3-9-2012

Power and Thermal Aware Scheduling for Real-time Computing Systems

Huang Huang
Florida International University, hhuan001@fiu.edu

DOI: 10.25148/etd.FI12042309

FLORIDA INTERNATIONAL UNIVERSITY

Miami, Florida

POWER AND THERMAL AWARE SCHEDULING FOR REAL-TIME COMPUTING SYSTEMS

A dissertation submitted in partial fulfillment of the

requirements for the degree of

DOCTOR OF PHILOSOPHY

in

ELECTRICAL ENGINEERING

by

Huang Huang

2012

To: Dean Amir Mirmiran
College of Engineering and Computing

This dissertation, written by Huang Huang, and entitled Power and Thermal Aware Scheduling for Real-time Computing Systems, having been approved in respect to style and intellectual content, is referred to you for judgment.

We have read this dissertation and recommend that it be approved.

Malek Adjouadi

Jiuhua Chen

Chen Liu

Nezih Pala

Gang Quan, Major Professor

Date of Defense: March 9, 2012

The dissertation of Huang Huang is approved.

Dean Amir Mirmiran
College of Engineering and Computing

Dean Lakshmi N. Reddi
University Graduate School

Florida Internatinal University, 2012

DEDICATION

I would like to dedicate this Doctoral dissertation to my beloved wife, Yuanyuan, my dear parents and terrific in-laws. Without their love, understanding, support, and encouragement, the completion of this endeavor would never have been possible.

ACKNOWLEDGMENTS

I wish to express my deepest appreciation to my major professor, Dr. Gang Quan, for his guidance, encouragement, support, and patience. His passion, dedication and sincere interests in scientific research have been a great inspiration to me.

I would also like to thank my committee members, Dr. Malek Adjouadi, Dr. Jiuhua Chen, Dr. Chen Liu and Dr. Nezih Pala, for their very helpful insights, comments and suggestions to improve the quality of this dissertation.

Finally, I would like to thank the National Science Foundation (NSF) for supporting the research described in this dissertation through grants CNS-0969013, CNS-0917021 and CNS-1018108.

ABSTRACT OF THE DISSERTATION

POWER AND THERMAL AWARE SCHEDULING FOR REAL-TIME COMPUTING SYSTEMS

Huang Huang

Florida International University, 2012

Miami, Florida

Professor Gang Quan, Major Professor

Over the past few decades, we have been enjoying tremendous benefits thanks to the revolutionary advancement of computing systems, driven mainly by the remarkable semiconductor technology scaling and the increasingly complicated processor architecture. However, the exponentially increased transistor density has directly led to exponentially increased power consumption and dramatically elevated system temperature, which not only adversely impacts the system's cost, performance and reliability, but also increases the leakage and thus the overall power consumption. Today, the power and thermal issues have posed enormous challenges and threaten to slow down the continuous evolvement of computer technology. Effective power/thermal-aware design techniques are urgently demanded, at all design abstraction levels, from the circuit-level, the logic-level, to the architectural-level and the system-level.

In this dissertation, we present our research efforts to employ real-time scheduling techniques to solve the resource-constrained power/thermal-aware, design-optimization problems. In our research, we developed a set of simple yet accurate system-level models to capture the processor's thermal dynamic as well as the interdependency of leakage power consumption, temperature, and supply voltage. Based on these models, we investigated the fundamental principles in power/thermal-aware scheduling, and developed real-time scheduling techniques targeting at a variety of design objectives, including peak temperature minimization, overall energy reduction, and performance

maximization.

The novelty of this work is that we integrate the cutting-edge research on power and thermal at the circuit and architectural-level into a set of accurate yet simplified system-level models, and are able to conduct system-level analysis and design based on these models. The theoretical study in this work serves as a solid foundation for the guidance of the power/thermal-aware scheduling algorithms development in practical computing systems.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

The increased transistor density and the exponentially growing power consumption of modern computing systems have imposed enormous challenges that not only threaten to slow down the pace of technology advancement, but also raise serious concerns related to the economic efficiency and environmental protection. High power consumption directly translates into high temperature, which also makes the thermal issue a challenging problem to be addressed. Evidently, power/thermal-aware techniques are urgently needed at every design abstraction level in contemporary computing system design. This chapter presents the motivations behind this research, introduces an overview of the research problem, and discusses the major contributions made in this dissertation.

1.1 High Power Consumption Problem

The continuous shrinking of the semiconductor transistor feature size, together with the increasingly complicated circuit architecture, have resulted in an exponential increase of power density, which has imposed enormous challenges and threatens to slow down this progress.

According to [20], more than 40 billion transistors are being integrated into a single $300mm^2$ die today, and as shown in Figure 1.1[1], the number is growing rapidly toward 100 billion by the middle of 2010's. The power consumed by these transistors is significant, reaching 300 watt in a matter of a few years. The soaring power consumption has posed immediate challenges for system designers in both portable devices and power-rich systems.

On one hand, battery-operated portable devices, in recent years, have been ex-

[1]Figure 1.1 is plotted based on the data reported in [20].

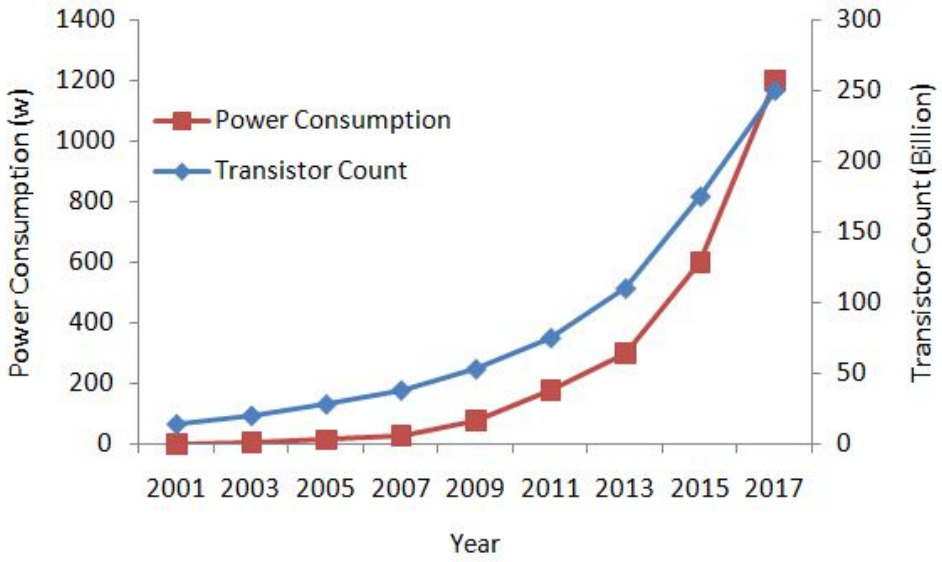


Figure 1.1: The trend of power consumption and transistor count for a $300mm^2$ die

periencing revolutionary improvement ranging from functionality, application and performance. All complicated applications and powerful hardware resources demand high-capacity batteries to sustain the battery life for the sake of mobility and applicability. Unfortunately, the trend that devices are simultaneously asked to do more and getting smaller severely limits the battery size and therefore the power they could generate. The current battery technology, with less-than-5% annual capacity improvement [9], cannot effectively address this problem. In fact, in many modern high-performance portable devices, such as smart phones and tablets, battery packs have already taken the majority of the space within devices' enclosures [1]. Evidently, unless sophisticated power reduction techniques are successfully implemented, the advancement of the mobile computing industry can be severely handicapped by the limitation of battery life, or the "gating factor".

On the other hand, even for power-rich computing systems, such as data centers, the energy consumption is also a major issue. The power usage of U.S. data centers had doubled between 2000 and 2005. According to an Environmental Protection Agency (EPA) report [11], in 2006, data centers within the U.S. consumed 61 billion

kilowatt hours of energy, which is equivalent to 1.5% of all energy consumed in the U.S. - at a cost of 4.5 billion dollars and continued to grow at 12% per year. It is estimated that, as of today, the electricity cost of a server over its lifetime will pass the price of the hardware. Even worse, an estimation of 70% of the electricity in the U.S. is generated by fossil fuel. Therefore, considering the costs related to power consumption and the impact on the environment, data center-related technological innovations are urgently needed from different aspects, such as cooling equipments, power conversion/distribution and most importantly, power-efficient computing systems.

