithm 7. At the beginning of each simulation step, the ATG heuristic saves the current system status as a checkpoint for newly arriving task graphs, so that the simulation can rewind to this checkpoint saved before the schedule for the new task graph takes effect (step 2~3). Subsequently, a list scheduler is invoked and the system executed for one simulation step with the tentative schedule (step 4~5). When energy inefficient events are detected during execution, the ATG heuristic will update the execution schedule accordingly and rewind to a previous checkpoint for another round of evaluation with an updated schedule (step 6~16). If ATG detects depletion of the energy budget before finishing all accepted task graphs in the current schedule (energy violation event), one accepted task graph with highest WCEC will be dropped in the updated schedule and simulation rewinds for re-analysis (step 6~9). If ATG detects a task node that missed its implicit or explicit deadline (timing violation event), which implies that a deadline miss for the task graph it belongs to is inevitable, the tentative schedule will be updated to boost execution frequency of related task nodes: the task node in the critical path with the lowest frequency assigned will get a frequency boost (step 11~13); and if there exists a task node from another task

graph allocated to the same core that finished just before the nodes with timing violation, it will also get a frequency boost (step 14~15).

Note that WCETs of selected task nodes change with their boosted frequencies, thus we call a nested priority assignment function starting from these nodes to recalculate implicit deadlines of their predecessors. Then simulation rewinds for re-analysis with the new schedule (step 16). If the current simulation step detects no energy inefficient events, the simulation continues to the next step (step 17~18). When the entire schedule window is analyzed without energy inefficient events, the analysis process ends and the updated schedule is saved as a schedule template for the specified energy budget (step 19).

Algorithm 7 Checkpoint-Based Iterative Analysis

Input:
(i) *EGY_BGT*, specified energy budget for one schedule window
(ii) T_{win}, duration of a schedule window
(iii) $implicit\_priority_j$, implicit deadlines as priority indicators of task node τ_j
(iv) initial tentative schedule from Algorithm 5
Output: static schedule template for energy budget of *EGY_BGT*

1. **while** $T_{cur} < T_{win}$:
2. **if** new task graph G_i arrives :
3. $checkpoint_i \leftarrow$ all system status (include T_{cur})
4. alloc ← list_schedule()
5. execute for one step using tentative schedule
6. **if** EGY_BGT depleted during execution :
7. find arrived task graph with highest WCEC, G_i
8. $miss_i \leftarrow$ TRUE
9. all system status ← $checkpoint_i$
10. **else if** node τ_j of task graph G_i missed its implicit deadline :
11. find the critical path in G_i that ends at τ_j
12. find $\tau_{j'}$, the task node with lowest frequency assigned
13. $freq_{j'} \leftarrow freq_{j'} + 1$, nested_priority($\tau_{j'}$)
14. find $\tau_{j''}$, the task finished just before τ_j on the same core
15. $freq_{j''} \leftarrow freq_{j''} + 1$, nested_priority($\tau_{j''}$)
16. all system status ← $checkpoint_i$
17. **else** :
18. $T_{cur} = T_{cur} + T_{step}$
19. save final tentative schedule as schedule template

At design-time, the ATG heuristic is executed multiple times for different energy budget levels (similar to the MILP approach) to generate a set of schedule templates for the run-time scheduler to select from, based on the harvested and available energy in the target multicore computing platform.

3.6. ADAPTIVE ONLINE MANAGEMENT

3.6.1. RUN-TIME TEMPLATE SELECTION

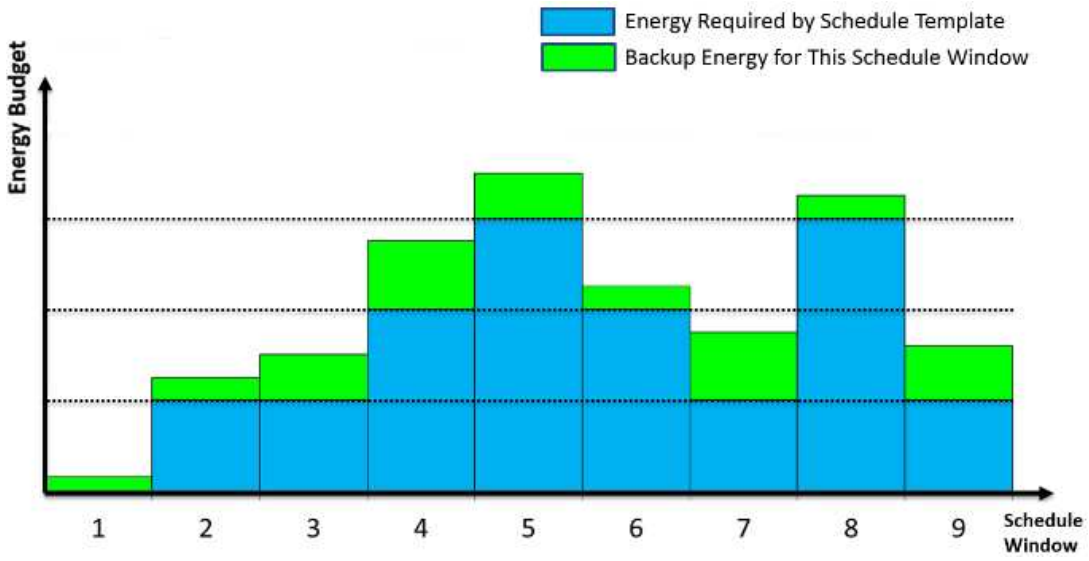

Figure 38 Residual Energy Availability over Time

The main goal of our run-time scheduler is to monitor harvested solar energy and select the best-fit template for an upcoming schedule. With schedule templates generated at design-time and energy budgets provided at the beginning of each schedule window, this is a low-overhead operation, done by selecting the schedule template that finishes the most task graph instances, contingent on the energy budget. Each selected template provides a schedule with task-to-core allocation, execution order, and frequency assignment for every task node. As the offline schedule

template assumes all cores to be identical, each task node is actually only assigned to a virtual core id. We call a set of task nodes assigned offline to a core id as a *workload partition*, each of which should be allocated to a dedicated physical core for execution in the upcoming schedule window. This partition-to-core allocation can be adjusted dynamically to mitigate aging effect that leads to hard failures (see section 3.6.2). On the other hand, the amount of residual energy that exceeds the energy requirement of the selected schedule template is used as backup energy (Figure 38) for possible task re-execution to recover from detected faults caused by soft errors during execution (see section 3.6.3).

3.6.2. AGING-AWARE ALLOCATION OF WORKLOAD PARTITIONS

After a schedule template is selected based on the energy budget for a schedule window, our framework can trigger a scheme to allocate workload to cores with awareness of core aging to enhance system lifetime. Although schedule templates set fixed execution strategies for all task nodes, there still exists some flexibility as the allocation of *workload partitions* to cores can still be altered from the default provided by the schedule template, for a homogeneous multicore platform.

Algorithm 8 Dynamic Workload Distribution in Awareness of Core Aging

Input:
(i) *work_partition_set*, set of workload chunks in schedule template, each chunk should be executed on an individual core
(ii) *R_set*, reliability of cores
Output: Allocation of workload partitions to cores

1. **for** each reliability detection interval :
2. update R_set
3. sort cores in non_decreasing order of hard reliabilities
4. **for** each schedule window :
5. get work_partition_set from selected schedule template
6. sort work_partition_set in non-decreasing order of workload

partitions' total task execution cycles
7. **for** all cores in system :
8. allocate workload partition with lowest execution cycles to unassigned core with lowest hard reliability

The outline of our aging-aware dynamic workload allocations scheme is shown in Algorithm 8. We assume that our scheduler can fetch hard reliability information of cores from aging detection circuitry [83] or execution history tracking mechanisms at certain interval (much longer interval than schedule windows) [75] (steps 1~3). Besides, at the beginning of each schedule window, workload partitions are fetched from the selected schedule template (step 4~6). Then recursively our heuristic allocates unassigned workload partitions with the lowest workload intensity to idle cores with the lowest hard reliability. As a result, cores with faster wear-out during previous system operation are more likely to receive less workload than others so that aging processing on the entire multicore system can be rebalanced. Otherwise, some cores may be utilized more intensively than others and detrimentally impact system lifetime of the multicore chip.

3.6.3. DYNAMIC ADJUSTMENT FOR SLACK RECLAMATION AND SOFT ERROR HANDLING AT RUN-TIME

Utilizing static schedule templates for run-time workload management shifts the burden associated with the complex task graph scheduling problem to design-time. However, embedded systems can encounter unpredictable variations at run-time such as those due to fluctuations in harvested solar energy, slight variations in task execution time on the same core, and randomness of soft error occurrences. Among these factors, the fluctuations in harvested solar energy are already dealt with in our framework by using the energy budget window-shifting technique and

the schedule template set prepared for different energy budget levels. In this section, we introduce a lightweight run-time management scheme that provides an integrated solution to address slack reclamation and soft error handling without diminishing the benefits of schedule templates generated at design-time. This scheme is described in Algorithm 9.

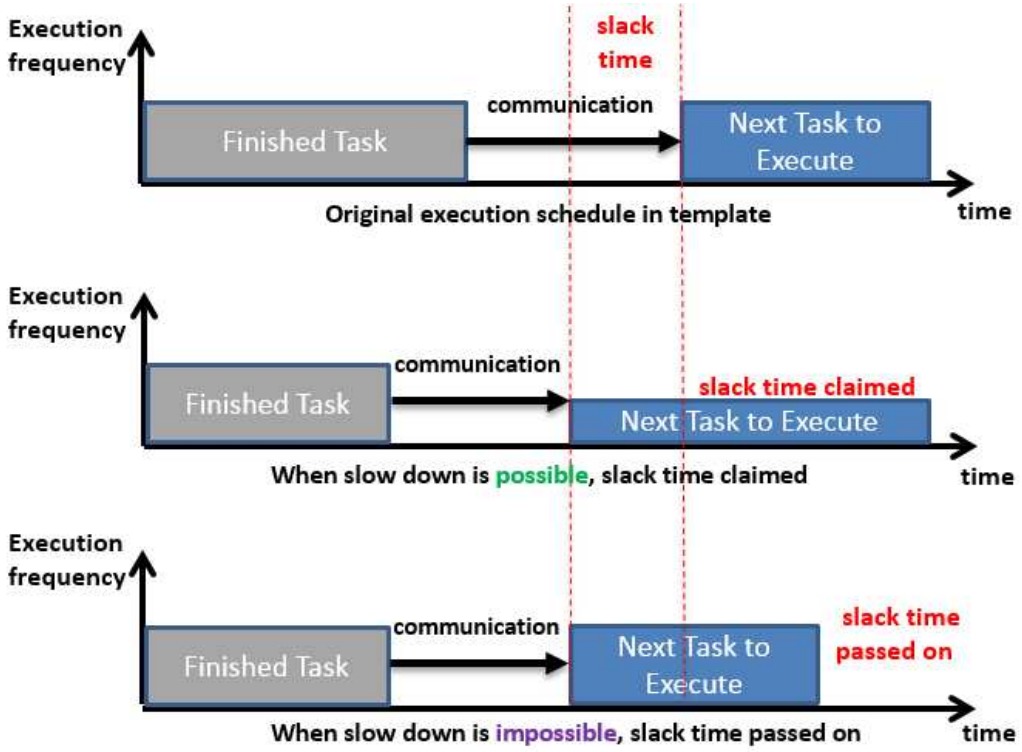

Figure 39 Illustrative Example of Slack Time Reclamation

Our run-time management scheme can reclaim slack time that becomes available when a task node finishes before its worst case finishing time. This slack time can be used to slow down execution of upcoming task nodes, to save energy. The offline generated schedule templates have a designated start time recorded for all task nodes, to help identify any instances of slack time. Whenever a new task node is about to start execution, the amount of slack time is calculated by subtracting the node's designated start time with the current time (steps 4~5). If the amount of

slack time is usable, slower execution frequency is assigned to the task node for the purpose of saving energy (step 6~7). Even if the amount is not sufficient to step down a frequency level, the task node will start execution earlier than the designated time and thus the slack time can be passed on to upcoming tasks, as shown in Figure 39. The estimated amount of energy saved is added to the backup energy for use during possible task re-execution in the presence of soft errors (step 8).

Algorithm 9 Dynamic Slack Reclamation and Soft Error Handling

Input:
(i) T_{win}, duration of a schedule window
(ii) ψ, task graph set to be scheduled
(iii) $start_j$, designated time to start execution of τ_j in selected schedule template
(iv) $bkup\_energy_j$, amount backup energy for a schedule window
Output: static schedule template for energy budget of EGY_BGT

1. **while** $T_{cur} < T_{win}$:
2. load schedule in template
3. **for** τ_j in taskpool :
4. **if** τ_j is about to start execution and $T_{cur} < start_j$:
5. slack_time ← $start_j - T_{cur}$
6. **while** slack_time > WCET increased at $freq_j - 1$:
7. $freq_j \leftarrow freq_j - 1$
8. bkup_energy ← bkup_energy + energy saved
9. execute task nodes based on schedule template
10. **for** τ_j in just finished tasks :
11. **if** error detected on τ_j :
12. **if** $T_{cur} \leq start_j$:
13. schedule another instance of τ_j to re-execute
14. **else if** ∃ a freq that has reduced WCET > T_{cur} - $start_j$: and can be supported by bkup_energy:
15. $freq_j \leftarrow$ freq,
16. bkup_energy ← bkup_energy – energy_used
17. schedule another instance of τ_j to re-execute
18. **else** :
19. find next node to execute on the same core, $\tau_{j'}$
20. **if** $\tau_j \in G_i$, $\tau_{j'} \in G_{i'}$ and $G_i \neq G_{i'}$:
21. update remain WCEC of both graphs
22. **if** $G_{j'}$ has more WCEC :
23. drop $G_{j'}$
24. schedule τ_j to re-execute
25. **else** :
26. drop G_j

27. **else** :
28. drop G_j

Our run-time management scheme is also capable of reacting to soft errors with node-to-node soft error detection. Whenever a task node finishes execution, the correctness of the result is verified to trigger an error handling heuristic if errors are detected during task node execution (steps 10~11). If there is slack time directly available, the system reclaims it to execute a new instance of the faulty task node (step 12~13). If sufficient slack time is not available, the error handling heuristic checks to determine if there exists a higher frequency supportable by the available backup energy to finish re-execution of the fault-affected task node before its implicit deadline (steps 14~17). If both options are not viable for the faulty task node, the heuristic will attempt to drop other task graphs with higher WCEC so that the faulty node can be rescheduled. This process involves checking if the next node scheduled to execute on the same core is from another task graph (step 18~20). If true, both task nodes have the WCEC of their unfinished nodes updated and the task graph with the higher WCEC is dropped (step 21~26).

The three error handling stages described above attempt to exploit slack time, backup energy and relatively less important task graphs to save the computation efforts invested into all predecessor nodes of the faulty task node, for better overall energy efficiency. During slack reclamation and error handling, all task nodes that do not belong to faulty or dropped task graphs will not have their template-designated finish time compromised, thus a chosen schedule template remains effective during run-time workload management.

3.7. EXPERIMENTAL RESULTS

3.7.1. EXPERIMENT SETUP

We developed a simulator in C++ to evaluate our proposed soft and hard reliability-aware ***hy**brid **w**orkload **m**anagement framework* (**HyWM**). For offline schedule template generation, we wrote a python script that constructs the data structure of task graphs using the NetworkX package. We formulated the MILP problem using a ***G**NU **l**inear **p**rogramming **k**it* (**GLPK**) [84]. We chose the Gurobi Optimizer [85] as our MILP solver to generate the optimal schedule templates. We generated task graph sets based on the *networking*, *telecom*, and *auto-industry* applications from the ***E**mbedded **S**ystem **S**ynthesis Benchmark **S**uite* (**E3S**) [86] and the distribution of actual execution times of task nodes is obtained from [78]. We also used synthetic task graph sets from TGFF [87]. In the rest of this section, we first analyze characteristics of the generated schedule templates and then study system performance for our proposed hybrid workload management scheme compared to prior work.

3.7.2. TEMPLATE GENERATION ANALYSIS

In the first set of experiments, we check the quality and optimality of the schedule templates generated using our MILP approach on a 4-core system. We scale task node execution time of four periodic task graphs from E3S with computation utilization set to 0.8×4 and communication utilization set to 0.15×4, i.e., a total workload utilization of 0.95×4, which sets a stringent timing requirement for a system with 4 cores. The resulting periodic task graphs with targeted utilization have periods ranging from 20 to 60 seconds and execution times at 1000MHz operating frequency ranging approximately from 16 to 48 seconds with maximum per-graph parallelism of 4. Besides, apart from the deadlines at task-graph termination nodes, we randomly select few task nodes in

each task graph to assign explicit deadlines that result in even more stringent timing requirements (Note: utilization of the entire task graph stays the same as it is calculated based on maximum frequency; see section 3.3.1). Based on the periods of the generated task graphs, we set the length of schedule window to be 1 minute, within which 9 task graph instances arrive in the system for execution. We generated 11 schedule templates with energy budgets evenly distributed from 0 to E_{peak}, where E_{peak} is the assumed peak energy budget (240 Joules) available from our solar energy harvesting system.