1.2 Why Temperature Matters

The soaring power consumption not only has presented significant challenge that stresses power supply, but also makes the heat dissipation and the temperature control even more critical and challenging.

The elevated temperature caused by heat dissipation significantly affects the reliability and the performance of a computing system. It can even cause catastrophic system failures. The reliability of an electronic system can be modeled by using the Arrhenius equation [53, 114], i.e. $MTF = MTF_0 e^{\frac{E_a}{K_b T}}$, where MTF is the mean-time-to-failure of a system, and T is the operating temperature. When a system's operating temperature increases, its mean-time-to-failure decreases exponentially. According to Yeh and Chu [114], even if a processor does not completely fail at a high temperature, a small increase in temperature, e.g. 10^oC, can result in as much as 50% reduction in the device's life span.

Moreover, temperature increase also worsens the performance. The clock timing in the CMOS circuit is very sensitive to temperature variations. Each 15^oC increase in temperature can add roughly 10% to 15% to the circuit delay [96].

Furthermore, the escalating chip temperature has directly led to high packaging and cooling costs. It is estimated that the thermal packaging cost increases at 1-3 dollar per watt [100, 101]. With estimated peak power of future processors well over 300 watts [55], this portion of cost seriously undermines the benefit of new generations of computing systems.

Even though there are novel and impressive cooling techniques and thermal materials being developed (e.g. [87]), the cost-effective heat removal capability remains almost flat in the foreseeable future [87, 55]. The severity of the thermal problem is further highlighted by Intel's acknowledgement that it has hit a "thermal wall" [78].

Similarly, thermal challenges exist in high-performance computing systems [12, 17, 21, 100]. Data center designers and operators have to expend tremendous effort on heat management to improve operational performance and minimize system downtime. At a data center with many power-hungry computing and networking equipments, cooling cost has been the primary source of the increased operational costs. It is estimated that $4-8$ million U.S. dollar a year is spent for cooling alone in a 30,000 square feet data center with 1000 standard computing racks [81]. It is further estimated that half to one watt has to be consumed just for cooling in order to sustain each watt consumed on computation [17, 21]. According to [21], the dramatic increase in engineering costs and financial burdens of providing power and cooling for data centers have become serious economic problems and will eventually cause the "economic meltdown of Moore's Law".

In addition, high temperature also increases the leakage power consumption, which is becoming one of the major components in overall power, e.g. up to 70% of total power [55]. As shown in Figure 1.2 [55], the leakage power consumption will increase dramatically in the near future. It is catching up and even surpassing the dynamic power consumption as the IC technology continues its marching toward the deep sub-

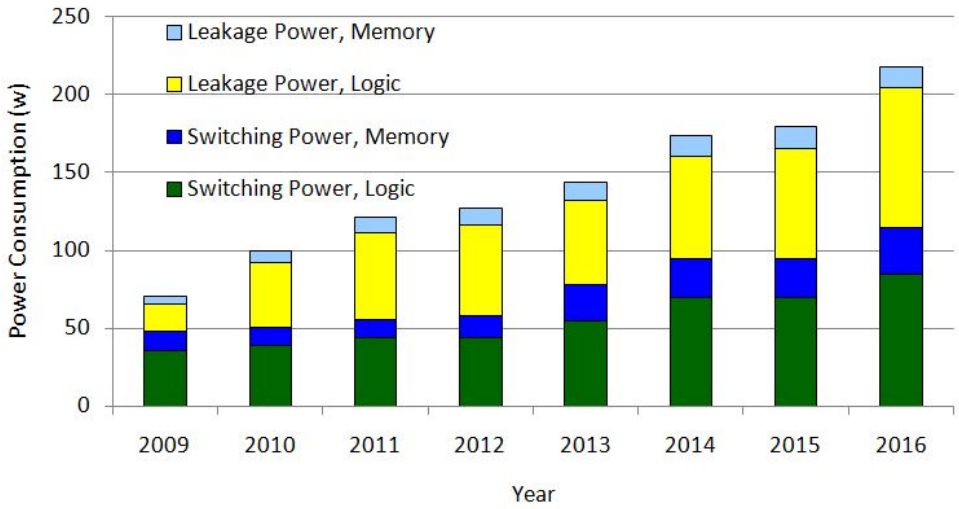


Figure 1.2: The portion of dynamic and leakage power consumption of stationary systems projected by ITRS (2010)

micron (DSM) era. Based on the UC Berkeley's BSIM device model [18], Liao et al. [65] showed that the leakage power consumption can be 2-3 times higher than the dynamic power consumption for processors using the 65nm technology.

It has been shown that the leakage power is highly dependent upon the system temperature and a positive feedback loop exists between the two. Taking the 65nm technology for example, the leakage power will increase by 21% when chip temperature raises from 60^oC to 80^oC and the increased power consumption will in turn produce more heat to escalade the temperature [66]. Therefore, as the thermal issue becomes increasingly prominent, the system temperature has to be incorporated as a critical design metric in computing system development. And any power/thermal-aware design technique can become ineffective if the interdependency between the leakage and temperature is not properly considered.

1.3 Research Problems and Our Contributions

In the face of the grave challenges to deal with the power consumption and heat generated by the computing systems today, power/thermal-aware design techniques are

urgently demanded from all perspectives and design abstraction levels. To tackle this problem, efforts have been made at every design abstraction level including circuit-level, logic-level, architectural-level and system-level. In this dissertation, we are focusing on addressing this problem from the system-level. Specifically, we study the power/thermal-aware scheduling problems for real-time computing systems. The aim of this research is to develop appropriate real-time scheduling algorithms to address the system-level power/thermal-aware design optimization problems with the interplay between temperature and leakage power consumption being taken into consideration. The research will incorporate advanced power manageable features, e.g. dynamic voltage/frequency scaling (DVS) and dynamic power down (DPD), in the state-of-the-art computer architecture to meet the increasingly stringent timing requirement, as well as optimize other performance metrics, such as peak temperature, energy consumption and throughput, of real-time systems.

The contributions of this dissertation can be summarized as follows:

1. We integrate the state-of-the-art research on power, leakage, and temperature at the circuit and architectural-level and develop a set of accurate yet simplified power and thermal models which can be used to effectively capture the processor's thermal dynamic as well as the leakage/temperature dependency. The linear-approximation leakage power model developed in this research helps to greatly simplify the complexity of temperature calculation while achieves reasonably good accuracy, e.g. 1.3% average relative error for leakage calculation and less than 4.8% relative error when calculating the temperature.

2. Based on the proposed system-level models, we study the problem on how to minimize the peak temperature of a processor when executing a periodic task set. We propose an approach based on an existing algorithm, called *M*-

Oscillating [28], that oscillates task executions between the high and low processor speeds. Different from the original algorithm proposed in [28], in this research, we incorporate the non-negligible processor mode switching overhead into analysis and present a fast searching algorithm that can be used to efficiently find the appropriate number of mode switchings that can lead to the minimized peak temperature. From our experimental study, we find that, without taking the transition overhead into consideration, the original *M-Oscillating* algorithm can in fact increase the peak temperature in practical scenarios, while our new approach can effectively accommodate the practical factors such as the transition overhead.

3. The temperature dependent leakage power makes the energy calculation complicated and computationally expensive. In our research, we further derive a novel, closed-form energy estimation method that can be used to accurately and efficiently calculate the energy consumption of a candidate schedule at both the transient and the thermal steady state. Based on the proposed energy calculation method, we then develop two scheduling techniques, i.e. an off-line method for a periodic task set and an on-line method targeting at an aperiodic task set, to minimize the overall energy consumption of real-time systems. Our experimental results show that the proposed energy estimation method can achieve up to 177X speedup compared with an existing approach [73] while still maintaining high accuracy (with relative error no more than 4.1%). With a large number of different test cases, the proposed off-line energy minimization scheduling method consistently outperforms two existing approaches, e.g. the Naive approach and the *Pattern-based* approach [110], by 26.5% and 15.3%, respectively. Our on-line approach, on the other hand, also shows superior performance in terms of over energy reduction. Compared with two existing approaches, the

Pattern-based approach and the *On-line DVS* approach [64], on average, our method gains 14% and 10% additional energy savings, respectively.

To the best of our knowledge, we are not aware of any other thermal-aware scheduling techniques that can guarantee the hard real-time deadlines for an aperiodic task set.

4. For a task set consisting of tasks with heterogeneous power and thermal profiles, we observe that the overall latency when running a task can be reduced if we split the task execution and insert cooling periods in between. Moreover, we notice that by frequently switching between the execution of tasks with different thermal characteristics, e.g. *hot* and *cool* tasks, the entire temperature curve can be maintained under a predefined threshold without introducing any cooling period. Based on these observations, we formally establish and successfully prove two important theorems, which are used to guide the development of our peak temperature constrained, throughput maximization scheduling algorithms for a periodic hard real-time system. Two scheduling approaches are presented for processors with and without DVS feature. Our experimental results, based on parameters drawn from the 65nm technology, show that on average our methods outperform the existing approaches [119] by over 23.3% and 5.3%, for a processor without and with DVS capability, respectively. Moreover, under different ambient temperature conditions, the proposed method can guarantee that all task can be completed without a single occurrence of peak temperature constraint violation, i.e. a 21% feasibility improvement over the existing approach [119].

1.4 Structure of the Dissertation

This dissertation is organized as follows. Chapter 2 introduces the background of this research. Specifically, we discuss a number of most important concepts in real-time computing. Then, we present a number of existing approaches in both power reduction and thermal management. Finally, we provide an overview of the literature on power/thermal-aware scheduling, which is closely related to our research in this work.

In Chapter 3, we provide some details about the system models, especially the leakage model that we used in this dissertation. We study a large spectrum of leakage power models, then analyze and compare the trade-off between the complexity and accuracy of these models, empirically. Based on our experimental results, the one that is able to account for the leakage/temperature dependency, and in the meantime, simple enough and thus suitable for the system-level design is chosen and used in this dissertation.

Based on the system models we derived in Chapter 3, three different, but closely related, problem areas are studied in Chapter 4, Chapter 5 and Chapter 6, respectively. Specifically, in Chapter 4, we investigate the operational temperature minimization problem. An effective speed scheduling algorithm is proposed to reduce the peak temperature of a processor when executing a hard real-time periodic task set. Despite that the temperature and the power consumption of a system are strongly correlated, a temperature minimization technique does not necessarily optimize the energy consumption. Therefore, in Chapter 5, we extend our study to the problem on how to schedule a hard real-time system to achieve the minimal overall energy, including both dynamic and leakage energy consumption. Chapter 6 focuses on the throughput maximization problem for a periodic real-time system under a given peak temperature constraint. We assume that different tasks in our system may have dif-

ferent power and thermal characteristics. Two scheduling approaches are presented. The first one is built upon processors that can be either in active or sleep mode. Then, we augment this approach to consider processors with DVS capability.

Finally, in Chapter 7, we conclude this dissertation and discuss the possible future work of this research.

CHAPTER 2

BACKGROUND AND RELATED WORK

This chapter presents our research background. We first introduce several important concepts related to real-time systems and real-time scheduling policies. Then, we discuss a number of studies on power reduction and thermal management techniques at different design abstraction levels. We further conduct a more specific survey on power/thermal-aware scheduling techniques, which are closely related to our research topic.

2.1 Real-time Computing and Scheduling

Real-time computing has become one of the most important problem areas in research and design of computing systems. In this section, we discuss some of the key concepts of real-time systems. Then, we briefly discuss some existing works on the real-time scheduling policy, i.e. a critical problem in studying real-time systems.

2.1.1 What is Real-time Computing

A real-time system is the one that has to respond to an externally generated stimuli within a finite and specified time period [98]. In a real-time system, time is critical that needs to be managed carefully. The correctness of an operation depends not only on the logical result but also on the time it is delivered. In a real-time system, failure to respond can cause a degraded quality of service (QoS), or even a catastrophical accident [98].

In order to understand the behaviors of real-time systems, it is necessary to develop models of real-time activities and to study their characteristics.

Periodic vs. Aperiodic

The real-time applications are consisting of a set of tasks. If the tasks are invoked at a regular interval, they are referred as periodic tasks. That is, there is a continuous and deterministic pattern of time intervals between requests of system resources. In addition, a real-time periodic task must be completed by a specified deadline relative to the time it is released. In contrast, aperiodic tasks refer to those who are only triggered by the occurrences of certain events. They are used to model real-time activities that request system resources during non-deterministic request periods.

There are ample examples exist in our everyday life, for both kinds of real-time applications. The auto-pilot system on an aircraft is one typical example of periodic real-time task. The onboard computer monitors a plane's current altitude, speed, position and other parameters on a periodic fashion. Then, based on the collected data, control signals are sent to adjust the plane's throttle, rudder, flap and etc. Examples of aperiodic real-time tasks can be found in event-driven real-time systems, from less sensitive applications, e.g. streaming media, satellite communication, to timing-critical applications, e.g. pilot seat ejection and missile defense systems.

Hard Real-time vs. Soft Real-time

Hard real-time tasks require deterministic guarantee to meet all deadlines for every instance, and the failure to meet even a single deadline can be catastrophic. Examples of hard real-time tasks can be found in aviation control system and automobile's ABS. In contrast, soft real-time tasks allow for a statistical bound on the number of deadline misses, which are neither desirable nor fatal. Examples of soft real-time applications include media streaming in distributed systems and non-mission-critical tasks in control systems. Similarly, a deadline is said to be firm if the result produced by the corresponding task ceases to be useful as soon as the deadline expires. In other

word, a late response has no value at all. Like the soft real-time tasks, firm real-time tasks can tolerate some deadline misses.

2.1.2 Research on Real-time Scheduling

In previous subsection, we have briefly introduced several important concepts of real-time computing. In this subsection, we discuss some of the existing works in real-time computing.

Over the years, there are a great number of studies focused on real-time computing related topics. These works have covered a large variety of applications including real-time task scheduling, e.g. [68, 33, 85, 79, 93], real-time architecture, e.g. [97, 24, 106, 32], real-time operating system, e.g. [36, 109, 54, 45, 37], real-time communication, e.g. [76, 123, 41, 82], and etc.

The real-time scheduling, among the above mentioned problem areas, is playing a critical role in a real-time system. Given tasks with their timing information, available system resources and design constraints, the real-time scheduling studies how to determine when and where a task needs to be executed so that the required design constraints, e.g. timing, are satisfied while some other design metrics, e.g. energy consumption, are optimized.

For a task set, a schedule is said to be feasible if we can ensure that every single task instance can be completed by its associated deadline. And a task set is called schedulable if there exists at least one feasible schedule. The utilization associated with a given task schedule and the available resource (e.g. CPU) is the fraction of time that the resource is allocated over the time during which the scheduler is active.

Many scheduling algorithms have been proposed for a variety of task and processor models. Based on the decision-making time, we have static scheduling and dynamic scheduling, where static scheduling makes decision off-line, e.g. during compilation or

synthesis time, whereas dynamic scheduling makes decision on-line, during the task execution. Moreover, if at any time instant, only the task with the highest priority can be executed, we call it preemptive scheduling, otherwise, non-preemptive scheduling. We also have priority-driven and non-priority-driven scheduling where priority-driven scheduling refers to the policy under which the task execution is dictated by its assigned priority. For non-priority-driven scheduling, some other policies are needed, such as round robin [5], to determine whether or not a task should start running. In addition, depending on whether tasks are executed on a single processor or multiple processors, we have single-processor and multi-processor scheduling.