Table 6 Results of MILP Based Schedule Template Generation for A 4-core Homogeneous Embedded System

Schedule template ID	Energy budget	Objective value	Energy budget usage	Energy usage	Number of misses
0	0J	9.000	0.0%	0J	9
1	24J	7.846	84.6%	20.3J	7
2	48J	5.920	92.0%	44.2J	5
3	72J	4.968	96.8%	69.7J	4
4	96J	4.726	72.6%	69.7J	4
5	120J	3.808	80.8%	97.0J	3
6	144J	2.904	90.4%	130.2J	2
7	168J	2.775	77.5%	130.2J	2
8	192J	1.923	92.3%	177.2J	1
9	216J	1.820	82.0%	177.2J	1
10	240J	0.965	96.5%	231.6J	0

The results of the schedule template generation for a system with four cores are shown in Table 6. We can observe that schedule template 10, with a peak energy budget can finish all task instances in time, showing the competence of our MILP optimization to deal with stringent timing constraints even for heavy workloads with per-core utilization as high as 0.95. Note that while 96.5% of E_{peak} is required to finish all task instances, template 3 with energy budget less than $1/3^{rd}$ of E_{peak} managed to successfully schedule more than half of the instances. *The results demonstrate how our approach can create efficient schedules even under highly constrained energy budget*

requirements. The schedule performance is a reflection of our MILP optimization approach that finds the optimal schedule by sacrificing more energy-hungry task graph instances, reserving energy for less energy-hungry ones, and scaling down execution frequency whenever possible for optimal energy efficiency, thereby minimizing the miss rate of task graphs. Note that there are three pairs of templates in Table 6 that are identical to each other with the same extent of energy usage and instance misses. Thus it is unnecessary to increase number of budget levels indefinitely (much beyond number of application task graph instances in a window) as the resulting smaller energy budget difference between levels will lead to identical and redundant schedule templates that increase storage overheads.

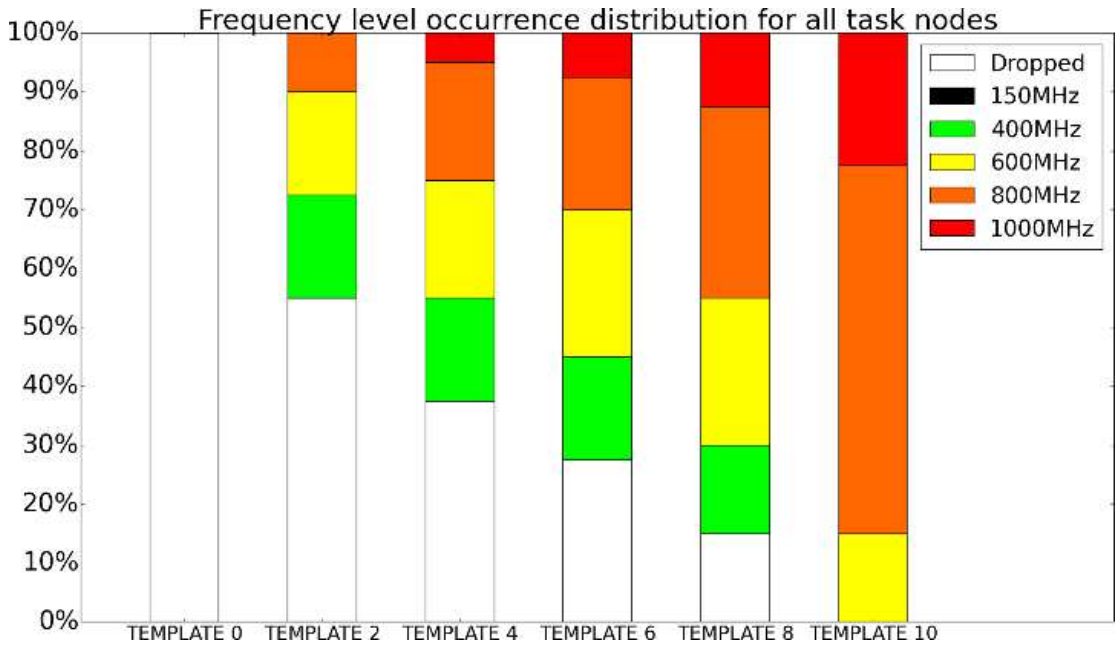

Figure 40 Frequency Level Occurrence Distribution for All Task Nodes

To study the quality of schedule templates from another perspective, we show how our MILP optimization approach selects frequencies for task nodes under different energy budget constraints, as shown in Figure 40. We can observe from the figure that templates with higher energy budgets

utilize higher frequency levels more frequently than templates with lower budgets. Templates with lower energy budget end up dropping more tasks and slow down execution for better energy efficiency. Note that the 150MHz frequency is never used by any schedule; this is due to the fact that the frequency level of 150MHz has lower efficiency and lower speed than the 400MHz level (see Table 1 in Chapter 2). Therefore our MILP optimization approach rules out this sub-optimal frequency choice as it is always better to schedule at 400MHz instead.

While the MILP approach generates optimized schedule templates, we found that the approach is not scalable for larger problem sizes. Table 7 shows a comparison between the MILP and ATG heuristics, in terms of execution time and memory footprint, for two problem instances of different sizes. It can be observed that the MILP approach requires significant computation resources for large problem sizes, which may not be practical even at design-time. The ATG heuristic is much faster, but this speedup comes at the cost of lower performance due to sub-optimal schedule templates generated (see next section).

Table 7 Computation Resource Requirement of MILP and ATG

Method	Complexity		Memory footprint	Execution time
	Number of nodes	Number of edges		
ATG	36	44	42 MB	0.1hour
MILP			257 MB	6.5hour
ATG	150	193	61 MB	1hour
MILP			7693 MB	492hour

3.7.3. EVALUATION OF SYSTEM PERFORMANCE WITHOUT ERROR INJECTION AND EXECUTION TIME VARIANCE

In this section, we compare overall system task graph miss rate for the two variants of our hybrid workload management framework: HyWM-LP and HyWM-ATG, against workload management approaches proposed in prior work. Our simulation uses realistic energy harvesting

profiles based on historical weather data from Golden, Colorado, USA, provided by the ***M**easurement and **I**nstrumentation **D**ata **C**enter* (**MIDC**) of the ***N**ational **R**enewable **E**nergy **L**aboratory* (**NREL**) [60]. As we assume that our system only operates in daylight, system performance is evaluated over a span of 750 minutes from 6:00 AM to 6:30 PM, when solar radiation is available.

To compare our approach with state-of-the-art approaches, we implemented two additional schemes: 1) SDA from Chapter 2, which divides system execution time into segments and selects a stable frequency to execute a subset of the workload that can be supported by the assigned energy budget; and 2) LP+SA [88], which finds a feasible but non-optimal schedule using MILP, and uses this schedule as an initial solution to a ***s**imulated **a**nnealing* (**SA**) based heuristic that finds a near-optimal solution. To compare HyWM with these approaches, we adapt the techniques to our environment and problem formulation. As SDA is designed for energy-constrained scheduling of independent periodic tasks while our workload in this section consists of multiple task graphs, we enhance these techniques so that our scheduler module analyzes inter-task dependency and provides ready task nodes for the techniques to schedule. In LP+SA, the original approach focuses on task graph scheduling while minimizing energy but without awareness of energy harvesting and not considering task dropping. We enhanced LP+SA by dropping tasks iteratively till the remaining task sets meet the energy budget, and these task sets are then sent as inputs to LP+SA.

The results of our comparison study on task graph sets extracted from E3S are shown in Figure 41. The figure shows the total task graph miss rate for three different platform complexities (with 4, 8, and 16 cores). For the platform with 4 cores, it can be observed that SDA has the highest miss rate. This is because SDA, with no awareness of task node dependencies, cannot arrange specific execution schedules for task nodes along critical paths of task graphs and thus all nodes

in a task graph are assigned the same frequencies, resulting in a less efficient schedule. LP+SA outperforms SDA as it can generate task dependency-aware offline schedules after comprehensive design space exploration unlike in SDA. However, the superior offline schedules obtained using our MILP formulation in the HyWM framework coupled with its intelligent run-time template selection and slack reclamation techniques allow HyWM to outperform both of these efforts. HyWM-LP reduces absolute miss rate by 5.6% and 9.0% over LP+SA and SDA, respectively. In terms of relative performance improvement, HyWM-LP accomplishes an improvement of 12.9% and 20.1% over LP+SA and SDA, respectively. HyWM-ATG ends up with higher miss rates than HyWM-LP, however it still outperforms the other two techniques from prior work. HyWM-ATG can however serve as an alternative approach when scalability is an issue, e.g., for larger problem sets.

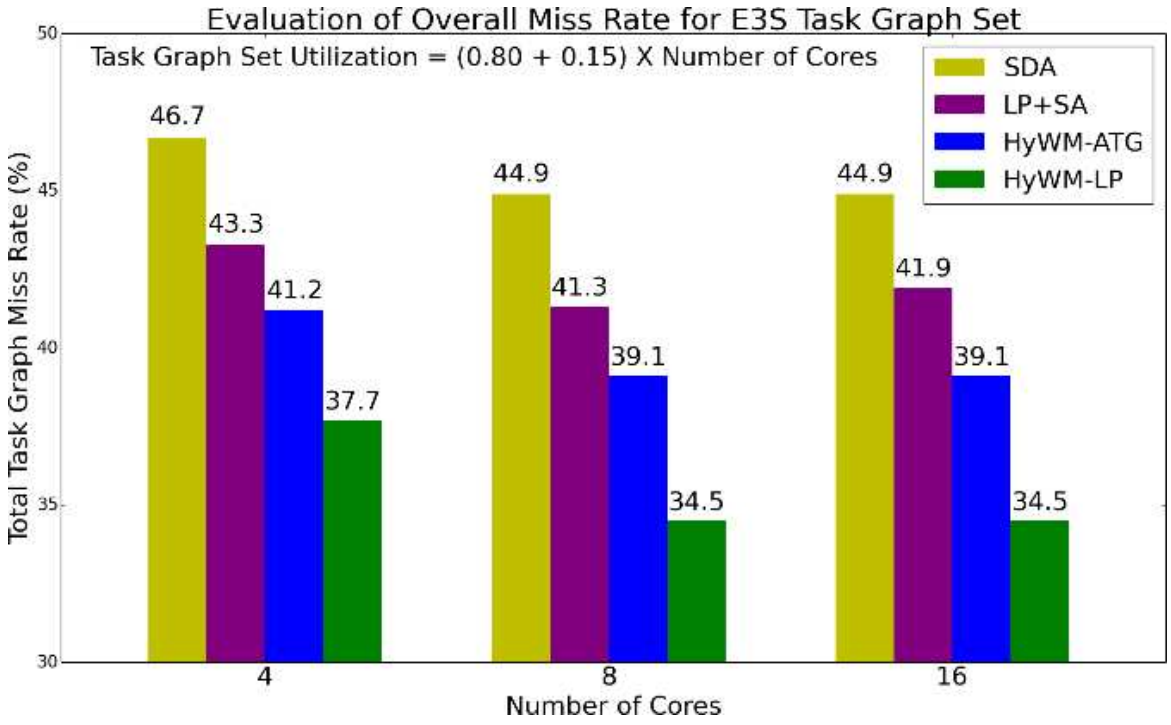

Figure 41 Task Nodes Comparison in Terms of Overall System Task Graph Miss Rate

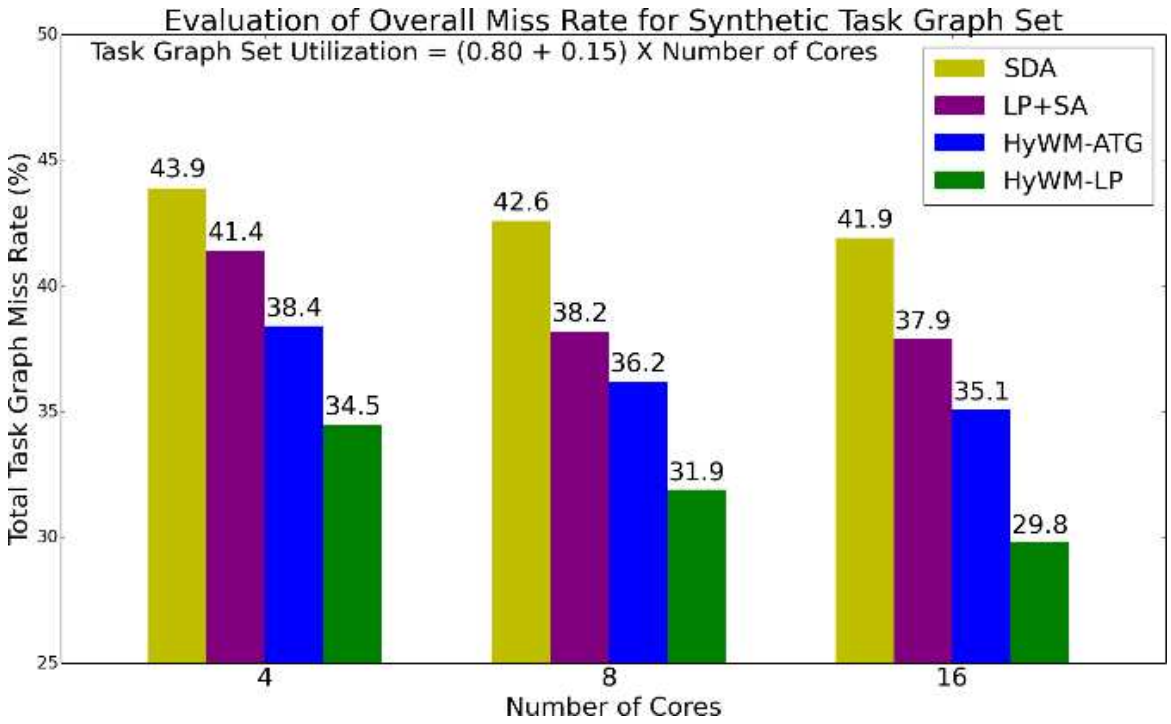

Figure 42 Comparison of Overall System Task Graph Miss Rate on Synthetic Task Graph Set with Higher DoP

Figure 41 also shows the scheduling performance of these frameworks for platforms with a greater number of available cores while keeping the workload and energy budget the same. When the core count doubles from 4 to 8, our two *HyWM* methods achieve lower miss rates (up to 23.2% reduction relatively) compared to other techniques, as they can better distribute the workload across more cores, directing these cores to operate at a lower execution frequency and with better energy efficiency. However, the system with 16 cores shows no further improvements because there is no additional parallelism available in the E3S task graph set, which has maximum per-graph parallelism of 4, to make use of the 16 cores. Note that LP+SA shows a slightly deteriorated result on 16 cores because even though there is no more parallelism to exploit, the search space of its SA heuristic enlarges, leading to slightly worse near-optimal solutions. Figure 42 shows another

group of results based on a synthetic task graph set generated using TGFF [87], with the same targeted utilization as E3S but maximum per-graph parallelism increased to 8. We can observe in Figure 42 that while performance differences among techniques are similar to the results shown in Figure 41, all techniques continue to get miss rate reduction on a 16-core system, as there is additional parallelism to exploit in the synthetic task graphs set (in contrast, miss rate improvements for E3S saturate for the 16-core system as shown in Figure 41).

3.7.4. EVALUATION OF SYSTEM PERFORMANCE WITH SOFT ERROR INJECTION AND EXECUTION TIME VARIANCE

In this section, we show the performance improvements due to our proposed run-time slack reclamation and error handling heuristics. In the experiment, we assume an average error rate of 10^{-5} soft errors per second per core at maximum frequency [70]. As there is no prior work on soft error handling for systems with energy harvesting, we conduct multiple tests with run-time management features enabled progressively on a 4-core system to show each feature's effectiveness, with results shown in Figure 43. Each of the configurations shown in the figure are described below:

- ***None*:** This base case uses HyWM-LP with soft error injection and no run-time adjustment technique enabled, and has a miss rate of 45.4%.
- ***+slack reclamation*:** System miss rate drops to 34.4% when the slack reclamation capability in run-time heuristic is activated.
- ***+drop*:** With the addition of basic soft error-awareness that causes faulty task graphs to be dropped as soon as an error is detected (to avoid unnecessary energy consumption), the miss rate reduces further to 31.9%.