Rate-monotonic (RM) policy and earliest deadline first (EDF) policy are two of the most important single-processor, priority-driven, preemptive scheduling policies [68]. They serve as the foundation of many other scheduling algorithms.

Rate Monotonic Under the fixed-priority rate monotonic scheduling algorithm, tasks' priorities are assigned based on their periods. It is shown by Liu and Layland [68] that RM is the optimal among all fixed-priority scheduling policies. They have proved that a feasible schedule can be found by using RM if the total utilization is less than or equal to $ln(2)$ (69.3%).

Earliest Deadline First The EDF is a preemptive, dynamic-priority scheduling algorithm. Task's priorities are assigned dynamically during run time. The task with the least time remaining before its deadline acquires the highest priority and thus executed before others. In fact, it is proved in [68] that if a task set is schedulable, then EDF algorithm can schedule it. Due to its 100% utilization bound, EDF becomes the underlying scheduling algorithm for a number of other scheduling techniques with different design objective, such as the "low power EDF" algorithm proposed in [112].

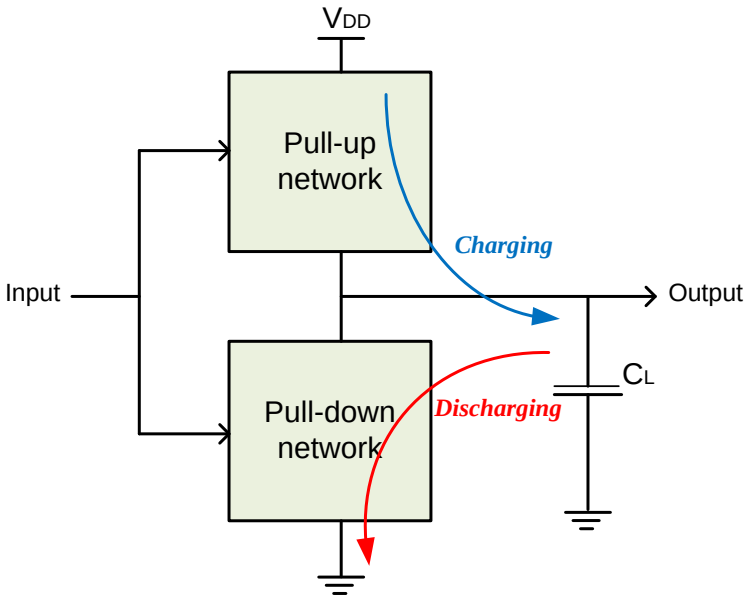


Figure 2.1: The illustration of dynamic power consumption in digital ICs

2.2 Power Reduction

In this section, we first introduce the sources of power consumption in DSM domain digital integrated circuits. Then, we discuss a number of existing power reduction techniques at different design levels.

2.2.1 Sources of Power Consumption in Digital Integrated Circuit

The power consumption in CMOS digital ICs, based on their sources, can be divided into two categories: dynamic power and static power [92].

Dynamic Power Consumption

The dynamic power consumption, also known as capacitive power, or switching power [92, 84], is associated with the switching of the logic value of a gate. As shown in Figure 2.1, this part of power consumption is essential to performing useful logic operation by charging and discharging the load capacitance.

The value of the switching power can be estimated by [92]

$$P_{switching} = \alpha C_L V_{dd}^2 f, \tag{2.1}$$

where α is the switching factor, which represents the number of state transitions in one clock cycle, C_L is the total load capacitance, V_{dd} is the supply voltage level and f is the clock frequency. As indicated by equation (2.1), the switching power is in proportion to the square of the V_{dd}, the complexity of a logic gate, the working frequency as well as the switching activity of a circuit.

Static Power Consumption

In contrast to the dynamic power, the static power is consumed due to the leakage mechanism of a CMOS transistor and it does not contribute to any useful computation.

To understand how leakage current occurs, we first need to understand how a transistor works. A transistor controls the flow of the current between two terminals, e.g. source and drain. When the transistor is in "off" state, no current is allowed to flow through the drain and the source terminal, because an insulating material, e.g. channel, is placed in between. However, when the voltage level increases at the gate terminal, the conductivity of the channel increases ("on" state), which allows the normal flowing of current between the drain and the gate. The value at which the gate's voltage is high enough to turn "on" the transistor is called the threshold voltage (V_t).

However, in an imperfect world, transistors are not built ideally. In other word, even when the applied gate voltage of a transistor is below its V_t, there are still different kinds of current leaking through.

As shown in Figure 2.2, based on the path of current flow, the leakage current

can be further categorized into three parts [84], the junction leakage current, the gate direct tunneling leakage current and the sub-threshold leakage current.

- The junction leakage is the current that flows from the source or drain to the substrate through a reverse-biased diode when a transistor is in off state. This part of leakage is determined by the physical property of a transistor.

- The gate direct tunneling leakage occurs from the gate terminal, through the dielectric material, to the substrate. Its magnitude depends on the thickness of the gate oxide material. With the introduction of high-k material, e.g. hafnium, this gate direct tunneling leakage current can be effectively reduced without sacrificing the circuit performance.

- The sub-threshold leakage is due to the diffusion current of the minority carriers in the channel of a transistor. It flows directly from the drain to the source even when a transistor is in off state. The sub-threshold leakage depends on the chip's temperature, V_t, supply voltage level as well as some other process dependent technology constants.

At the current CMOS technology node, the sub-threshold leakage is much larger than other leakage components [55], and thus, a great number of recent research efforts are focusing on dealing with the sub-threshold leakage (with details provided later).

2.2.2 Power Reduction Techniques

In this subsection, we introduce some of the most effective power reduction techniques, targeting at both the dynamic and the leakage power.

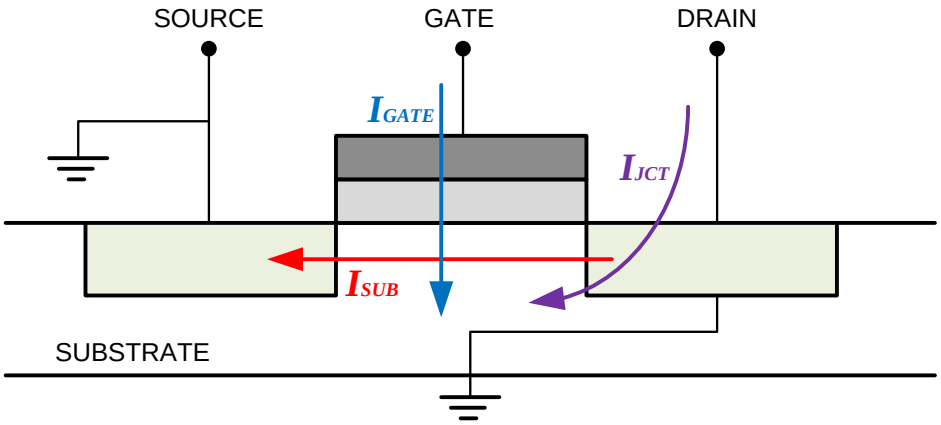


Figure 2.2: Three types of leakage current in CMOS transistor

Dynamic Power Reduction Techniques

The dynamic power used to be the dominant component of the overall power consumption. Early studies on power reduction techniques are mainly focusing on how to reduce the dynamic power. A great number of techniques are proposed at different design levels to tackle this problem.

Based on equation (2.1), the dynamic power consumption can be reduced from the following aspects.

- *Reducing the load capacitance*: Since the physical capacitance depends on low-level design parameters such as transistor sizes and wire lengths, we can reduce the capacitance, for example, by reducing transistor sizes.

- *Reducing the switching activity*: It is important to devise techniques in this category since the chip's switching activity is intensified by the increasingly complicated processor architecture. One intuitive method is to cut off the clock signal from reaching those functional blocks that are not working.

- *Reducing frequency*: The dynamic power can be reduced by lowering the clock frequency. However, this method can affect the system performance and does

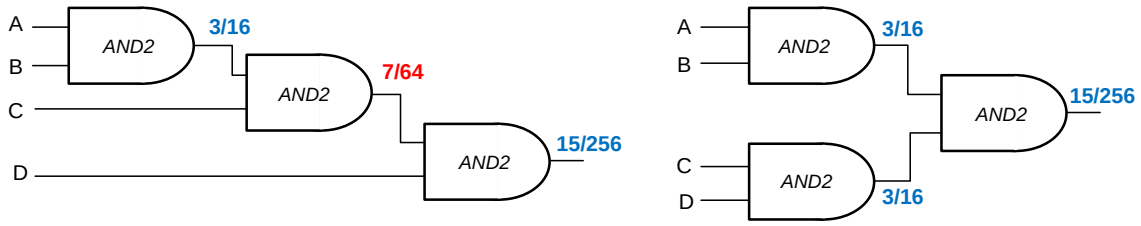
not help to reduce the energy consumption.

- *Reducing supply voltage*: Reducing the supply voltage is usually associated with a proportional lowering of clock frequency. Therefore, this is the most effective way to reduce dynamic power and theoretically, a linear scaling down of the execution speed can achieve a cubic dynamic power consumption reduction.

With these facts in mind, a number of dynamic power reduction schemes are proposed at the circuit, logic, architectural and system-level, including transistor sizing [58], input reordering [80], logic gate restructuring [92, 107], clock gating [77] and dynamic voltage scaling (DVS) [73].

Transistor sizing is a circuit-level approach that tries to determine the width of a transistor so that the dynamic power consumption can be minimized without sacrificing the circuit performance. The size of the transistor affects its equivalent capacitance, which determines the dynamic power consumption (equation (2.1)). In general, a larger transistor helps to reduce the logic gate's delay, but consumes more power and vice versa. Therefore, this technique can only be applied to those transistors that are not on the critical path to avoid performance penalty. Algorithms for applying this technique usually try to scale down the size of each transistor to be as small as possible without violating its delay constraint.

Input reordering is another effective circuit-level method that rearranges the order of transistors to minimize their switching activities. The guideline to implement this technique is to place transistors closer to a circuits output, if they switch more frequently, to prevent unnecessary switching activities of other transistors. This method requires a profiling stage to determine how frequently different transistors are likely to switch [107].



(a). Four input AND gate implemented in chain structure

(b). Four input AND gate implemented in tree structure

Figure 2.3: Apply logic restructuring to reduce switching activity

Logic gate restructuring studies how to arrange logic gates and their input signals to reduce switching activities. It works at the logic-level. An example of this method is shown in Figure 2.3. In this example, by using two input AND gate, we have two ways to build a four input AND gate, the chain structure (a), and the tree structure (b). These two setups result in an identical logic function, but different switching patterns. Assuming all primary input (A, B, C and D) have equal chance to be 1 and 0, then the possibility that the logic value at each gate's output is calculated [1] and marked in Figure 2.3. In this example, the chain implementation has a lower overall switching activity than the tree implementation for random inputs ($\frac{7}{64} < \frac{3}{16}$) when the glitching effects are ignored [2]. The reason is that each gate in a chain has a lower probability of having a 0-1 transition than its predecessor since the probability depends on those of all its predecessors.

Clock gating, at the architectural-level, is one straightforward but effective method

[1] The transition probability of a logic gate can be calculated by $\frac{N_0 \cdot N_1}{2^{2N}}$, where N is the number of input. N_0 and N_1 are the number of *zero* entries and *one* entries in the output column of a gate's truth table, respectively [92].

[2] This is a simple example to demonstrate the influence of circuit topology on a circuit's switching activity when glitching effects are *ignored*. However, when the timing behavior are considered, tree topology will have lower (or even no) glitching activity and less propagation delay than the tree topology, since the signal paths are balanced to all the gates [92].

to reduce a circuit's switching activity. This technique simply disables the clock signal of those components which are not currently in use. By applying the clock gating, the switching activities of flip-flops within a component as well as the components along the fanout data path can be reduced.

DVS, at the system-level, is one of the most effective methods to minimize the dynamic power due to the convex relationship between the dynamic power and the supply voltage level. For instance, the dynamic power consumption can be reduced to one fourth of the original value if we decrease the supply voltage by half (assuming the frequency is unchanged). The rationale behind this technique is to operate the circuit "just fast enough", so that the workload can be finished at the predefined deadline or certain QoS can be guaranteed.

Static Power Reduction Techniques

Reducing the supply voltage, indeed, helps to minimize the dynamic power. It also adversely affects the system performance. In fact, logic gates become less responsive because the scaled V_{dd} weakens the driving current, which in turn increases the gate's propagation delay. Therefore, in order to prevent performance loss while at the same time minimize dynamic power consumption, one has to scale down the threshold voltage of a transistor simultaneously [92]. Unfortunately, the reduced transistor threshold voltage significantly increases the leakage current. According to [84], every 100mv decrease of the threshold voltage results in a 10 times increase of the leakage current. And this leakage mechanism is further exacerbated as the transistor feature size continues to shrink.

A number of circuit and architecture-level leakage minimization techniques are proposed. Such techniques include multiple-threshold voltage transistor [49, 25, 13,

59, 95, 99], body bias control, dual-threshold voltage cells, and input vector control [39, 44].

Multiple-threshold voltage transistor, also known as sleep transistor, is an effective way to reduce the leakage power when a circuit is in the standby state. According to this method, all functional units are implemented by using low V_t transistors, which are fast but leakage prone. Between functional units and the ground, a high V_t transistor, or sleep transistor is inserted. In the active state, the sleep transistor is turned on to allow normal operation of a circuit. If functional units are in the standby state, the sleep transistor is turned off to cut off the leakage current path to the ground. Inserting high V_t sleep transistor, however, can degrade the circuit performance since it reduces the magnitude of the driving current. Therefore, designers must carefully adjust the size of the transistor to gain a balance between leakage reduction and circuit performance.

Body bias control is another concept to reduce the leakage during the idle state. The key idea is to bias the source terminal of an "off" transistor in order to exponentially reduce the leakage current of a device. Based on this method, a positive bias voltage is applied during the standby state to the source terminal. By doing so, the threshold voltage of a device is raised and as a result, the transistor is turned off more strongly to reduce leakage current.

Dual-threshold voltage cells can be used to reduce the leakage power consumption in the *active* mode. According to this method, a circuit is partitioned into high and low threshold voltage gates. Low V_t transistors are implemented in critical paths to maximize the performance while high V_t transistors are assigned to gates

along non-critical paths to minimize the leakage current. A trade off between the performance and the leakage reduction needs to be found. The effectiveness of leakage minimization can be reduced if there exist multiple critical paths, so that only leaky, low V_t transistors can be implemented without sacrificing circuit's performance.

Minimum leakage vector is based on the fact that the leakage current of a logic gate is a strong function of its input value. As reported in [10], the ratio between the maximum and the minimum leakage of a circuit can be as high as 6. The reason is that the input values affect the number of "off" transistors in a logic gate. There are certain input patterns exist that can maximize the effective resistance between the V_{DD} and the ground, and as a result, minimize the leakage current.

As we can see, the aforementioned low-level techniques are effective to reduce the leakage. However, leakage current varies not only with the logic topology and transistor parameters such as gate length, oxide thickness and threshold voltage, but also with the supply voltage and the operating temperature. This presents a unique advantage and opportunity for dealing with the leakage problem at the system-level.

2.3 Thermal Management

In previous section, we introduced the sources of power consumption in digital ICs as well as some state-of-the-art technologies to reduce both the dynamic and leakage power consumption. As we know that the exponentially increased power has directly translated into high temperature, which negatively affects a system's cost, performance and reliability. Therefore, as the semiconductor technology continues to scale, the thermal issue is becoming a critical problem to be studied. In this section, we introduce the background of the thermal-aware design. We first discuss the importance

of thermal management. Then, we introduce a number of existing works in this area.