- ***+compare before drop*:** When the heuristic adds support for dropping other task graphs with high WCET to allow re-execution of the faulty task node, the system sees a drop in miss rate to 30.2%.
- ***+backup energy*:** Finally, when the fully-enabled heuristic is utilized that adds further support for utilizing backup energy to speed up faulty node re-execution, we end up with the lowest miss rate of 25.6%.

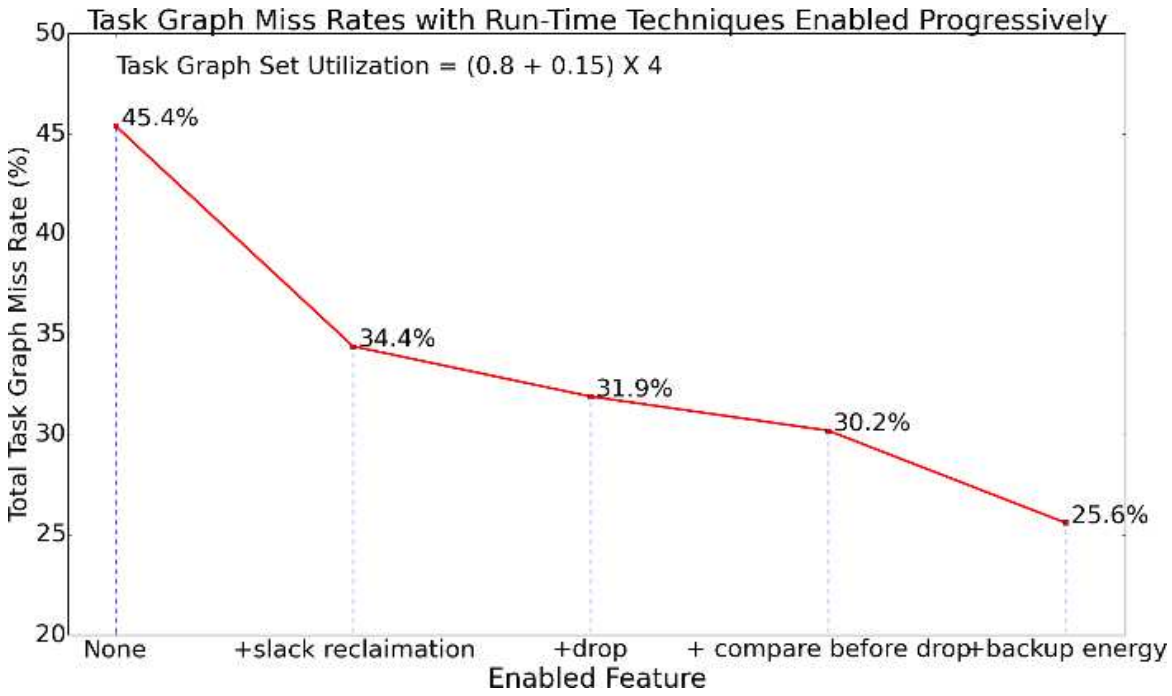

Figure 43 Miss Rate Comparison with Run-Time Techniques Enabled Progressively

The results in Figure 43 highlight the significance of slack reclamation and soft error handling in our run-time framework with a relative 43.6% miss rate reduction for the best configuration compared to the baseline case.

3.7.5. EVALUATION OF SYSTEM HARD RELIABILITY AND MTTF

In this section, we explore the impact of aging on multicore embedded systems with energy harvesting. For our experiments, we implemented the aging model proposed in section 3.3.3, considering ***e**lectro**m**igration* (**EM**) as the primary hard failure mechanism. In the model, we set the critical currently density $J_0 = 1.5\times10^6$ A/cm^2, the activation energy $E_a = 0.48$eV, and assume a slope parameter in the Weibull distribution $\beta = 2$ [72]. We simulated execution of systems over a long period of time with solar harvesting profiles randomly selected from a preset pool. At the beginning of each schedule window, the aging progress is estimated based on average core frequencies, supply voltages, and core temperatures of previous schedule windows. All experiments in this section target 8-core systems executing the same workload as in experiments of previous sections.

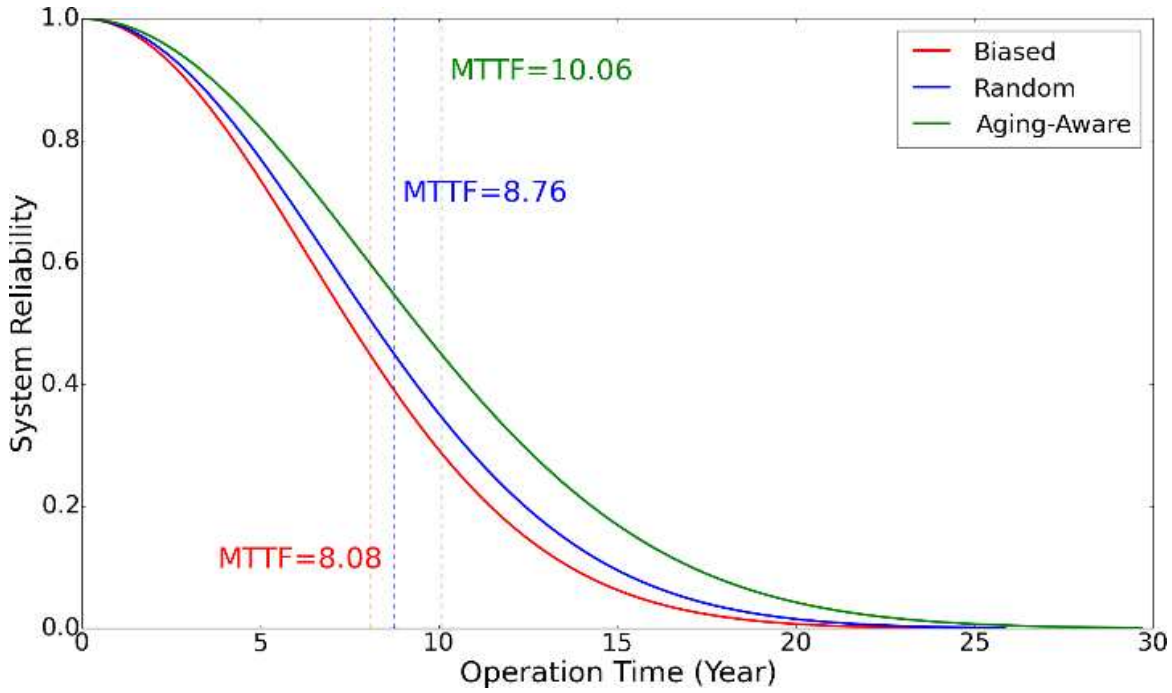

Figure 44 Comparison of reliability and MTTF for different workload allocation schemes

The first set of experiments is designed to evaluate the benefit of our aging-aware workload allocation scheme, which is compared with *Biased*, an allocation scheme that always allocates workload partitions with low to high workload intensities on to cores with low to high core id respectively, and *Random*, the original partition-to-core allocation randomly generated during schedule template generation. All experiments have failure thresholds set to 0 and the results on hard reliability and MTTF of system are shown in Figure 44. We can observe that our *Aging-Aware* scheme (which is used in our HyWM framework) results in better hard reliability over time as it can reallocate workload partitions to balance aging progress among cores, ending up with 14.8% and 24.5% MTTF improvements compared to *Biased* and *Random* without diminishing system performance.

The last set of experiments performs sensitivity analysis for our aging-aware workload allocation scheme, focusing on system MTTF and performance analysis when different failure thresholds are considered. The results of this experiment are shown in Table 8. As we can see, increasing failure threshold allows the system to operate for longer periods of time (higher MTTF), however, this comes at the cost of a decrease in peak processing capability before failure and average system processing capability over time.

Table 8 System MTTF and Performance Comparison with Different Failure Thresholds

Failure Threshold*	0	1	2	3	4	5	6	7
MTTF (years)	10.06	15.58	20.22	24.66	29.27	34.44	40.94	51.35
Processing Capability Before System Failure (%)	100	92.3	80.1	68.8	52.4	34.9	18.4	7.0
Average Processing Capability during System Lifetime (%)	100	96.5	92.5	87.7	81.7	75.0	67.3	60.3

*Failure threshold: number of cores that must fail before a chip is considered unusable

3.8. CHAPTER SUMMARY

In this chapter, we proposed a hybrid design-time and run-time framework for reliable resource allocation in multicore embedded systems with solar energy harvesting. Our framework was shown to cope with the complexity of an application model with data dependencies and run-time variations in solar radiance, execution time, and transient faults. Our experimental results indicated improvements in performance and adaptivity using our framework, with up to 23.2% miss rate reduction compared to prior work, 43.6% performance benefits from adaptive run-time workload management compared to a baseline framework with no soft error and slack time handling, and up to 24.5 % expected system lifetime improvement with aging-aware workload allocation compared to aging-agnostic schemes, under stringent energy constraints and varying system conditions at run-time. With the increasing prevalence of energy-constrained computing, energy scavenging, execution time variability, and the rise in soft errors and hard failures with technology scaling, our proposed framework provides a comprehensive and practical solution that considers all of these factors to perform efficient resource management that improves upon prior efforts in both scope and performance, for emerging multicore embedded computing platforms.

4. MIXED-CRITICALITY SCHEDULING ON HETEROGENEOUS SYSTEMS

In this chapter, we utilize the semi-dynamic approach proposed and utilized in previous chapters to address the scheduling problem for single-ISA heterogeneous multicore processors running hybrid mixed-criticality workloads with a limited and fluctuating energy budget provided by solar energy harvesting. The hybrid workloads consist of a set of *firm-deadline timing-centric task graphs* and a set of *soft-deadline throughput-centric multithreaded applications*. Our framework exploits traits of the different types of cores in heterogeneous multicore systems to service timing-centric workloads with a few big out-of-order cores, while servicing throughput-centric workloads with many smaller in-order cores clocked in the energy-efficient ***n****ear-****t****hreshold* ***c****omputing* (**NTC**) region. Guided by a novel timing intensity-aware penalty density metric, our proposed mixed-criticality scheduling framework creates an optimized schedule that minimizes overall miss penalty for a time-varying energy budget. Experimental results indicate that our framework achieves a 9.5% miss penalty reduction with the proposed timing intensity metric compared to metrics from prior work, a 13.6% performance improvement over a state-of-the-art scheduling approach for single-ISA heterogeneous platforms, and a 23.2% performance benefit from exploiting platform heterogeneity.

4.1. BACKGROUND AND CONTRIBUTION

Recent years have seen billions of embedded systems deployed around the world to support a variety of different applications domains. For an increasing number of embedded applications, there is a critical need for energy autonomous devices that can utilize ambient energy from the environment to perform computations without relying on an external power supply or frequent

battery charges. As the most widely available energy source, solar energy has become an important source of ambient energy for several harvesting-aware embedded systems.

As discussed in Section 1.2.2, embedded computing systems that include timing behavior as part of their performance or correctness criteria are called real-time embedded systems. In such real-time systems, a deadline is called *firm* if missing it results in an immediate performance penalty, otherwise the deadline is considered to be *soft*. If critical system failure can happen after a deadline miss, the deadline is considered to be a *hard* deadline [89]. Due to the variable nature of solar radiation intensity, the most suitable role of embedded systems with solar energy harvesting as the only energy source is to host applications without strict real-time requirements. Thus it may not be desirable to consider such systems for real-time applications with *hard* deadlines, such as life-support mechanism, automotive system control, aircraft navigation, etc., for which any deadline miss is consider a critical system failure that may have catastrophic consequences. Instead, it is more practical to deploy such systems without energy guarantees for best-effort execution of applications where a *firm* or *soft* deadline miss is not considered a failure of the entire system.

Consider an example of such a best-effort embedded system powered by energy harvesting, which is deployed for continuous data collection, data post-processing, and data transmission at a remote location. For each operation interval, a raw data point can be recorded from sensor modules by executing certain control tasks, for which each miss immediately results in inaccuracy in the averaged values of data features. Such tasks can be considered to be *timing-centric* with *firm deadlines*. On the other hand, post-processing of raw data and data transmission tasks can be delayed somewhat as the system can buffer a certain amount of raw data or clients can accept lower rate of transmitted data. Such tasks are generally *throughput-centric* with *soft deadlines*. In

this chapter, we represent such applications with different levels of real-time constraints as mixed-criticality workloads that consist of a mix of timing-centric tasks with firm deadlines and throughput-centric tasks with soft deadlines [90] [91].

Recent years have also seen the rise of multicore processing and heterogeneous computing in low-power embedded devices [23] [24]. Multicore processors with heterogeneous cores have been shown to provide substantial improvements in energy-efficiency and performance for energy-constrained systems [92]. With the rise in computing capabilities of emerging heterogeneous multicore processors, run-time workload distribution and energy-management in these architectures are becoming crucial steps towards minimizing the overall system energy consumption while maximizing achievable application performance. Heterogeneous computing platforms are particularly well-suited to execute mixed-criticality workloads as different types of cores can be utilized to better match specific criticality requirements of different type of tasks.

In addition to multiprocessing and heterogeneous computing, a new design paradigm has emerged to further help minimize energy in contemporary chip designs, called ***n**ear-**t**hreshold **c**omputing* (**NTC**) [93] [94] [95] [96] [97] [98]. In NTC, the supply voltage is set just slightly higher than threshold voltage, and execution at this NTC mode achieves several times better energy-efficiency than conventional ***s**uper-**t**hreshold **c**omputing* (**STC**) [96] operation modes. NTC is thus a very effective strategy to minimize energy for energy-constrained embedded systems. However, as NTC mode operation typically sacrifices performance in favor of energy-efficiency, it is not straightforward to use it for mixed criticality real-time embedded systems with timing constraints.

Based on the above observations, *there is clearly a critical need to explore the design and management of STC/NTC capable heterogeneous multicore platforms powered by solar energy*

harvesting and running mixed-criticality workloads, to optimize cost, performance and energy efficiency of such systems. In this chapter, we propose a novel ***m**ixed-**c**riticality **s**cheduling **f**ramework* (**McSF**), that for the first time addresses the problem of allocating and scheduling workloads with different degrees of criticality on a heterogeneous multicore embedded system powered by energy harvesting and supporting NTC operation. Our framework employs NTC for *throughput-centric tasks* with loose timing constraints and a high ***d**egree **o**f **p**arallelism* (**DoP**), maintaining their computation throughput by executing their threads concurrently on many cores in an energy-efficient manner. By improving the energy-efficiency for throughput-centric tasks, more energy budget becomes available for *timing-centric tasks*, which are allocated with awareness of harvested energy fluctuations. The novel contributions of our work can be summarized as follows:

- Unlike any prior work, we formulate and solve the challenging problem of scheduling mixed-criticality, real-time applications on heterogeneous energy-harvesting embedded system platforms;
- The hybrid mapping and scheduling framework from last chapter is adopted to offload scheduling complexity of timing-centric task graphs to a comprehensive design-time methodology so that only lightweight adjustments are required at run-time (e.g., selecting among a small set of schedule templates, core operation modes, and task DoPs) to cope with changing energy harvesting scenarios over time;
- For efficient execution of throughput-centric tasks, we utilize ***n**ear-**t**hreshold **c**omputing* (**NTC**) on several small cores to maintain high throughput levels without sacrificing energy efficiency of the computation;

- A new energy-aware priority metric, *timing intensity-aware penalty density*, is proposed to dynamically measure the importance of instances of different task criticality types within a mixed-criticality workload.

4.2. RELATED WORK

Several prior efforts have explored workload scheduling for embedded systems with solar energy harvesting, as discussed in Section 1.3. However, *none of those prior studies on scheduling for embedded systems with solar energy harvesting consider the scheduling problem for heterogeneous multicore systems, utilize the NTC execution paradigm, or support mixed-criticality workloads, as done in this chapter.*

The high energy-efficiency achievable with ***n**ear-**t**hreshold **c**omputing* (**NTC**) and its design challenges are discussed in [94]. Fick et al. [94] applied NTC to address the power density problem that is crucial for 3D-stacked chips. As NTC systems tend to be more sensitive to process variations with their lower supply voltage, a few recent works propose novel management techniques for NTC to alleviate the performance impact of process variations [96] [95] [97]. More recently, Karpuzcu et al. proposed Accordion, a framework that executes workloads with adjustable problem sizes and fault resilience on NTC-enabled cores [98]. Chen et al. [99] studied the impact of NTC on architectural design of processors by analyzing resulting shifts in performance bottlenecks. But to the best of our knowledge, *no prior work has addressed the scheduling problem for NTC-enabled cores powered by energy harvesting. Moreover prior work has also not considered allocation of mixed criticality workloads on heterogeneous NTC-capable platforms.*

Mixed-criticality workloads are becoming pervasive in many embedded systems today. These workloads consist of applications with different timing or reliability requirements. Systems

designed to support such workloads are often referred to as mixed-criticality platforms. The problem of managing mixed-criticality workload on a single physical platform has attracted a lot of attention in recent years. An early work by Vestal studied schedulability analysis and preemptive fixed priority scheduling for tasks with different criticalities [100]. Mollison et al. brought this problem to multicore systems by proposing a global mixed-criticality scheduling algorithm that can redistribute slack among tasks while maintain isolation for tasks of different criticality levels [101]. Giannopoulou et al. proposed a time-triggered mixed-criticality scheduling approach with barrier synchronization to resolve resource sharing conflict between applications with different criticality levels [102]. Saraswat et al. studied the topic of fault-tolerance for mixed-critical systems [103]. Their proposed framework tackles soft errors using checkpointing-based rollback recovery and tolerates permanent core failures by task migration. Huang et al. studied fault-tolerant mixed-criticality scheduling in the presence of transient faults in the system to provide safety guarantees to tasks with different criticality levels according to established safety standards [104]. The applicability of the proposed scheduling technique was verified for a ***f****light* ***m****anagement* ***s****ystem* (**FMS**) application. Huang et al. also suggested a "run and be safe" strategy that boosts processor frequency temporarily to satisfy timing requirements of critical tasks without degrading service for other tasks. [105] Recently several works have also focused on mapping/partitioning of mixed-criticality applications on multi-core architectures [106] [107] [108]. However, none of these works consider heterogeneous multicore processors as the target platform for mixed-criticality scheduling. Tamas-Selicean and Pop [109] explored optimization for mixed-criticality real-time applications on a distributed heterogeneous node architecture, but not for heterogeneous multicores integrated on a single processor chip. In [110], although heterogeneous multicore processors are initially considered as the hardware platform, the platform

is virtualized to behave as a ***s**ymmetric **m**ulti-**p**rocessor* (**SMP**). Craeynest et al. proposed the ***p**erformance **i**mpact **e**stimation* (**PIE**) scheduling and allocation framework for thread scheduling in single-ISA heterogeneous systems [111]. However, it did not consider applications with mixed-criticality constraints. *Unlike any of these research efforts, this paper is the first to specifically address the mixed-criticality scheduling problem for a unique platform that consists of a heterogeneous multiprocessor powered by solar energy harvesting.*

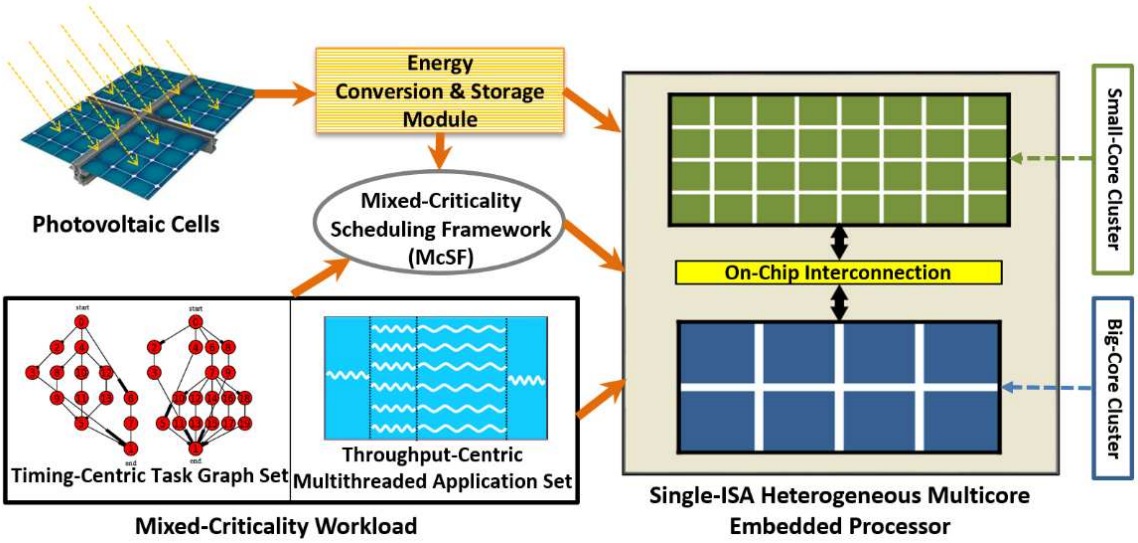

Figure 45 Overview of the Proposed Harvesting-Aware McSF Framework with A Mixed-Criticality Workload and A Single-ISA Heterogeneous Multicore Embedded System

4.3. PROBLEM FORMULATION

Figure 45 shows an overview of our system model that consists of a mixed-criticality workload, single-ISA heterogeneous multicore processor with NTC operation mode capability, an energy harvesting/storage/conversion module, and our ***m**ixed-**c**riticality **s**cheduling **f**ramework* (**McSF**). In the following subsections we describe components and assumptions of our system model before presenting our problem objective.

4.3.1. MIXED-CRITICALITY WORKLOAD MODEL

We differentiate the criticalities of real-time tasks based on the widely applied *(m,k)* model proposed by Hamdaoui et al. [91] and the *task miss penalty* for each task. A task in a system with an *(m,k)* deadline needs to finish at least *m* task instances out of each *k* consecutive instances to avoid system performance degradation. Every task has a user-defined *miss penalty* that is applied to the system whenever an *(m,k)* deadline miss is detected. Our mixed-criticality workload is composed of tasks classified into two categories: the first is *timing-centric* real-time tasks with *(1,1)-firm* deadline constraints; the other is a set of *throughput-centric* tasks with *(m,k)-soft* deadline constraints. The criticalities of tasks of both types can be compared based on combinations of their miss penalties and *(m,k)* constraints.

Timing-centric workloads represent lightweight real-time tasks in the application domain of control, sensing, communication, etc., that require a response before a specified deadline. We assume that these workloads come with highly customized and fixed ***d****egree* ***o****f* ***p****arallelism* (**DoP**) adapted for efficient scheduling and, thus, can be best modeled as periodic task graphs [86]. *Throughput-centric* workloads represent applications in the domain of image processing, data mining, etc., that can tolerate some delay between samples. We model these workloads as barrier-synchronized multithreaded applications [112] [113] with flexible DoP. Even though timing constraints for these workloads are less stringent, they require more computing resources and support high degrees of parallelism, making it essential to exploit parallelism in order to achieve high throughput.

In the rest of this chapter, we refer to these two types of workloads as *timing-centric task graphs* and *throughput-centric multithreaded applications*, respectively. Table 9 summarizes the differences between these two types of workloads.

Table 9 Characteristics of Mixed-Criticality Workloads

Criticality Type	Timing-Centric	Throughput-Centric
Structure Model	task graphs	multithreaded applications
Parallelism	highly customized	barrier-synchronized
Execution Time	few seconds	few minutes
Period	tens of seconds	tens of minutes
Deadline Model	*(1,1)-firm*	*(m, k)-soft*
Execution Rate	related to period	relate to *(m, k)* and period

4.3.2. HETEROGENEOUS MULTICORE COMPUTING PLATFORM

We consider a single-ISA heterogeneous multicore platform to service mixed-criticality workloads. Similar to ARM's big.LITTLE [23], our platform combines one cluster of big cores and one cluster of small cores. In our work, both types of cores (big, small) are based on the x86 instruction set architecture. The big-core-cluster has several high-performance out-of-order cores with per-core DVFS capability [114] that allows execution at several discrete frequency-voltage levels. The small-core-cluster has several power-efficient in-order cores, all of which are clocked with uniform frequency in the NTC region to maximize energy-efficiency. The high performance big-core-cluster is mainly, but not exclusively, utilized to execute timing-centric tasks graphs, while the small-core-cluster executes parallel phases for throughput-centric multithreaded applications.

4.3.3. ENERGY HARVESTING, STORAGE, AND BUDGETING

Similar to pervious chapters, a **p**hoto**v**oltaic (**PV**) system is used as the power source for our multicore embedded system, converting ambient solar energy into electric power. Naturally, the amount of harvested power varies over time due to changing environmental conditions. To cope with the unstable nature of the solar energy source, we assume an energy harvesting subsystem with ***m****aximum* ***p****ower* ***p****oint* ***t****racking* (**MPPT**) to extract the maximum amount of energy possible from the PV system [12] and a hybrid supercapacitor-battery storage to bridge the PV system with our embedded system efficiently [47]. We adopted the hybrid supercapacitor-battery storage design proposed in Chapter 2 that combines supercapacitors and batteries to support both higher-capacity energy storage and lower-overhead energy conversion than a battery-only or a supercapacitor-only solution. We assume that our run-time scheduler can cooperate with this subsystem to inquire about the energy available in storage.

As solar harvesting power can vary dramatically within a very short period of time, it is important to filter out the *noise* from incoming power so that scheduling decisions can be made and executed based on a stable and reliable energy supply. Thus, we use the semi-dynamic energy budget assignment scheme from Chapter 3 (see Figure 46), which partitions time into schedule windows of identical length, the least common multiple of all timing-centric task graphs' periods. Then the energy harvested within each schedule window is used as the energy budget for the next schedule window. Although utilization of harvested energy is delayed for a short period of time in this scheme, it provides the run-time scheduler with a known and stable energy budget at the beginning of each window, making it easier to split the energy budget between timing-centric and throughput-centric workloads.

4.3.4. PROBLEM OBJECTIVE

As solar energy harvesting does not guarantee energy sufficiency, our system is positioned as a soft real-time system that ensures best-effort operation adapted to a given level of energy supply available at run-time. The main objective is to allocate and schedule mixed-criticality workloads composed of multiple timing-centric task graphs and throughput-centric multithreaded applications running simultaneously at run-time, such that total miss penalty for the entire system is minimized, under a varying and unpredictable harvested energy budget over time.

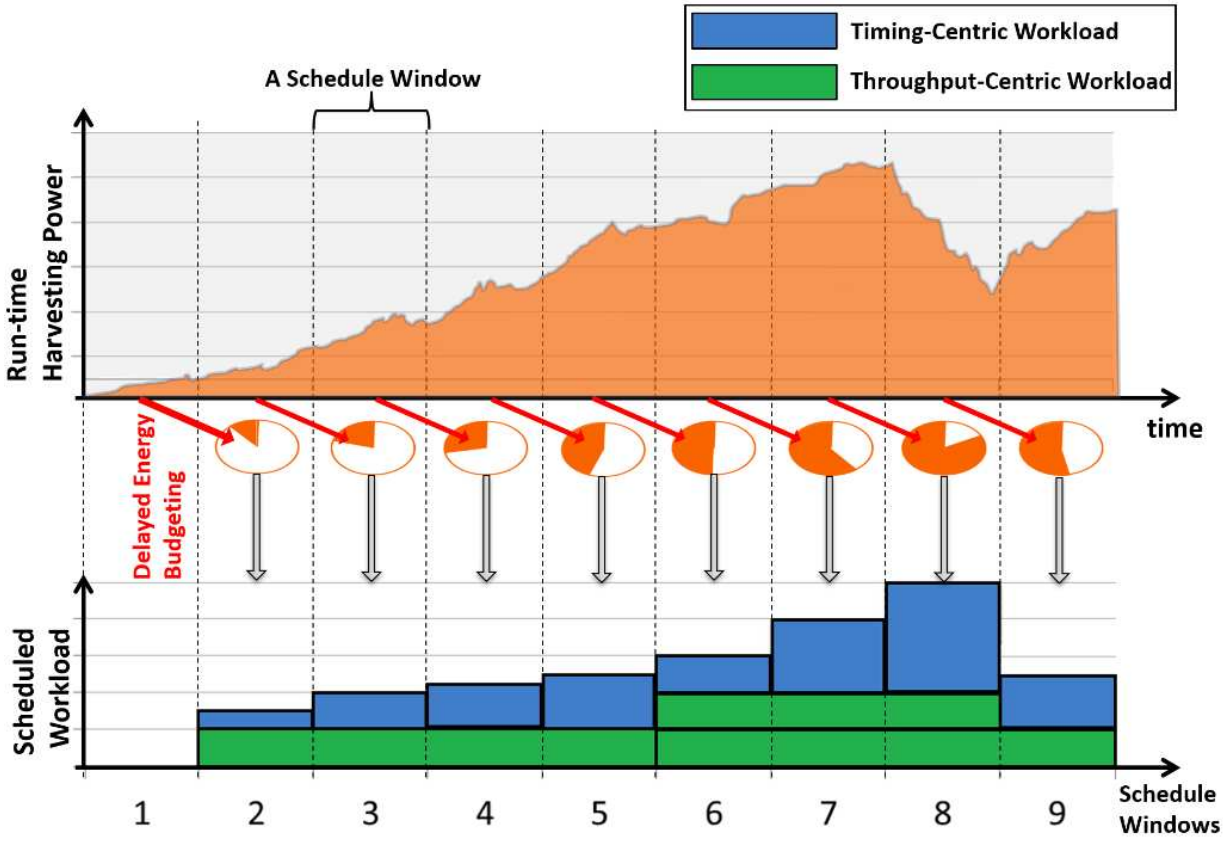

Figure 46 Illustration of Energy Budgeting and Execution Scheduling Across Schedule Windows over Time

4.4. SEMI-DYNAMIC FRAMEWORK FOR MIXED-CRITICALITY SCHEDULING

In this section, we give a brief overview of our semi-dynamic ***m**ixed-**c**riticality **s**cheduling **f**ramework* (**McSF**), which consists of both design-time and run-time components.

As illustrated in Figure 46, for each schedule window, our run-time scheduler dispatches a mix of timing-centric and throughput-centric workloads for execution, given the available energy budget and computing resources. At the top level, our scheduler intelligently sets a balanced distribution of energy budget between the two types of workloads while aiming to minimize overall system miss penalty. Due to the different characteristics and needs of these two types of workloads, each type of workload is scheduled with a specifically designed approach, as discussed next.

Timing-centric task graph workloads in a schedule window can be executed without considering other schedule windows, as the length of a schedule window is the least common multiple of their periods. The general problem of scheduling a task graph under optimization goals and constraints is known to be NP-complete [64]. Thus our scheduling scheme for timing-centric task graphs is designed to offload their scheduling complexity to design-time by offline generation of schedule templates that can be quickly selected for each schedule window at run-time based on the energy budget and cores made available for them after top-level resource distribution. In contrast, instances of throughput-centric multithreaded applications require execution times that can span multiple schedule windows, and thus their execution has to be scheduled dynamically. However, as the execution phases of throughput-centric multithread applications are barrier-synchronized, their scheduling complexity is much lower than that of timing-centric task graphs.

The following describes our *run-time heuristic* for penalty-aware workload filtering and scheduling for mixed-criticality workloads in detail.

4.5. RUN-TIME MIXED-CRITICALITY SCHEDULING

In this section we describe our run-time mixed-criticality scheduling heuristic for scheduling timing-centric task graphs and throughput-centric multithreaded applications to minimize total miss penalty in the system. First, we define a priority metric to represent the impact of each task instance on system miss penalty with consideration of *(m,k)* soft deadline constraints. Then we propose a heuristic to dynamically select and schedule high-priority instances of timing-centric and throughput-centric workloads.

4.5.1. SOFT DEADLINE-AWARE PRIORITY METRIC

As we consider best-effort execution under insufficient solar energy harvesting conditions, it is necessary to dynamically rank priorities of instances of both timing-centric task graphs and throughput-centric multithreaded applications to compare their impact on system miss penalty per unit energy. Based on this guideline, we define a *penalty density* metric based on miss penalty, energy requirement, and timing intensity of a task instance, as shown below:

$$penalty\ density = \frac{miss\ penalty \times timing\ intensity}{energy\ requirement} \tag{34}$$

Among the three components, *miss penalty* of each instance is user defined and assumed to be known at design-time and *energy requirement* can be obtained by profiling applications under different frequency levels. However the *timing intensity* of an instance can change dynamically at run-time based on its *(m,k)* constraint and finish/miss history of previous instances. Hamdaoui et al. have previously proposed a *distance-to-failure* metric to characterize timing intensity of task instances [91]. However, that metric only considers the next nearest instance failure in the worst case while we want to consider all upcoming instances affected by recent execution history to

enable minimization of overall system miss penalty. Thus in this chapter we propose a more comprehensive way to characterize timing intensity of a task instance:

$$timing\ intensity = \sum_{p=0}^{k-1} \frac{m - m'_p}{(k-p)^2}\ , m \geq 1, k \geq 1\ and\ m < k \qquad (35)$$

where, m_p' is the total number of deadlines met (instances finished) in the last p periods, and the values of m and k are based on the user-defined *(m,k)* constraint of the task instance. We refer to every k instances as an *evaluation window*. A finish or miss of an upcoming task instance affects the results for the k upcoming evaluation windows. The *timing intensity* of an upcoming instance is essentially the accumulation of its importance factors to these k evaluation windows. For an evaluation window consisting of p previous instances and $k - p$ future instances, as m_p' instances have already finished, m – m_p' out of $k - p$ upcoming task instances should be finished to avoid miss penalty, resulting in a finish rate requirement of $(m - m'_p)/(k-p)$. As the upcoming instance is only one of the future $k - p$ instances to contribute to this finish rate, we divide finish rate by $k - p$ to get $(m - m'_p)/(k-p)^2$ as the importance factor. This definition also applies to task graphs with *(1, 1)-firm* deadlines, which is a special case with $m= 1$, $k=1$, $p=0$, $m_p'=0$ that always results in instance timing intensity of 1.