2.3.1 The Need for Thermal Management

The aggressive semiconductor technology scaling has been pushing the device feature size into the deep sub-micron region. As a result, the chip power density has been doubled every two to three years [4, 19, 101]. One immediate consequence of this elevated power consumption is the exponentially raised heat density. As we introduced in Chapter 1 that high temperature can adversely affect the computing system in various ways. Without proper solutions, the high-temperature issue may threaten to slow down the advancement of the entire semiconductor industry.

Although the high system temperature results from the high power consumption, a low power or power-aware design technique alone, cannot be directly applied to solve the thermal-aware design problem [101]. As evidenced in [101], one effective low-power technique may have little or no effect on operating temperature due to the spatial and temporal non-uniformity of the power density and hotspot. Therefore, thermal management and power management are closely related but distinctly different problem areas. And thermal management techniques, at every design levels, are worth to be investigated and urgently needed.

2.3.2 Thermal-Aware Design Techniques

In this subsection, a number of existing works related to thermal-aware design, at different design abstraction levels are introduced.

Packaging-Level Approach

Early research on thermal-aware design techniques are mainly targeting at the packaging-level. These works include but not limited to the study on how to design thermal

package with high heat-removal capability, high-performance fan, advanced thermal conducting materials, circuit board arrangement and etc. Conventionally, thermal packages are designed for worst-case scenarios to sustain the hottest spot a chip could reach.

However, as the chip power as well as the heat density continue to increase exponentially, processor packages are prohibitively expensive to be designed for worst-case scenarios. Instead, it has been suggested [22, 46] that the package should be designed for the average-case to reduce the packaging cost. If the heat dissipated by a processor when running an application exceeds the heat-removing capacity that the package can provide, some alternative dynamic thermal management (DTM) mechanisms should be engaged. Therefore, thermal-aware design techniques at architectural and system-levels, have been intensively studied in recent years.

Circuit-Level Leakage/Temperature Modeling

As we mentioned earlier, the leakage/temperature dependency plays an important role in the development of power and thermal-aware design techniques. Therefore, one of the most important topics in this field, is to investigate in depth, how exactly leakage and temperature interact with each other.

Based on the circuit-level analysis, a complex relationship between the leakage and temperature is established by [65, 120], where the leakage current is formulated as

$$I_{leak} = I_s \cdot (\mathcal{A} \cdot T^2 \cdot e^{((\alpha \cdot V_{dd} + \beta)/T)} + \mathcal{B} \cdot e^{(\gamma \cdot V_{dd} + \delta)}), \tag{2.2}$$

where I_s is the leakage current at certain reference temperature and supply voltage, T is the operating temperature, V_{dd} is the supply voltage, $\mathcal{A}, \mathcal{B}, \alpha, \beta, \gamma, \delta$ are empirically determined technology constants. By using these relations, many practical power and thermal analysis tools are developed, for example "HotSpot" [8, 101], which can be

used to simulate and study a processor's thermal phenomena at the architectural-level.

Architectural-Level Thermal Modeling

By using the well known duality between the heat transfer and the electrical current flow (as shown in Table 2.1), Skadron et al. [8, 101] proposed an architectural-level thermal analysis tool called "Hotspot".

Table 2.1: Duality between thermal and electrical quantities

Thermal Quantity	Electrical Quantity
Power consumption: P (W)	Current flow: I (A)
Temperature: T (^oC)	Voltage: V (V)
Thermal resistance: R $(^oC/W)$	Electrical resistance: R (Ω)
Thermal capacitance: C $(J/^oC)$	Electrical capacitance: C (F)

Given the floorplan of the functional units within a processor as well as some other physical properties, such as dimensions and materials, "Hotspot" can generate a three dimensional RC network to model the heat transfer (Figure 2.4 (adopted from [101])). Each functional unit on the chip is represented by one or more nodes within the RC network. Based on this RC network and basic circuit laws, a differential equation system can be established. In these equations, R and C are constants, which represent the physical characteristics of the chip, e.g. the thermal resistance and the thermal conductance. Current sources are used to model the power consumption of a given functional unit. The unknowns of these equations are the node voltages, which denote the temperature values of those corresponding locations. Then, if the power profile (current change) when running an application is given, by solving the differential equations, the corresponding temperature variation (voltage change) can

be obtained.

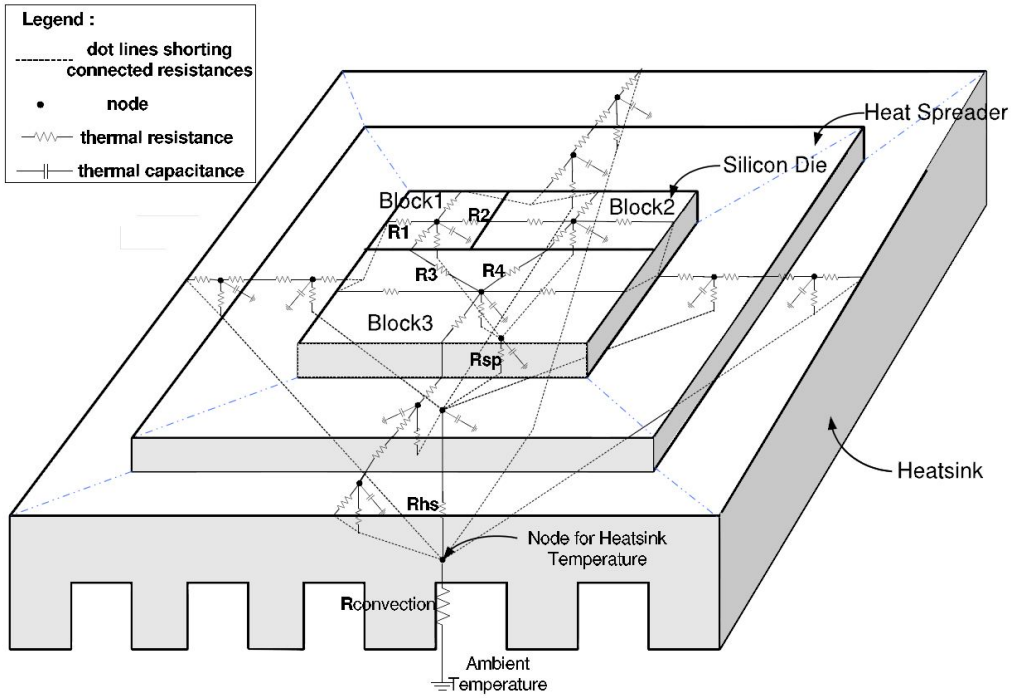


Figure 2.4: Using RC network to model a processor's heat transfer [101]

According to [101], an effective architecture-level thermal model must be simple enough to allow architects to reason about thermal effects; detailed enough to model dynamic temperature change within different functional units; and computationally efficient for use in a variety of architecture simulators. "Hotspot" is one of such tools and has already become the most popular one that has been widely used in the research and development of power/thermal-aware design techniques.

Architectural-Level Approaches

Dynamic thermal management refers to a range of possible hardware and software strategies which work dynamically, at run-time, to control the operating temperature of a chip [22]. At the architectural-level, a number of DTM approaches are proposed recently, such as task migration [67], global clock gating [46], fetch toggling [22] and

decode throttling [101].

Task migration proposes to modify the architecture and the floorplan of a processor and add spare functional units to some cold areas of a chip. When thermal emergency occurs at the primary unit, the task execution can be migrated to a spare unit. One important trade-off needs to be considered that hot units tend to run even hotter when adjacent whereas they introduce additional communication latency if too far away. The dual-pipeline processor proposed in [67] could be considered as one example of migrating computation. However, in this scheme, the secondary pipeline is not designed for high-performance. Therefore, considerable performance losses are expected in the case of thermal emergency.

Clock gating is a widely used technology that cuts off the clock signal when the temperature exceeds a predefined threshold and allows normal operation when the temperature drops down.

Fetch toggling is similar to clock gating method. Instead of cutting off the clock signal, it throttles the instruction cache when the temperature emergency is detected. This method can effectively reduce the processor's fetch bandwidth and thus cool down the temperature.