Figure 47 shows an example of a *(2,5)-soft* constraint workload execution under three different scenarios. To calculate timing intensity of the upcoming instance in case *(a)*, 5 (k=5) evaluation windows are involved. For the first evaluation window, as 2 instances have already finished, 0 out of 1 instances in the future are require to finish, resulting in an importance factor of $0/1^2$. For the fourth evaluation window, only 1 instance has already finished. Thus 1 additional instance should be finished in put of 4 future instances, resulting in an importance factor of $1/4^2$. In all, the upcoming instance in case *(a)* has timing intensity of 0.143, which is calculated by

accumulating importance factors of all involved evaluation windows. Case *(b)* also has 2 out of 4 previous instances finished, as in case *(a)*. However, the first finished instance only affects the importance factor for the first evaluation window. Consequently, the other 4 evaluation windows all have higher importance factors compared to case *(a),* causing the timing intensity of the upcoming instance to be much higher (0.504). Thus, for previously finished instances, not only their number but also their distribution affects timing intensity of the upcoming instance. Case *(c)* shows that the instance with soft-deadline constraint can have intensity greater than 1, as it not only must be finished to avoid miss penalty in the current period, similar to *(1,1)-firm* instances, but it also affects timing intensities of future instances.

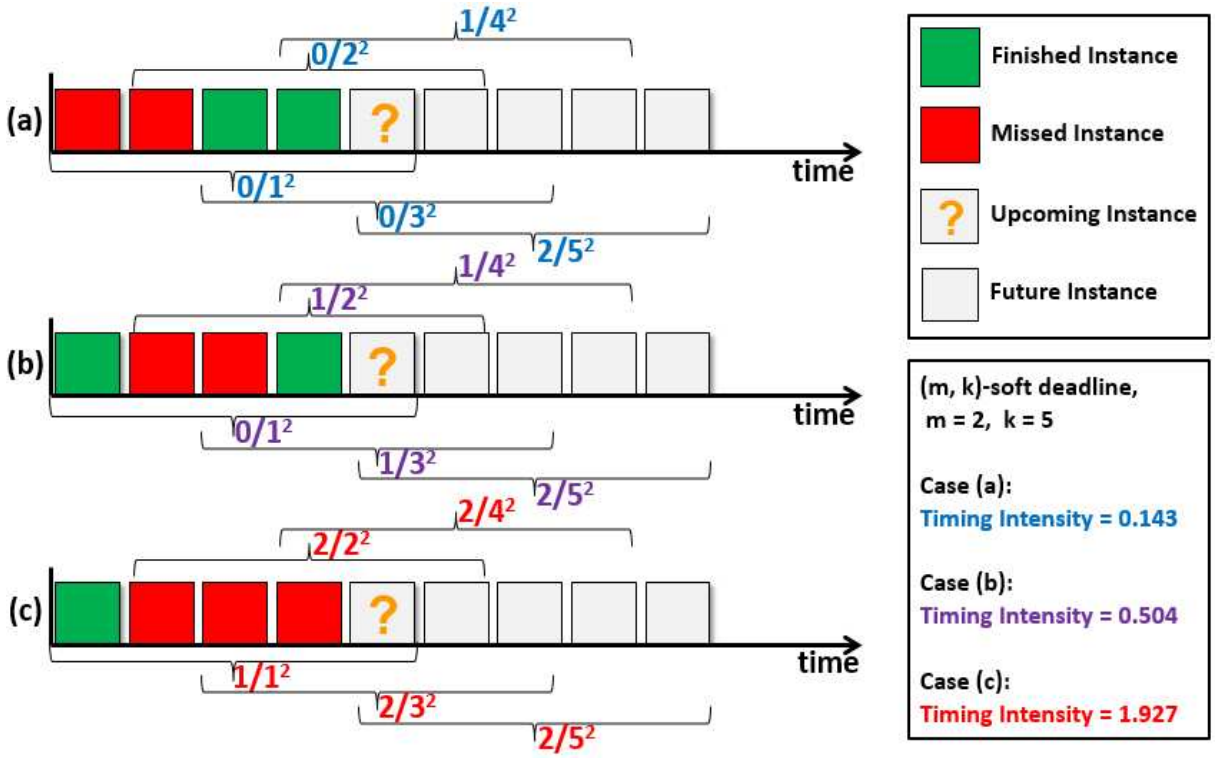

Figure 47 Illustration of Timing Intensity for *(2, 5)-soft* Deadline Case

4.5.2. DYNAMIC WORKLOAD FILTERING AND BALANCING

Guided by our proposed timing *intensity-aware penalty density metric*, our workload filtering and scheduling heuristic perform resource allocation for both timing-centric task graphs and throughput-centric multithreaded applications based on the energy budget assigned or predicted in the current and future schedule windows, with the goal of minimizing overall system miss penalty. The heuristic is shown in Algorithm 10.

Algorithm 10 Dynamic Workload Filtering and Scheduling

Input:
(i) *app_pool*, multithreaded application instances arrived or in execution
(ii) *EGY_BGT*, energy harvested and unused during last schedule window
(iii) EGY_PRD_w, harvesting energy prediction for next w schedule windows
(iv) Set of offline-generated task graph scheduling templates optimized for different number of big cores and energy budget levels (see Section 4.6)

Output:
(i) Execution schedule for multithreaded applications
(ii) Selected schedule template for task graphs

Triggered at the beginning of each schedule window:
1. update priorities (penalty densities) of all instances in app_pool
2. **while** there are unscheduled instances and remaining energy budget:
3. in app_pool, select instance, app, with highest priority, $density_{app}$
4. find task graph schedule template with more workload, next_temp
5. $density_{tg} \leftarrow \Delta penalty/\Delta energy$ †
6. **if** $density_{app} < density_{tg}$ **and** ENG_BGT is sufficient:
7. use next_temp as the selected schedule template for task graphs
8. **else if** $density_{app} < density_{tg}$ and EGY_BGT, EGY_PRD_w are sufficient:
9. start/resume execution of app with as even as possible schedule
10. remove app from app_pool
11. **if** sequential phase detected in this schedule window:
12. steal one big core from timing-centric task graphs
13. re-select task graph schedule template for one less core
14. **for** app remaining **in** app_pool:
15. drop and record instance miss

† Δ values are based on comparison between current and found template from step 4

The heuristic progressively compares and accepts instances of timing-centric task graph applications and throughput-centric multithreaded applications at the beginning of each schedule

window. Dynamic instance priorities (penalty densities) of these two types of applications are assigned in different manners: *(i) For throughput-centric multithreaded application instances*, priorities are updated individually at the beginning of each schedule window (step 1). The priority of a new task instance will typically be different from previous ones as timing intensity keeps changing with respect to the *(m,k)* constraint (Section 4.5.1). For an instance already in execution, priority will increase because the more energy it has already consumed, the less energy it requires to finish. This mechanism encourages the heuristic to resume application instances in progress so that the effort already invested in execution can be preserved; *(ii) For timing-centric task graph instances*, as their *(m,k)* timing intensity is always equal to 1, their dynamic priorities only change with varying energy requirements for different frequencies assigned in different schedule templates. Unlike the case of multithreaded applications, here our heuristic considers total priority of extra instances accepted when considering the use of another schedule template, which is deduced by comparing the new template's miss penalty and energy requirements to those of the current one (steps 4, 5). In each iteration of the while loop, priorities of candidate instances from timing-centric task graphs and throughput-centric multithreaded applications are compared to decide which ones to accept for execution (steps 6 – 13).

During workload filtering, the execution schedule of accepted task graph and multithreaded application instances are also decided. For accepted multithreaded application instances, the execution is dynamically deduced by a scheduling method called *as-even-as-possible*, which attempts to evenly distribute execution effort over time by starting execution of an instance on arrival and finishing it before its deadline (step 9). In this schedule, parallelizable phases of an application instance are executed on the small-core-cluster clocked at an energy-efficient frequency level in the NTC region, while sequential phases are executed by stealing big cores from

task graph instances (steps 11-13), leaving more time to spread execution effort of parallelizable phases. *As-even-as-possible* execution scheduling improves energy-efficiency of the system in two ways: *(i)* an even execution scheduling minimizes the number of small cores required for each parallel phase, reducing multithreading energy-overhead which increases with thread count [115]; *(ii)* as this scheduling method distributes energy consumption of multithreaded applications more evenly across multiple schedule windows, timing-centric task graphs in these windows also tend to get more even energy budgets among them, resulting in better overall energy-efficiency. An even execution schedule has been shown to result in high energy efficiency for systems with DVFS capability [116]. For the same reason, our scheduler does not consider shutting down cores as energy saved will not justify the efficiency loss of the resulting uneven schedule. When a sequential phase of a multithreaded application steals a big core, a new schedule template for timing-centric task graphs is selected to execute with one less core available. Lastly, an instance miss is recorded for application instances that remain unaccepted (steps 14, 15).

4.6. EXPERIMENTAL RESULTS

4.6.1. EXPERIMENT SETUP

Our experiments use real-world energy harvesting profiles based on historical weather data provided by the ***M**easurement and **I**nstrumentation **D**ata **C**enter* (**MIDC**) of the ***N**ational **R**enewable **E**nergy **L**aboratory* (**NREL**) [60]. Again, we evaluate system performance over a span of 750 minutes, from 6:00AM to 6:30PM in a day. We assume peak energy harvesting power to be equal to maximum power required by system to execute all workload instances.

For timing-centric task graph applications, we select examples in the domain of *networking*, *telecom*, and *auto-industry* from the ***E**mbedded **S**ystem **S**ynthesis Benchmark **S**uite* (**E3S**) [86].

Task graphs are assigned with periods ranging from 10 to 60 seconds. For throughput-centric multithreaded applications, we select a set of barrier-synchronized parallel applications, including *fft*, *cholesky*, *bodytrack*, *vips*, and *blackscholes*, from SPLASH-2 [112] and PARSEC [113] benchmark suites, which have different periods and *(m, k)-soft* constraints assigned.

Table 10 Configuration of Heterogeneous Multicore Processor

Architectural Parameters		
Core Types	**Big Cores**	**Small Cores**
Execution	Out-of-Order	In-Order
Issue Width	4	2
Reorder Buffer Size	128	N/A
Cache	64KB, 4-way	16KB, direct
Core Area	15.7 mm^2	4 mm^2
Cluster Parameters		
Cluster Type	**Big-Core-Cluster**	**Small-Core-Cluster**
Core Count	8	32
Frequency Control	Per-Core DVFS	Uniform Frequency
f, V_{dd} **Range**	0.5~1.2GHz, 0.4~1 V	f^{nth}, V_{dd}^{nth}
Technology Parameters		
Technology Node	22 nm	
V_{th}	0.289 V	
V_{dd}^{nth}, f^{nth}	0.4 V, 500 MHz	

To acquire power and performance metrics for mixed-criticality workloads on different types of cores, we use Sniper [117], an x86 multicore simulator, and the McPAT [118] power model extended to support V_{dd} in the NTC region for the 22 nm node. Table 10 shows the configuration of our platform with big-core-clusters and small-core-clusters. For intra-cluster transfers, a 2D-mesh ***n**etwork-**o**n-**c**hip* (**NoC**) and XY routing over conflict-free TDMA virtual channels is assumed. For inter-cluster communication, we assume delay in the range of hundreds of milliseconds to cross clusters.

We assume threshold voltage, V_{th}, of 0.289V for the 22nm technology node [119]. Based on power simulation results over multiple runs, we set the NTC supply voltage, V_{dd}^{nth} to be $0.4V$, which not only achieves high energy-efficiency but also keeps a safe margin with V_{th} to avoid errors due to the impact of process variations [95]. According to architectural level delay analysis result for CMOS processors in [120] with assumption of slightly shorter critical path for our small cores compared to Intel Atom processors [121], we set NTC operation frequency, f^{nth}, to be 500MHz.

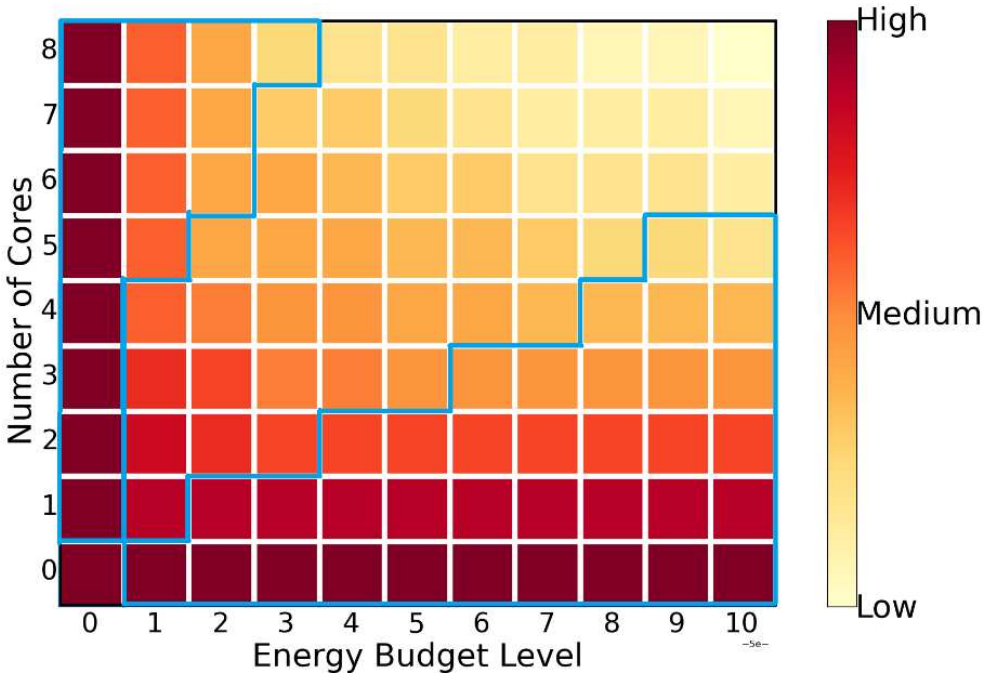

Figure 48 Miss Penalties for Generated Schedule Templates

4.6.2. DESIGN-TIME TEMPLATE GENERATION ANALYSIS

Our ***m**ixed-**c**riticality **s**cheduling **f**ramework* (**McSF**) executes timing-centric task graphs based on schedule templates generated at design-time using the ***a**nalysis-based **t**emplate **g**eneration method* (**ATG**) proposed in Chapter 3. The per-schedule-window miss penalties of the

generated template set are shown in Figure 48, which shows decreasing penalty when more energy budget and cores are made available for an execution schedule. It should be noted that some templates are ignored in our scheduling, e.g., those highlighted in the upper-left and bottom-right regions of Figure 48 enclosed by blue lines. For example, for the 2 core case, looking at the highlighted region on the bottom-right, increasing the energy budget level beyond 3 does not reduce miss penalty. Similarly, for energy budget level 2, increasing the number of cores beyond 5 does not improve miss penalty. Thus templates in these two regions can be safely ignored.

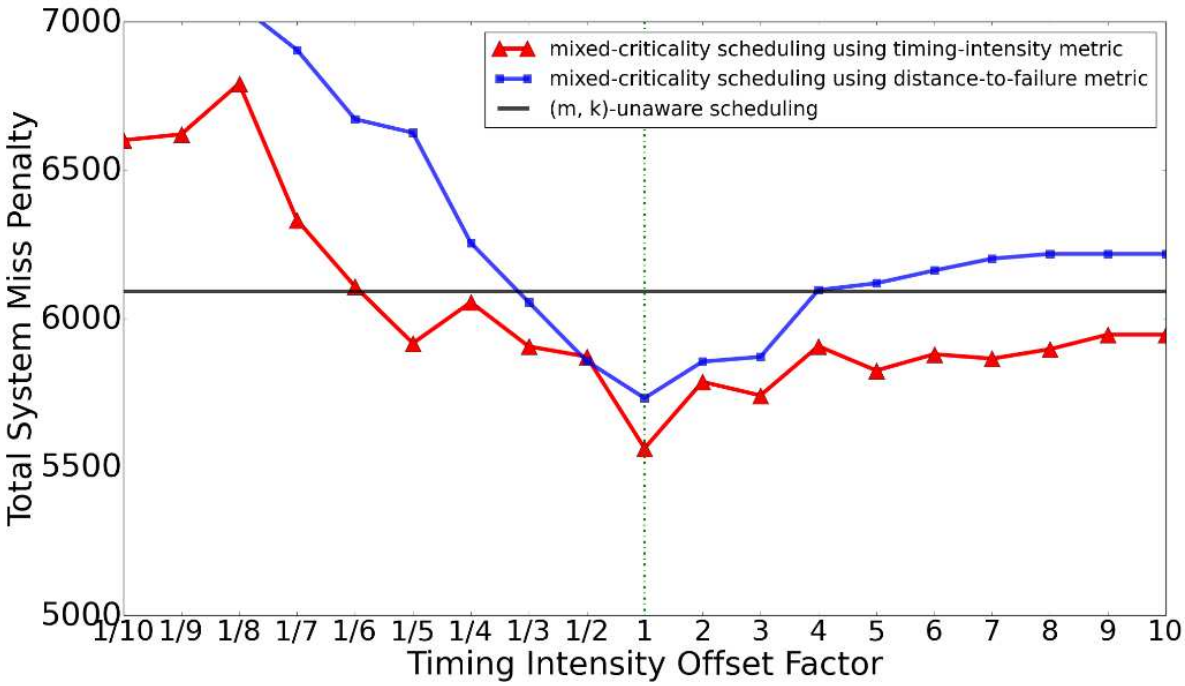

Figure 49 System Miss Penalties under Different Intensity Scale Factors

4.6.3. TIMING INTENSITY METRIC EVALUATION

We tested if our proposed timing intensity metric (see Equation (35) of Section 4.5.1) accurately characterizes the importance of application instances with respect to the *(m,k)* constraint in a mixed-criticality workload. For this purpose, we offset timing intensity calculated for instances

of throughput-centric multithreaded applications by multiplying them with a factor ranging from 1/10 to 10, while the timing intensity for timing-centric task graph instances was fixed to 1.

The results in Figure 49 show that keeping the original calculated timing intensity minimizes overall system miss penalty, while offsetting timing intensity to higher or lower values leads to more miss penalties. Thus our defined timing intensity metric can accurately evaluate importance of instances to achieve the best balance between throughput-centric and timing-centric tasks to minimize miss penalty of the entire mixed-criticality workload. We also compared our timing intensity metric with the *distance-to-failure* metric proposed in [91], which also finds its peak when no offset is applied (Figure 49). We found that the *distance-to-failure* metric results in up to 9.5% higher miss penalties, compared to our timing intensity-based priority assignment method, as the distance-to-failure metric only considers the next nearest timing failure in the worst case. Besides, both metric evaluation methods outperform the *(m,k)*-unaware scheduling method that assumes firm deadlines for all application instances (see black dash line in Figure 49).

4.6.4. MIXED-CRITICALITY SCHEDULING PERFORMANCE EVALUATION

As ours is the first framework to address the scheduling and allocation problem for mixed-criticality heterogeneous systems powered by energy harvesting, there is no prior work to directly compare the overall system performance against. However, we did adapt the ***p**erformance **i**mpact **e**stimation* (**PIE**) methodology as an exemplar state-of-art thread scheduling technique for single-ISA heterogeneous systems from [111] (even though it does not support energy harvesting). To fit into the experimental setup of this paper, our version of *PIE* estimates the performance benefit of mapping each phase in throughput-centric applications to big cores and the scheduler dynamically

selects one phase with the most benefit to share bigger cores with timing-centric task graphs for each schedule window.

We additionally compare the performance of our proposed ***m**ixed-**c**riticality **s**cheduling **f**ramework* (**McSF**) across four different setups: 1) *B8-S32*, the default configuration with 8 big cores and 32 small cores, which adapts the *moving average* solar energy prediction method used in [41]; 2) *Perfect-Pred*, a setup with identical core configuration as the default one, but with the assumption of perfect energy harvesting prediction; 3) *B8-B32*, a configuration that replaces the default 32 small cores with 32 big ones; and 4) *B8-B8*, a configuration that replaces the default 32 small cores with 8 big cores, to keep overall area footprint the same as *B8-S32* (Table 10).

Figure 50 shows the results of our comparison study. For throughput-centric applications the total miss rate (*throughput-centric: all*) represents all the instances that are dropped. However, because of the (*m,k*)-soft deadline constraint in these applications, some dropped instances do not violate the constraint. Therefore the effective miss rate (*throughput-centric: (m, k)-only*) is much lower.

The default *B8-S32* configuration only suffers slight increase in system miss penalty compared to *Perfect-Pred* that has ideal energy prediction, showing the ability of McSF to mitigate the performance impact of energy harvesting mispredictions. Compared to *Perfect-Pred*, *B8-S32* has higher miss rate for timing-centric tasks graph instances and lower miss rate for multithreaded application instances. This is because *B8-S32* accepts higher than optimal multithreaded application instances, without awareness of hard-to-predict instantaneous drops in harvesting power. Then our dynamic workload filtering framework in Section 4.5.2 allocates fewer resources to timing-centric task graphs to sustain the energy supply for those extra throughput-centric instances already in execution to minimize energy wasted due to misprediction. As a result, miss

penalty increases slightly because the balance between the two types of workloads is affected during this process.

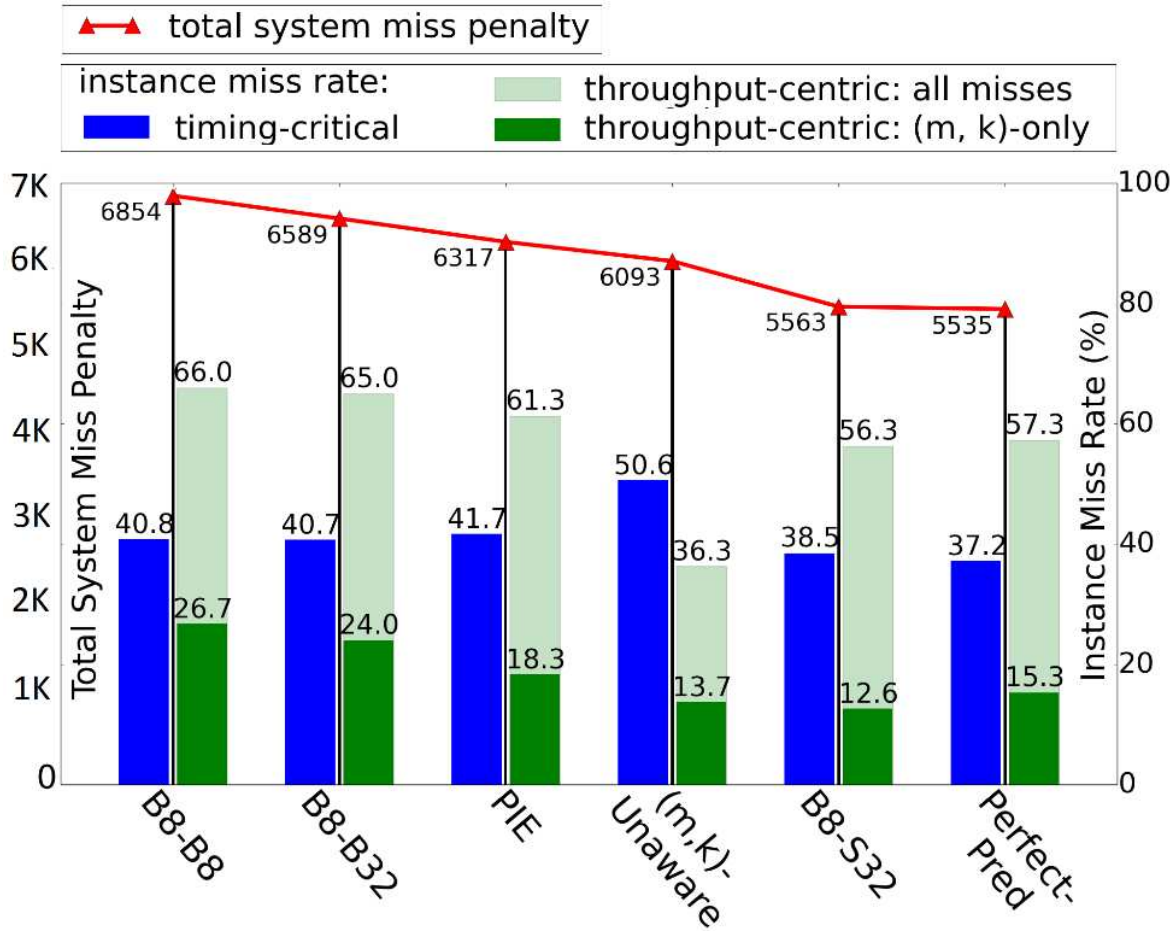

Figure 50 Miss Penalties and Instance Miss Rates across Configurations

For the *(m, k)-Unaware* setup, which utilizes the same core-configuration as *B8-S32* but has no awareness of *(m, k)* constraints, the result shows much lower total miss rate for throughput-centric instances as it considers all instances as necessary for penalty avoidance. However, the actual *(m, k)*-miss rate increases as *(m, k)-Unaware* allocates energy to instances that are less important for *(m, k)* constraints. Besides, it also leads to higher miss rate for timing-centric task

graphs as the balance between the two types of workload is notably affected. Thus *(m, k)-Unaware* has higher overall system miss penalty compared to *B8-S32*.

Comparing *B8-S32* with *B8-B32*, it can be seen that although *B8-B32* provides better computing capability, it leads to much higher overall miss penalty due to a decrease in energy efficiency. On average, big cores bring performance speedup of approximately 3×, with an average jump of 7× in power consumption, ending up with a 2× degradation in energy efficiency. Moreover *B8-B32* also has a much higher area footprint than *B8-S32*, given that the area of big cores is close to 4× that of small cores. *B8-B8* is a multicore configuration with the same chip area footprint as *B8-S32*. But *B8-B8* suffers even higher miss penalty than *B8-B32,* as it not only has lower energy efficiency than *B8-B32* but also possesses lower computation throughput than *B8-B32*. *B8-S32* outperforms *B8-B8* by 23.2% miss penalty reduction, *highlighting the importance of core heterogeneity to improve energy-efficiency and performance in multicore computing platforms*.

Lastly, Figure 50 also shows a comparison with the ***p**erformance **i**mpact **e**stimation* (**PIE**) scheduling and allocation framework from [111] for thread scheduling in single-ISA heterogeneous systems, It can be seen that *PIE* has 13.6% higher miss rate and penalty compared to *B8-S32*, *as it does not focus on energy efficiency but throughput performance, causing more workload to be allocated to big cores for less overall efficiency*.

4.6.5. CHAPTER SUMMARY

In this chapter, we addressed the scheduling problem for single-ISA heterogeneous multicore processors running hybrid mixed-criticality workloads with a limited and fluctuating energy budget provided by solar energy harvesting. We modeled a mixed-criticality workload by combining timing-centric real-time task graphs with firm deadlines and throughput-centric

multithreaded phases with soft deadlines, with different associated miss penalties. We utilized a single-ISA heterogeneous platform design to fulfil requirements for this mixed-criticality workload. To achieve a balance that minimizes overall system miss penalty, we proposed a novel timing intensity estimation method, based on which we can allocate resources dynamically to different types of workload according to energy harvesting conditions. In experiments, our proposed mixed-criticality scheduling framework achieves a 9.5% miss penalty reduction with the proposed timing intensity metric compared to metrics from prior work, a 13.6% performance improvement over a state-of-the-art scheduling approach for single-ISA heterogeneous platforms, and a 23.2% performance benefit from exploiting platform heterogeneity.

5. CONCLUSION AND FUTURE WORK

5.1. RESEARCH CONCLUSION

In this dissertation, we addressed important challenges faced by real-time embedded multicore systems with energy harvesting, by proposing a novel semi-dynamic resource management framework. This framework is designed to cope with run-time variations in harvesting power with optimal low-overhead task scheduling to maximize system throughput and high functionality flexibility to adapt to the changing run-time dynamics. As presented in previous chapters, our proposed semi-dynamic framework utilizes various optimization algorithms such as graph algorithms, linear programming, and custom heuristics to optimize system performance, efficiency, and reliability at run-time and/or design-time. Experimental results for our proposed semi-dynamic framework validate and motivate its deployment in future embedded systems powered by energy harvesting, because this framework demonstrates significant improvement in energy efficiency with extensibility to adapt emerging and increasingly relevant design concerns, such as overheating, transient errors, and aging effect. Therefore, our proposed semi-dynamic framework has the potential to be applied as a general strategy for resource management on systems powered by time-varying energy harvesting.

Our first contribution is *SDA* (Chapter 2), a novel semi-dynamic scheduling algorithm aimed at scheduling periodic independent real-time tasks with awareness of energy harvesting. Its fundamental idea is time-segmentation, which guarantees uniform execution frequency within each scheduling window for higher energy efficiency. Experimental results indicate a significant (up to 70%) improvement in system performance, compared to state-of-the-art algorithms under an identical system setup. We extended SDA to consider support for task drop penalty awareness,

run-time thermal management, core-heterogeneity mitigation, and hybrid energy storage utilization. Moreover, from SDA we derive the design methodology of semi-dynamic resource management, which is the core idea of this dissertation to effectively tackle various problems for managing systems with energy harvesting.

Based on the concept of semi-dynamic resource management, we proposed HyWM (Chapter 3), a hybrid design-time and run-time workload management framework to cope with the complexity of scheduling task graphs with data dependencies and run-time variations in solar radiance, execution time, transient faults, and aging progress. Our experimental results indicated improvements in performance and adaptivity of target systems due to the efficiency and flexibility of our semi-dynamic framework, with up to 23.2% miss rate reduction compared to prior work, 43.6% performance benefits from adaptive run-time workload management compared to a baseline framework with no soft error and slack time handling, and up to 24.5 % expected system lifetime improvement with aging-aware workload allocation compared to aging-agnostic schemes, under stringent energy constraints and varying system conditions at run-time. Therefore, our semi-dynamic framework proves to be a promising and practical solution to transform the future of energy-autonomous embedded computing with boosted scope and efficiency.

Finally, we applied the semi-dynamic framework to address the scheduling of mixed-criticality workloads on single-ISA heterogeneous multicore platform powered by solar energy harvesting (Chapter 4). To achieve a balance between different types of workload that minimizes overall system miss penalty, we proposed a novel timing intensity metric for mixed-criticality tasks, which are utilized to guide resource allocation in the semi-dynamic framework. In experiments, our proposed mixed-criticality scheduling framework achieves a 9.5% miss penalty reduction with the proposed timing intensity metric compared to metrics from prior work, a 13.6%

performance improvement over a state-of-the-art scheduling approach for single-ISA heterogeneous platforms, and a 23.2% performance benefit from exploiting platform heterogeneity.

5.2. FUTURE WORK

Embedded computing powered by solar energy harvesting will continue to face new challenges and opportunities on the path towards a future with pervasive computing, as applications and platforms evolve rapidly. Thus we further envision the following future work directions:

- *Mobile Computing*: Limited battery lifetime is the major factor that affects daily user experience in today's smartphones [122]. With miniaturization of high-efficiency PV cells [123], solar energy harvesting could become the auxiliary or even standalone energy source for future smartphones. The major difference between smart mobile devices and other embedded computing platforms lies in their unique interface between apps and operating systems and their emphasis on user experience such as input delay or interface transition lag. Therefore, for such smart mobile devices, it is necessary to implement a framework seamlessly integrated with the OS [30] to co-optimize energy efficiency and user experience with awareness of energy harvesting. We can also view mobile computing platforms as a type of mixed-criticality system that hosts tasks with very different timing requirements, including user-centric interface rendering tasks, user-centric foreground threads, background system/user level services, and real-time communications tasks. It will be interesting to study the interaction between energy harvesting and these tasks with different timing requirements to optimize user experience and energy efficiency.