These architectural-level DTM techniques introduced above are proved to be effective to maintain the processor's temperature under a predefined threshold [101, 22]. However, they are reactive in nature. These mechanisms are triggered whenever the system temperature reaches a predefined limit. Therefore, unexpected performance penalty can be incurred by different degrees as we turn off, slow down or migrate the

task execution in between the functional units. As a result, these techniques can only be applied for applications without hard deadlines.

2.4 Power/Thermal-Aware Scheduling

At the system-level, speed scheduling studies the problem on how to apply advanced DVS and DPD mechanism to dynamically adjust the speed level, e.g. clock frequency and supply voltage, of a processor when executing a set of real-time tasks so that the workload completion can be guaranteed while some other design metrics are achieved. Working at the system-level, a unique advantage is that it has the knowledge of both the characteristics of the applications and the availability of the hardware resources. Therefore, it is considered as one of the most effective power and thermal management techniques.

Early research on speed scheduling problem are mainly focused on reducing the dynamic power and energy, i.e. the predominant component of the overall power and energy consumption. By taking advantage of the convex relationship between the dynamic power and supply voltage, a number of methods (e.g. [64, 112]) were proposed to lower down the processor's supply voltage and working frequency. The processor speed is reduced to a minimum so that the workload can be finished just before the "deadline".

As the leakage becomes more prominent, it is no longer optimal to use all the timing slack and bring the speed down to the minimum simply because the saved dynamic energy might be overweighed by the increased leakage. With this fact in mind, a few techniques are proposed to minimize the overall energy. For example, Jejurikar et al. [57] proposed a scheduling technique, based on the so called *critical speed* to balance the dynamic and leakage energy consumption, with the goal to minimize the overall energy consumption. These approaches (e.g. [30, 57]) assume

that the leakage current is constant. However, as semiconductor technology ventures into the DSM domain, the leakage/temperature dependency becomes too significant to be ignored. As shown in [51], a power/thermal-aware design technique simply becomes ineffective if the interdependence of leakage and temperature is not properly addressed.

Recently, by taking into consideration the critical leakage/temperature dependency, a great number of literature are published on solving the power/thermal-aware scheduling problems. Different criteria can be used to sort these existing works into categories. For example, based on the target platforms, we have techniques proposed for single-core [110, 119], multi-core [47, 48], or even 3D multi-core platforms [71, 121]. Based on the task models, we have tasks with stochastic [70] or deterministic workload [27, 118]. Based on the timing requirement, we have soft real-time [116] or hard real-time scheduling [28, 110]. We also have on-line approaches [116, 75] and off-line approaches [52, 119] depending on during which stage the scheduling decisions are made. Based on the design objectives, there are methods proposed for temperature minimization under the timing constraint [28, 56], energy minimization under the timing or temperature constraint [116, 16] or throughput maximization under the peak temperature constraint [27, 118].

In this dissertation, we are interested in the problem of developing scheduling algorithms for single-core platform, hard real-time system with deterministic workload, to achieve various design objective, e.g. (*i*) temperature minimization, (*ii*) energy minimization and (*iii*) throughput maximization.

In what follows, we introduce our system models in Chapter 3. The temperature minimization, energy reduction and throughput maximization scheduling problems are tackled one by one from Chapter 4 to 6. Finally, we conclude this dissertation in Chapter 7.

CHAPTER 3

SYSTEM-LEVEL POWER AND THERMAL MODELS

To study the power/thermal-aware scheduling problem, the first priority is to effectively model the power consumption and thermal behavior of a system. As discussed before, one key aspect of system modeling is to capture the interdependency between the leakage power and the temperature. While previous circuit-level research results can capture the leakage/temperature dependency accurately, they are too complex and thus ineffective in high-level, e.g. system-level, design and analysis.

In this chapter, we present a set of system-level models we used throughout this dissertation. Specifically, we study a large spectrum of leakage power models that are able to account for the leakage/temperature dependency, and at the same time, are simple enough and suitable for system-level design. We analyze and compare the trade-off between the complexity and accuracy of these models empirically. Our experimental results strengthen the important role that the leakage power consumption plays in the electronic system design as the transistor size continues to shrink. More importantly, our results highlight the fact that it is vital to take the leakage/temperature and leakage/supply voltage dependency into considerations for system-level power/thermal-aware design.

3.1 Preliminaries

For system-level thermal analysis, we adopt the widely used RC thermal model to capture the temperature dynamic of a processor (e.g. [15, 31, 90, 118]). Based on the circuit depicted in Figure 3.1, we have

$$RC\frac{dT(t)}{dt} + T(t) - RP(t) = T_{amb}, \tag{3.1}$$

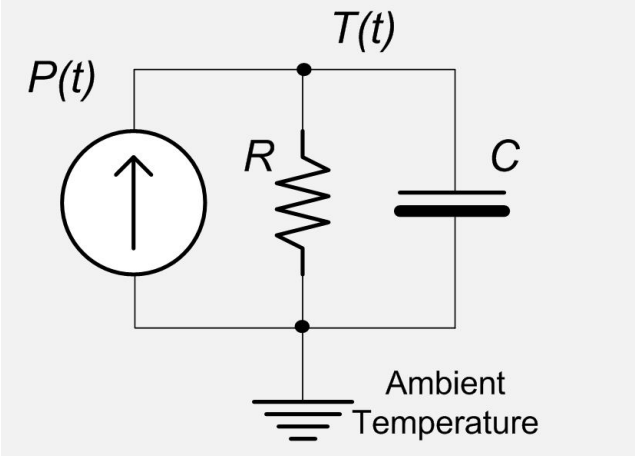


Figure 3.1: System-level thermal model for single core processor

where T_{amb} is the ambient temperature, $P(t)$ denotes the power consumption (in $Watt$) at time t, and R, C denote the thermal resistance (in $^oC/W$) and thermal capacitance (in $J/^oC$), respectively. By scaling T such that T_{amb} is zero, we have

$$\frac{dT(t)}{dt} = aP(t) - bT(t), \tag{3.2}$$

where $a = 1/C$ and $b = 1/RC$.

Assuming that all processor running modes are safe and do not cause processor temperature to "run away", i.e. the scenario when the processor temperature increases indefinitely, the temperature becomes gradually stable if the processor runs in one mode long enough. Consider the processor's thermal steady state, we have

$$\frac{dT(t)}{dt} \to 0. \tag{3.3}$$

As a result, from equation (3.2), when the processor temperature becomes stable at T_{max}, we have

$$aP(v) - bT_{max}(v) = 0, \tag{3.4}$$

or

$$\triangle T_{max}(v) = \frac{a}{b} \triangle P(v). \tag{3.5}$$

Equation (3.5) shows the relationship between the estimation error of the stable temperature and overall power consumption. As an example, for the conventional air cooling option, we have $Rth = 0.8^oC/W$ and $Cth = 340J/^oC$ [101], and thus we have

$$\triangle T_{max}(v) = 0.8 \triangle P(v). \tag{3.6}$$

From equation (3.6), to ensure an accurate thermal analysis result, we need to model the processor power consumption accurately.

The processor considered in this research is assumed to be able to run in n different modes, with each mode characterized by (v_i, f_i), $i = 0, 1, ..., n-1$, where v_i is the supply voltage and f_i is the working frequency in mode i [1]. We assume that $v_i < v_j$, if $i < j$. We also assume that the processor speed is in proportion to the supply voltage. In what follows, we use processor speed and supply voltage interchangeably.

Given a supply voltage level v, the processor power consumption is composed of two parts, i.e. dynamic P_{dyn} and leakage P_{leak},

$$P = P_{dyn} + P_{leak}. \tag{3.7}$$

The dynamic power consumption is independent to the temperature and can be formulated as $P_{dyn} = C_{load} f v_k^2$, where C_{load} is the equivalent parasitic capacitance, f is the clock frequency and v is the supply voltage. Since the working frequency is in proportion to the supply voltage level, we can further assume that $P_{dyn} = C_2 v_k^3$ [92], where v_k is the kth supply voltage level and C_2 is a constant.