- *Nonvolatile Computing*: By adopting nonvolatile registers and nonvolatile SRAM, the emerging nonvolatile processors support *in place system recovery* to enable the seamless transition between different power states of systems with energy harvesting [124]. Besides, as nonvolatile processors do not need a power supply to sustain the memory state, leakage power can be reduced significantly by turning off memory system when possible. However, nonvolatile computing also comes with overheads in terms of energy, area, and performance. Thus comprehensive research from the circuit to the system level for nonvolatile processors is required before we can exploit their full potential for systems powered by energy harvesting [125]. For example, we may develop an efficient sleep/recover scheme that only preserves information necessary for the resumption of system execution, minimizing required footprint of nonvolatile memory for lower overhead in chip area and recovery energy.
- *Approximate Computing*: Approximate computing has recently emerged as a promising approach that relies on systems and applications' tolerance on loss of quality and optimality in the computing results to achieve substantial improvements in energy efficiency [126]. As solar energy harvesting offers no guarantees related to the sufficiency of the energy supply, it is usually used for applications with lax requirements on systems output. Thus, the combination of approximate computing and energy harvesting can be a promising research direction as they share similar design concerns. However, most recent efforts on approximate computing focus on hardware design methodologies for approximate computing platforms [127], which have enabled energy efficiency improvements in general but have failed to provide an approach to trade-off between result accuracy and energy efficiency on-the-fly. For approximate computing systems with energy harvesting,

we believe that in addition to applying approximate hardware platforms, considerations of the software stack and resource management infrastructure are also important. It would be interesting to explore the possibility of dynamic trade-offs between result accuracy and energy efficiency by utilizing the inherent fault-tolerance of certain probabilistic applications such as stochastic optimization algorithms and machine learning procedures, where we can adjust computation load and accuracy without failure of entire applications.

BIBLIOGRAPHY

[1] M. Satyanarayanan, "Pervasive computing: Vision and Challenges," *the IEEE Personal Communications (PC),* vol. 8, no. 4, pp. 10-17, 2001.

[2] T. Simunic, L. Benini and G. De Micheli, "Energy-Efficient Design of Battery-Powered Embedded Systems," *the IEEE Transactions on Very Large Scale Integration Systems (TVLSI),* vol. 9, no. 1, pp. 15-28, 2002.

[3] Wikipedia, "Energy Harvesting," [Online]. Available: https://en.wikipedia.org/wiki/Energy_harvesting. [Accessed June 2015].

[4] Winter Green Research, "Energy Harvesters: Market Shares, Strategies, and Forecasts, Worldwide, 2013 to 2019," 2013.

[5] S. P. Beeby, M. J. Tudor and N. M. White, "Energy Harvesting Vibration Sources for Microsystems Applications," *Measurement Science and Technology,* vol. 17, no. 12, pp. 175-195, 2006.

[6] Google, "Google Trends," [Online]. Available: https://www.google.com/trends/. [Accessed July 2015].

[7] G. Ottman, H. Hofmann, A. Bhatt and G. Lesieutre, "Adaptive Piezoelectric Energy Harvesting Circuit for Wireless Remote Power Supply," *the IEEE Transactions on Power Electronics (TPE),* vol. 17, no. 5, pp. 669-676, 2002.

[8] Micropelt, "TE-Power PROBE," [Online]. Available: http://www.micropelt.com/applications/te_power_probe.php. [Accessed August 2015].

[9] X. Lu and S. H. Yang, "Thermal Energy Harvesting for WSNs," in *the IEEE International Conference on Systems Man and Cybernetics (SMC)*, Istanbul, Turkey, 2010.

[10] J. Carrasco, L. Franquelo, J. Bialasiewicz, E. Galvan, R. Guisado, M. Prats, J. Leon and N. Moreno-Alfonso, "Power-Electronic Systems for the Grid Integration of Renewable Energy Sources: A Survey," *the IEEE Transactions on Industrial Electronics (TIE)*, vol. 53, no. 4, pp. 1002-1016, 2006.

[11] V. Raghunathan, A. Kansal, J. Hsu, J. Friedman and M. Srivastava, "Design Considerations for Solar Energy Harvesting Wireless Embedded Systems," in *the International Symposium on Information Processing in Sensor Networks (IPSN)*, Los Angeles, CA, USA, 2014.

[12] M. S. T. Veerachary and K. Uezato, "Maximum Power Point Tracking of Coupled Inductor Interleaved Boost Converter Supplied PV System," *the IEE Proceedings on Electric Power Applications (EPA)*, vol. 150, no. 1, pp. 71 - 80, 2003.

[13] Wikipedia, "Embedded system," [Online]. Available: https://en.wikipedia.org/wiki/Embedded_system. [Accessed July 2015].

[14] Wikipedia, "Real-Time Computing," [Online]. Available: https://en.wikipedia.org/wiki/Real-time_computing. [Accessed July 2015].

[15] C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in A Hard-Real-Time Environment," *the Journal of the ACM (JACM)*, vol. 20, no. 1, pp. 46-61, 1973.

[16] W. Yuan and K. Nahrstedt, "Energy-Efficient Soft Real-Time CPU Scheduling for Mobile Multimedia Systems," *the ACM SIGOPS Operating Systems Review*, vol. 37, no. 5, pp. 149-163, 2003.

[17] B. Srinivasan, S. Pather, R. Hill, F. Ansari and D. Niehaus, "A Firm Real-Time System Implementation using Commercial Off-the-Shelf Hardware and Free Software," in *the Real-Time Technology and Applications Symposium (RTTAS)*, Denver, CO, USA, 1998.

[18] S. Baruah, J. Gehrke and C. Plaxton, "Fast Scheduling of Periodic Tasks on Multiple Resources," in *the International Parallel Processing Symposium (IPPS)*, Santa Barbara, CA, USA, 1995.

[19] D. A. Patterson and J. L. Hennessy, Computer Organization and Design, Fifth Edition, Morgan Kaufmann, 2013.

[20] K. Khubaib, M. A. Suleman, M. Hashemi, C. Wilkerson and Y. N. Patt, "MorphCore: An Energy-Efficient Microarchitecture for High Performance ILP and High Throughput TLP," in *The IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Vancouver, BC, Canada, 2012.

[21] R. Fish, "Future of Computers - Part 2: The Power Wall," [Online]. Available: http://www.edn.com/design/systems-design/4368858/Future-of-computers--Part-2-The-Power-Wall. [Accessed August 2015].

[22] ARM, "ARM Cortex-A9 Processor.," [Online]. Available: http://www.arm.com/products/processors/cortex-a/cortex-a9.php. [Accessed November 2014].

[23] P. Greenhalgh, "Big.LITTLE Processing with ARM Cortex-A15 & Cortex-A7," ARM, 2011.

[24] Nvidia, "The Benefits of Multiple CPU Cores in Mobile Devices," [Online]. Available: http://www.nvidia.com/content/PDF/tegra_white_papers/Benefits-of-Multi-core-CPUs-in-Mobile-Devices_Ver1.2.pdf. [Accessed June 2015].

[25] S. Bell, B. Edwards, J. Amann, R. Conlin, K. Joyce, V. Leung, J. MacKay, M. Reif, L. B., J. Brown, M. Mattina, C. C. Miao, C. Ramey, D. Wentzlaff, W. Anderson, E. Berger, N. Fairbanks, D. Khan, F. Montenegro, J. Stickney and J. Zook, "TILE64 - Processor: A 64-Core SoC with Mesh Interconnect," in *the International Solid-State Circuits Conference (ISSCC)*, San Francisco, CA, USA, 2008.

[26] R. Kumar, K. Farkas, N. Jouppi, P. Ranganathan and D. Tullsen, "Single-ISA Heterogeneous Multi-Core Architectures: The Potential for Processor Power Reduction," in *the IEEE/ACM International Symposium on Microarchitecture (MICRO)*, San Diego, CA, USA, 2003.

[27] AMD, "The Programmer's Guide to the APU Galaxy," [Online]. Available: http://developer.amd.com/wordpress/media/2013/06/Phil-Rogers-Keynote-FINAL.pdf. [Accessed July 2015].

[28] E. Humenay, D. Tarjan and K. Skadron, "Impact of Process Variations on Multicore Performance Symmetry," in *the Conference on Design, Automation and Test in Europe (DATE)*, San Jose, CA, USA, 2007.

[29] A. Tiwari and J. Torrellas, "Facelift: Hiding and Slowing Down Aging in Multicores," in *The IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Lake Como, Italy, 2008.

[30] P. Pillai and K. G. Shin, "Real-time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems," in *the ACM Symposium on Operating Systems Principles (SOSP)*, Chateau Lake Louise, Banff, Canada, 2001.

[31] T. D. Burd and R. W. Brodersen, "Design Issues for Dynamic Voltage Scaling," in *the International Symposium on Low Power Electronics and Design (ISLPED)*, New York, NY, USA, 2000.

[32] J. Pouwelse, K. Langendoen and H. Sips, "Dynamic Voltage Scaling on A Low-Power Microprocessor," in *the International Conference on Mobile Computing and Networking (MobiCom)*, Rome, Italy, 2001.

[33] L. Benini, A. Bogliolo and G. D. Micheli, "A Survey of Design Techniques for System-Level Dynamic Power Management," *the IEEE Transactions on Very Large Scale Integration Systems (TVLSI)*, vol. 8, no. 3, pp. 299-316, 2000.

[34] H. Aydin, P. M. Alvarez, D. Mossé and R. Melhem, "Dynamic and Aggressive Scheduling Techniques for Power-Aware Real-Time Systems," in *the IEEE Real-Time Systems Symposium (RTSS)*, Washington, DC, USA, 2001.

[35] J. J. Chen, T. W. Kuo, C. L. Yang and K. J. King, "Energy-Efficient Real-Time Task Scheduling with Task Rejection," in *the Conference on Design, Automation and Test in Europe (DATE)*, San Jose, CA, USA, 2007.

[36] C. Li, W. Zhang, C. B. Cho and T. Li, "SolarCore: Solar Energy Driven Multi-Core Architecture Power Management," in *the International Symposium On High Performance Computer Architecture (HPCA)*, San Antonio, TX, USA, 2011.

[37] X. Lin, Y. Wang, D. Zhu, N. Chang and M. Pedram, "Online Fault Detection and Tolerance for Photovoltaic Energy Harvesting Systems," in *the International Conference on Computer-Aided Design (ICCAD)*, San Jose, CA, USA, 2012.

[38] M. Veerachary, T. Senjyu and K. Uezato, "Voltage-Based Maximum Power Point Tracking Control of PV System," *the IEEE Transactions on Aerospace and Electronic Systems (TAES)*, vol. 38, no. 1, pp. 262-270, 2002.

[39] C. Moser, D. Brunelli, L. Thiele and L. Benini, "Lazy Scheduling for Energy Harvesting Sensor Nodes," in *the Conference on Distributed and Parallel Embedded Systems (DIPES)*, Braga, Portugal, 2006.

[40] S. Liu, Q. Qiu and Q. Wu, "Energy Aware Dynamic Voltage and Frequency Selection for Real-time Systems with Energy Harvesting," in *the Conference on Design, Automation and Test in Europe (DATE)*, Munich, Germany, 2008.

[41] S. Liu, J. Lu, Q. Wu and Q. Qiu, "Harvesting-Aware Power Management for Real-Time Systems with Renewable Energy," *the IEEE Transactions on Very Large Scale Integration Systems (TVLSI)*, vol. 20, no. 8, pp. 1473-1486, 2012.

[42] M. Chetto, "Optimal Scheduling for Real-Time Jobs in Energy Harvesting Computing Systems," *The IEEE Transactions on Emerging Topics in Computing (TETC)*, vol. 2, no. 2, pp. 122-133, 2014.

[43] J. Lu and Q. Qiu, "Scheduling and Mapping of Periodic Tasks on Multi-Core Embedded Systems with Energy Harvesting," in *the International Green Computing Conference (IGCC)*, Los Alamitos, CA, USA, 2011.

[44] D. Zhang, Y. Liu, X. Sheng, J. W. T. Li, C. J. Xue and H. Yang, "Deadline-Aware Task Scheduling for Solar-Powered Nonvolatile Sensor Nodes with Global Energy Migration," in *the Design Automation Conference (DAC)*, San Francisco, CA, USA, 2015.

[45] Y. Zhang, Y. Ge and Q. Qiu, "Improving Charging Efficiency with Workload Scheduling in Energy Harvesting Embedded Systems," in *the Design Automation Conference (DAC)*, Austin, TX, USA, 2013.

[46] F. Ongaro, S. Saggini and P. Mattavelli, "Li-Ion Battery-Supercapacitor Hybrid Storage System for A Long Lifetime, Photovoltaic-Based Wireless Sensor Network," *the IEEE Transactions on Power Electronics (TPE)*, vol. 27, no. 9, pp. 3944-3952, 2012.

[47] A. Mirhoseini and F. Koushanfar, "HypoEnergy: Hybrid Supercapacitor-Battery Power-Supply Optimization for Energy Efficiency," in *the Conference on Design, Automation and Test in Europe (DATE)*, Los Alamitos, CA, USA, 2011.

[48] R. Teodorescu and J. Torrellas, "Variation-Aware Application Scheduling and Power Management for Chip Multiprocessors," in *the International Symposium on Computer Architecture (ISCA)*, Beijing, China, 2008.

[49] W. L. Hung, Y. Xie, N. Vijaykrishnan, M. Kandemir and M. J. Irwin, "Thermal-Aware Task Allocation and Scheduling for Embedded Systems," in *the Conference on Design, Automation and Test in Europe (DATE)*, Washington, DC, USA, 2005.

[50] E. L. Sueur and G. Heiser, "Dynamic Voltage and Frequency Scaling: The Laws of Diminishing Returns," in *the International Conference on Power Aware Computing and Systems (HotPower)*, Berkeley, CA, USA, 2010.

[51] W. Kim, M. S. Gupta, G. Y. Wei and D. Brooks, "System Level Analysis of Fast, Per-Core DVFS Using On-Chip Switching Regulators," in *the International Symposium On High Performance Computer Architecture (HPCA)*, Salt Lake City, UT, USA, 2008.

[52] R. Jejurikar, C. Pereira and R. Gupta, "Leakage Aware Dynamic Voltage Scaling for Real-Time Embedded Systems," in *the Design Automation Conference (DAC)*, San Diego, CA, USA, 2004.

[53] I. Yeo, C. C. Liu and E. J. Kim, "Predictive Dynamic Thermal Management for Multicore Systems," in *the Design Automation Conference (DAC)*, Anaheim, CA, USA, 2008.

[54] A. K. Coskun, T. T. Rosing, K. A. Whisnant and K. C. Gross, "Static and Dynamic Temperature-Aware Scheduling for Multiprocessor SoCs," *the IEEE Transactions on Very Large Scale Integration Systems (TVLSI)*, vol. 16, no. 9, pp. 1127-1140, 2008.

[55] B. Carter, J. Matsumoto, A. Prater and D. Smith, "Lithium Ion Battery Performance and Charge Control," in *the International Energy Conversion Engineering Conference (IECEC)*, Washington, DC, USA, 1996.

[56] Z. Xu, Z. Li, C. M. B. Holt, X. Tan, H. Wang, B. S. Amirkhiz, T. Stephenson and D. Mitlin, "Electrochemical Supercapacitor Electrodes from Sponge-Like Graphene Nanoarchitectures with Ultrahigh Power Density," *the Journal of Physical Chemistry Letters (JPCL)*, vol. 3, no. 20, pp. 2928-2933, 2012.

[57] B. Hargreaves, H. Hult and S. Reda, "Within-Die Process Variations: How Accurately Can They Be Statistically Modeled," in *the Asia and South Pacific Design Automation Conference (ASPDAC)*, Seoul, South Korea, 2008.

[58] D. Rajan, R. Zuck and C. Poellabauer, "Workload-Aware Dual-Speed Dynamic Voltage Scaling," in *the International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, Sydney, Qld., Australia, 2006.

[59] J. Park, S. D., C. N. and M. Pedram, "Accurate Modeling and Calculation of Delay and Energy Overheads of Dynamic Voltage Scaling in Modern High-Performance Microprocessors," in *the International Symposium on Low Power Electronics and Design (ISLPED)*, Austin, TX, USA, 2010.

[60] "Measurement and Instrumentation Data Center," National Renewable Energy Laboratory, [Online]. Available: http://www.nrel.gov/midc/. [Accessed June 2015].

[61] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge and R. B. Brown, "MiBench: A Free, Commercially Representative Embedded Benchmark Suite," in *the International Workshop on Workload Characterization (WWC)*, Washington, DC, USA, 2001.

[62] W. Huang, S. Ghosh, S. Velusamy, K. Sankaranarayanan, K. Skadron and M. R. Stan, "Hotspot: A Compact Thermal Modeling Methodology for Early-Stage VLSI Design," *the IEEE Transactions on Very Large Scale Integration Systems (TVLSI)*, vol. 14, no. 5, pp. 501-513, 2006.

[63] "The Gem5 Simulator," [Online]. Available: http://www.m5sim.org/. [Accessed June 2015].

[64] Y. K. Kwok and I. Ahmad, "Benchmarking the Task Graph Scheduling Algorithms," in *the International Parallel Processing Symposium (IPPS)*, Orlando, FL, USA, 1998.

[65] D. Zhu and H. Aydin, "Energy Management for Real-Time Embedded Systems with Reliability Requirements," in *the International Conference on Computer-Aided Design (ICCAD)*, San Jose, CA, USA, 2006.

[66] B. Zhao, H. Aydin and D. Zhu, "Generalized Reliability-Oriented Energy Management for Real-Time Embedded Applications," in *the Design Automation Conference (DAC)*, San Diego, CA, USA, 2011.

[67] B. Zhao, H. Aydin and D. Zhu, "Shared Recovery for Energy Efficiency and Reliability Enhancements in Real-Time Applications with Precedence Constraints," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 18, no. 2, p. 21, 2013.

[68] Y. Zou and S. Pasricha, "Reliability-Aware and Energy-Efficient Synthesis of NoC Based MPSoCs," in *the International Symposium on Quality Electronic Design (ISQED)*, Santa Clara, CA, USA, 2013.

[69] Y. Zou, Y. Xiang and S. Pasricha, "Analysis of On-Chip Interconnection Network Interface Reliability in Multicore Systems," in *the International Conference on Computer Design (ICCD)*, Amherst, MA, USA, 2011.

[70] Y. Zou, Y. Xiang and P. S., "Characterizing Vulnerability of Network Interfaces in Embedded Chip Multiprocessors," *the IEEE Embedded Systems Letters*, vol. 4, no. 2, pp. 41-44, 2012.

[71] A. K. Coskun, R. Strong, D. M. Tullsen and T. S. Rosing, "Evaluating the Impact of Job Scheduling and Power Management on Processor Lifetime for Chip Multiprocessors," in

the International Joint Conference on Measurement and Modeling of Computer Systems (SIGMETRICS), Seattle, WA, USA, 2009.

[72] L. Huang, F. Yuan and Q. Xu, "Lifetime Reliability-Aware Task Allocation and Scheduling for MPSoC Platforms," in *the Conference on Design, Automation and Test in Europe (DATE)*, Nice, France, 2009.

[73] M. Basoglu, M. Orshansky and E. M., "NBTI-Aware DVFS: A New Approach to Saving Energy and Increasing Processor Lifetime," in *the International Symposium on Low Power Electronics and Design (ISLPED)*, Austin, TX, USA, 2010.

[74] S. S. Mukherjee, J. Emer and S. K. Reinhardt, "The Soft Error Problem: An Architectural Perspective," in *the International Symposium On High Performance Computer Architecture (HPCA)*, Hudson, MA, USA, 2005.

[75] D. Zhu, R. Melhem and D. Mosse, "The Effects of Energy Management on Reliability in Real-Time Embedded Systems," in *the International Conference on Computer-Aided Design (ICCAD)*, San Jose, CA, USA, 2004.

[76] J. Luo and N. K. Jha, "Power-Conscious Joint Scheduling of Periodic Task Graphs and Aperiodic Tasks in Distributed Real-Time Embedded Systems," in *the International Conference on Computer-Aided Design (ICCAD)*, San Jose, California, USA, 2000.

[77] R. Sakellariou and H. Zhao, "A Hybrid Heuristic for DAG Scheduling on Heterogeneous Systems," in *the International Parallel and Distributed Processing Symposium (IPDPS)*, Santa Fe, NM, USA, 2004.

[78] H. F. Sheikh and I. Ahmad, "Dynamic Task Graph Scheduling on Multicore Processors for Performance, Energy, and Temperature Optimization," in *the International Green Computing Conference (IGCC)*, Arlington, VA, USA, 2013.

[79] ReliaSoft, "The Limitations of Using the MTTF as a Reliability Specification," [Online]. Available: http://www.weibull.com/hotwire/issue32/hottopics32.htm. [Accessed November 2014].

[80] ReliaSoft, "The Weibull Distribution," [Online]. Available: http://reliawiki.org/index.php/The_Weibull_Distribution . [Accessed July 2015].

[81] V. Suhendra, C. Raghavan and T. Mitra, "Integrated Scratchpad Memory Optimization and Task Scheduling for MPSoC Architectures," in *the International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES)*, Seoul, Republic of Korea, 2006.

[82] B. Xia and Z. Tan, "Tighter Bounds of the First Fit Algorithm for the Bin-Packing Problem," *the Discrete Applied Mathematics*, vol. 158, no. 15, pp. 1668-1675, 2010.

[83] X. Wang, M. Tehranipoor, S. George, D. Tran and L. Winemberg, "Design and Analysis of A Delay Sensor Applicable to Process/Environmental Variations and Aging Measurements," *the IEEE Transactions on Very Large Scale Integration Systems (TVLSI)*, vol. 20, no. 8, pp. 1405-1418, 2011.

[84] A. Makhorin, "GNU Linear Programming Kit," [Online]. Available: http://www.gnu.org/software/glpk/. [Accessed March 2015].

[85] Gurobi, "Gurobi Optimizer Reference Manual," 2014. [Online]. Available: http://www.gurobi.com/documentation/6.0/refman.pdf. [Accessed March 2014].

[86] R. Dick, "Embedded System Synthesis Benchmarks Suites (E3S)," [Online]. Available: http://ziyang.eecs.umich.edu/~dickrp/e3s/. [Accessed May 2014].

[87] R. P. Dick, D. L. Rhodes and W. Wolf, "TGFF: Task Graphs for Free," in *the International Workshop on Hardware/Software Codesign (CODES/CASHE)*, Seattle, WA, USA, 1998.

[88] R. Watanabe, M. Kondo, M. Imai, H. Nakamura and T. Nanya, "Task Scheduling under Performance Constraints for Reducing the Energy Consumption of the GALS Multi-Processor SoC," in *the Conference on Design, Automation and Test in Europe (DATE)*, Nice, France, 2007.

[89] R. Kirner, "Ingredients for the Specification of Mixed-Criticality Real-Time Systems," in *the International Symposium on Object/Component-Oriented Real-Time Distributed Computing (ISORC)*, Reno, NV, USA, 2014.

[90] L. Sha, "Resilient Mixed-Criticality Systems," *CrossTalk: the Journal of Defense Software (JDS)*, 2009.

[91] M. Hamdaoui and P. Ramanathan, "A Dynamic Priority Assignment Technique for Streams with (m, k)-Firm Deadlines," *the IEEE Transactions on Computers (TC)*, vol. 44, no. 12, pp. 1443-1451, 2012.

[92] Nvidia, "Variable SMP – A Multicore CPU Architecture for Low Power and High Performance," [Online]. Available: http://www.nvidia.com/content/PDF/tegra_white_papers/tegra-whitepaper-0911b.pdf. [Accessed June 2015].

[93] R. G. Dreslinski, M. Wieckowski, D. Blaauw, D. Sylvester and T. Mudge, "Near-Threshold Computing: Reclaiming Moore's Law through Energy Efficient Integrated Circuits," *the Proceedings of IEEE (IEEE)*, vol. 98, no. 2, pp. 253-266, 2010.

[94] D. Fick, R. G. Dreslinski, B. Giridhar, G. Kim, S. Seo, M. Fojtik, S. Satpathy, Y. Lee, D. Kim, N. Liu, M. Wieckowski, G. Chen, T. Mudge, D. Blaauw and S. D., "Centip3De: A 3930DMIPS/W Configurable Near-Threshold 3D System with 64 ARM Cortex-M3 Cores," in *the IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, San Francisco, CA, USA, 2012.

[95] U. R. Karpuzcu, K. B. Kolluru, N. S. Kim and J. Torrellas, "VARIUS-NTV: A Microarchitectural Model to Capture the Increased Sensitivity of Manycores to Process Variations at Near-Threshold Voltages," in *the l IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, Boston, MA, USA, 2012.

[96] T. N. Miller, X. Pan, R. Thomas, N. Sedaghati and R. Teodorescu, "Booster: Reactive Core Acceleration for Mitigating the Effects of Process Variation and Application Imbalance in Low-Voltage Chips," in *the International Symposium on High Performance Computer Architecture (HPCA)*, New Orleans, LA, USA, 2012.

[97] U. R. Karpuzcu, A. Sinkar, N. S. Kim and J. Torrellas, "EnergySmart: Toward Energy-Efficient Manycores for Near-Threshold Computing," in *the International Symposium on High Performance Computer Architecture (HPCA)*, Shenzhen, China, 2013.

[98] U. R. Karpuzcu, I. Akturk and N. S. Kim, "Accordion: Toward Soft Near-Threshold Voltage Computing," in *the International Symposium on High Performance Computer Architecture (HPCA)*, Orlando, FL, USA, 2014.

[99] H. Chen, D. Manzi, R. Sanghamitra and K. Chakraborty, "NTC: Exploiting the Paradigm Shift in Performance Bottlenecks," in *the Design Automation Conference (DAC)*, New York, NY, USA, 2015.

[100] S. Vestal, "Preemptive Scheduling of Multi-Criticality Systems with Varying Degrees of Execution Time Tssurance," in *the IEEE International Real-Time Systems Symposium (RTSS)*, Tucson, AZ, USA, 2007.

[101] M. S. Mollison, J. P. Erickson, J. H. Anderson and S. K. S. J. A. Baruah, "Mixed-Criticality Real-Time Scheduling for Multicore Systems," in *the IEEE International Conference on Computer and Information Technology (CIT)*, Bradford, UK, 2010.

[102] G. Giannopoulou, N. Stoimenov, P. Huang and T. L., "Scheduling of Mixed-Criticality Applications on Resource-Sharing Multicore Systems," in *the ACM International Conference on Embedded Software (EMSOFT)*, Montreal, QC, Canada, 2013.

[103] P. K. Saraswat, P. Pop and J. Madsen, "Task Migration for Fault-Tolerance in Mixed-Criticality Embedded Systems," *ACM SIGBED Review - Special Issue on the 2nd International Workshop on Adaptive and Reconfigurable Embedded Systems (APRES)*, vol. 6, no. 3, p. 5, 2009.

[104] P. Huang, H. Yang and L. Thiele, "On the Scheduling of Fault-Tolerant Mixed-Criticality Systems," in *the Design Automation Conference (DAC)*, San Francisco, CA, USA, 2014.

[105] P. Huang, P. Kumar, G. Giannopoulou and L. Thiele, "Run and be Safe: Mixed-Criticality Scheduling with Temporary Processor Speedup," in *the Conference on Design, Automation and Test in Europe (DATE)*, Grenoble, France, 2015.

[106] G. Giannopoulou, N. Stoimenov, P. Huang and L. Thiele, "Mapping Mixed-Criticality Applications on Multi-Core Architectures," in *the Conference on Design, Automation & Test in Europe (DATE)* , Dresden, Germany, 2014.

[107] C. Gu, N. Guan, Q. Deng and W. Yi, "Partitioned Mixed-criticality Scheduling on Multiprocessor Platforms," in *the Conference on Design, Automation & Test in Europe (DATE)*, Dresden, Germany, 2014.

[108] S. H. Kang, H. Yang, S. Kim, I. Bacivarov, S. Ha and L. Thiele, "Reliability-Aware Mapping Optimization of Multi-Core Systems with Mixed-Criticality," in *the Conference on Design, Automation & Test in Europe (DATE)*, Dresden, Germany, 2014.

[109] D. Tamas-Selicean and P. Pop, "Design Optimization of Mixed-Criticality Real-Time Applications on Cost-Constrained Partitioned Architecture," in *the IEEE Real-Time Systems Symposium (RTSS)* , Vienna, Austria, 2011.

[110] S. Trujillo, A. Crespo and A. Alonso, "MultiPARTES: Multicore Virtualization for Mixed-Criticality Systems," in *the Conference on Digital System Design (DSD)*, Los Alamitos, CA, USA, 2013.

[111] K. V. Craeynest, A. Jaleel, L. Eeckhout, P. Narvaez and J. Emer, "Scheduling Heterogeneous Multi-Cores through Performance Impact Estimation (PIE)," in *the International Symposium on Computer Architecture (ISCA)*, Portland, OR, USA, 2012.

[112] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh and A. Gupta, "The SPLASH-2 Programs: Characterization and Methodological Considerations," in *the International Symposium on Computer architecture (ISCA)* , Santa Margherita Ligure, Italy, 1995.

[113] C. Bienia, "Benchmarking Modern Multiprocessors," Ph.D. Dissertation. Princeton University, Princeton, NJ, USA, 2011.

[114] W. Kim, M. S. Gupta, G. Y. Wei and D. Brooks, "System Level Analysis of Fast, Per-Core DVFS Using On-Chip Switching Regulators," in *the International Symposium on High Performance Computer Architecture (HPCA)*, Salt Lake City, UT, USA, 2008.

[115] M. Bhadauria, V. Weaver and S. A. McKee, "A Characterization of the PARSEC Benchmark Suite for CMP Design," Cornell University, 2008.

[116] C. Xian, Y. H. Lu and Z. Li, "Energy-Aware Scheduling for Real-Time Multiprocessor Systems with Uncertain Task Execution Time," in *the Design Automation Conference (DAC)*, San Diego, CA, USA, 2007.

[117] T. E. Carlson, W. Heirman and E. L., "Sniper: Exploring the Level of Abstraction for Scalable and Accurate Parallel Multi-Core Simulation," in *the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, Seatle, WA, USA, 2011.

[118] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen and N. P. Jouppi, "McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures," in *the IEEE/ACM International Symposium on Microarchitecture (MICRO)*, New York, NY, USA, 2009.

[119] A. H. Maheran, P. Menon, I. Ahmad and Z. Yusoff, "Threshold Voltage Optimization in A 22nm High-k/Silicide PMOS Device," in *the IEEE Regional Symposium on Micro and Nanoelectronics (RSM)*, Langkawi, Malaysia, 2013.

[120] V. Saripalli, "Device and Architecture Co-Design for Ultra-Low Power Logic Using Emerging Tunneling-Based Devices," Ph.D. Dissertation. Pennsylvania State University., 2011.

[121] Intel, "Intel Atom Processor," [Online]. Available: www.intel.com/technology/atom. [Accessed June 2015].

[122] B. Donohoo, O. C., S. Pasricha, C. Anderson and Y. Xiang, "Context-Aware Energy Enhancements for Smart Mobile Devices," *the IEEE Transactions on Mobile Computing (TMC)*, vol. 13, no. 8, pp. 1720-1732, 2014.

[123] R. Prabha, G. Rincon-Mora and S. Kim, "Harvesting Circuits for Miniaturized Photovoltaic Cells," in *the IEEE International Symposium on Circuits and Systems (ISCAS)*, Rio de Janeiro, Brazil, 2011.

[124] H. G. Lee and N. Chang, "Energy-Aware Memory Allocation in Heterogeneous Non-Volatile Memory Systems," in *the International Symposium on Low Power Electronics and Design (ISLPED)*, Seoul, Korea, 2003.

[125] Y. Liu, Z. Li, H. Li, Y. Wang, X. Li, K. Ma, S. Li, M. F. Chang, S. John, Y. Xie, J. Shu and Y. H., "Ambient Energy Harvesting Nonvolatile Processors: From Circuit to System," in *the Design Automation Conference (DAC)*, San Francisco, CA, USA, 2015.

[126] J. Han and M. Orshansky, "Approximate Computing: An Emerging Paradigm for Energy-Efficient Design," in *the IEEE European Test Symposium (ETC)*, Avignon, France, 2013.

[127] V. Gupta, D. Mohapatra, S. P. Park, A. Raghunathan and K. Roy, "IMPACT: Imprecise Adders for Low-Power Approximate Computing," in *the IEEE/ACM International Symposium on Low-Power Electronics and Design (ISLPED)*, Fukuoka, Japan, 2011.

[128] Y. Xiang and S. Pasricha, "Thermal-Aware Semi-Dynamic Power Management for Multicore Systems with Energy Harvesting," in *the International Symposium on Quality Electronic Design (ISQED)*, Santa Clara, CA, USA, 2013.

[129] Y. Xiang and S. Pasricha, "Harvesting-Aware Energy Management for Multicore Platforms with Hybrid Energy Storage," in *the Great Lakes Symposium on VLSI (GLSVLSI)*, Paris, France, 2013.

[130] Y. Xiang and S. Pasricha, "A Hybrid Framework for Application Allocation and Scheduling in Multicore Systems with Energy Harvesting," in *the Great Lakes Symposium on VLSI (GLSVLSI)*, Houston, TX, USA, 2014.

[131] Y. Xiang and S. Pasricha, "Fault-Aware Application Scheduling in Low Power Embedded Systems with Energy Harvesting," in *the International Conference on Hardware/Software Codesign and System Synthesis (CODES)*, New Delhi, India, 2014.

[132] Y. Xiang and S. Pasricha, "Run-Time Management for Multi-Core Embedded Systems with Energy Harvesting," *the IEEE Transactions on Very Large Scale Integration Systems (TVLSI)*, 2014.