[1] In this research, we only consider n distinctive supply voltage and frequency pairs. The scenarios that one supply voltage level supports different frequencies or one frequency runs under different supply voltages are not considered.

The leakage power consumption, on the other hand, depends on temperature and can be formulated as:

$$P_{leak} = N_{gate} \cdot I_{leak} \cdot v_{dd}, \tag{3.8}$$

where N_{gate} represents the number of gates, v_{dd} is the voltage level, and I_{leak} is the leakage current. I_{leak} varies with both temperature and supply voltage and can be calculated by using equation (2.2). While using equation (2.2) can accurately estimate the leakage current, with relative error less than 1% [65], it is too complicated to be used for high level system analysis due to its high-order and exponential terms. Liu et al. [72] found that using the linear approximation can estimate the leakage with a reasonably good accuracy, i.e. with error within 1% using the piece-wise linear function or less than 5.5% using single linear function.

In what follows, we derive six different linear leakage models, and study the complexity/accuracy trade-off in power and thermal-aware system-level analysis.

3.2 Linear Leakage/Tempearture Models

Since leakage power consumption depends on both temperature and supply voltage, a general polynomial model to simplify the leakage/temperature dependency can be formulated as

$$P_{leak} = \mathcal{C}_0 + \mathcal{C}_1 T + \mathcal{C}_2 v_{dd} + \mathcal{C}_3 T v_{dd}, \tag{3.9}$$

where $\mathcal{C}_0$, $\mathcal{C}_1$, $\mathcal{C}_2$, $\mathcal{C}_3$ are constants. In this section, we develop six leakage models, Model 1 to Model 6, based on equation (3.9), to simplify the leakage/temperature relationship.

Note that the leakage model, described in equation (3.9), is one of the linear-approximated leakage models. However, the methodology we used to determine the constants of a linear model is based on the first-order polynomial curve fitting, which

produces two coefficients to describe a single line. By using this method, we are unable to determine the appropriate values of the four constants in equation (3.9). And thus, we cannot really take advantage of the additional coefficients provided by equation (3.9) to further improve the accuracy of the linear approximation. Therefore, we exclude equation (3.9) from our leakage model comparison.

- **Model 1** can be formulated as follows:

$$P_{leak}(i) = C_0, \tag{3.10}$$

  where $P_{leak}(i)$ denotes the leakage power consumption with processor running in mode i. In this model, the leakage power is assumed to be constant, and depends on neither temperature nor supply voltage. This is the simplest leakage model.

- **Model 2** can be formulated in equation (3.11), that

$$P_{leak}(i) = C_0 + C_1 T. \tag{3.11}$$

  This model assumes that the leakage power varies with temperature linearly but not with supply voltage. This model is adopted in a number of recent studies such as [31].

- **Model 3** is formulated in equation (3.12).

$$P_{leak}(i) = C_0(i) v_i. \tag{3.12}$$

  In contrast to Model 2, this model assumes that the leakage power changes linearly with supply voltage but not with temperature.

- **Model 4** is formulated in equation (3.13), that

$$P_{leak}(i) = C_0(i)v_i + C_1T. \tag{3.13}$$

This model improves upon Model 2 and Model 3 by assuming that the leakage power varies not only with supply voltage but also with temperature. Note that, C_1 in equation (3.13) is a constant independent of supply voltage levels, e.g. i. Therefore, the leakage power consumption estimated based on equation (3.13) increases at the same rate with respect to temperature, regardless of the processor running mode.

- **Model 5** also assumes that the leakage power consumption varies with both temperature and supply voltage, as formulated in equation (3.14):

$$P_{leak}(i) = (C_0(i) + C_1(i)T)v_i. \tag{3.14}$$

This model is first proposed in [90]. The difference between Model 4 and Model 5 is that Model 5 assumes the leakage power varies at different rates with temperature based on different supply voltages, while Model 4 assumes a uniform rate.

- **Model 6** uses a piece-wise linear function rather than a single linear function to approximate the leakage/temperature relationship, as formulated in equation (3.15),

$$P_{leak}(i) = \begin{cases} (C_{00}(i) + C_{10}(i)T)v_i, & T \leq T_z \\ (C_{01}(i) + C_{11}(i)T)v_i & T > T_z \end{cases} \tag{3.15}$$

Specifically, equation (3.15) adopts a piece-wise linear function consisting of two linear functions, with T_z as the conjunction point. When the temperature is

lower than T_z, the first linear function is used to estimate the leakage power consumption, or the second one otherwise.

For the sake of comparison, we call the leakage model described by equation (2.2) as Model 0. Table 3.1 summarizes all seven leakage models. It is not difficult to see from Table 3.1 that, while different leakage models (i.e. Model 1 to Model 6) have different complexities, they all have greatly simplified the complex leakage/temperature relationship as described in Model 0 and hence more suitable for system-level analysis.

The question now becomes how accurate these models are when used for leakage power estimation at the system-level, and how effective they can be in system-level design of power and thermal management techniques. It is difficult to compare the accuracy of these models since the constants in Table 3.1 are obtained through curve-fitting methods rather than from certain analytical formulas. In the next section, we launched a series of experiments to answer these questions.

Table 3.1: Leakage model definition

Model #	Formula	Description
Model 0	$P_{leak} = V_{dd} \cdot (I_s \cdot (\mathcal{A} \cdot T^2 \cdot e^{((\alpha \cdot V_{dd} + \beta)/T)} + \mathcal{B} \cdot e^{(\gamma \cdot V_{dd} + \delta)}))$	Non-linear leakage model
Model 1	$P_{leak}(i) = C_0$	Constant leakage model
Model 2	$P_{leak}(i) = C_0 + C_1 T$	Leakage depends on temperature only
Model 3	$P_{leak}(i) = C_0(i) v_i$	Leakage depends on supply voltage only
Model 4	$P_{leak}(i) = C_0(i) v_i + C_1 T$	Leakage depends on both supply voltage and temperature, but leakage increases with temperature uniformly.
Model 5	$P_{leak}(i) = (C_0(i) + C_1(i) T) v_i$	Leakage varies with supply voltage and temperature non-uniformly.
Model 6	$P_{leak}(i) = \begin{cases} (C_{00}(i) + C_{10}(i) T) v_i, & T \leq T_z \\ (C_{01}(i) + C_{11}(i) T) v_i & T > T_z \end{cases}$	Two-segment piece-wise linear model. Each segment is obtained by the same method with Model 5

3.3 Performance Evaluation of Different Leakage Models

We conducted two sets of experiments to validate the leakage models introduced above. In the first set of experiments, we compared the leakage power consumptions at different temperatures and different supply voltages by using different models, i.e. Model 1 to Model 6. By using Model 0 as the baseline model, we compared the average and maximal estimation errors achieved by different linear leakage models. This set of experiments help to identify the accuracy of each linear leakage model.

To study how a proposed leakage model may impact on the system-level power and thermal analysis, we launched the second set of experiments, in which we compared the peak temperature estimated by different leakage models for a processor running with a single processor speed.

3.3.1 Experiment Setup

In our experiments, we built our processor model based on the technical parameters drawn from the 65nm IC technology [65]. We assume that the supply voltage can change from 0.6v to 1.3v with step size of 0.05v, and thus the processor can work in total $k = 15$ modes.

We set the frequency for each mode according to the formula [65]

$$f = \frac{1}{delay} = \frac{(V_{dd} - v_t)^{\mu}}{V_{dd}T^{\eta}} \times 4.2824 \times 10^{14}, \tag{3.16}$$

with $\mu = 1.19$, $\eta = 1.2$ and T is set to the highest temperature as 100^oC, and then normalize the frequency to the highest one. The number of gates, i.e. N_{Gate} in equation (3.8), is set to 1×10^6.

For the thermal constants, we selected $R_{th} = 0.8K/W$, $C_{th} = 340J/K$, and the ambient temperature was set to 25^oC [102].

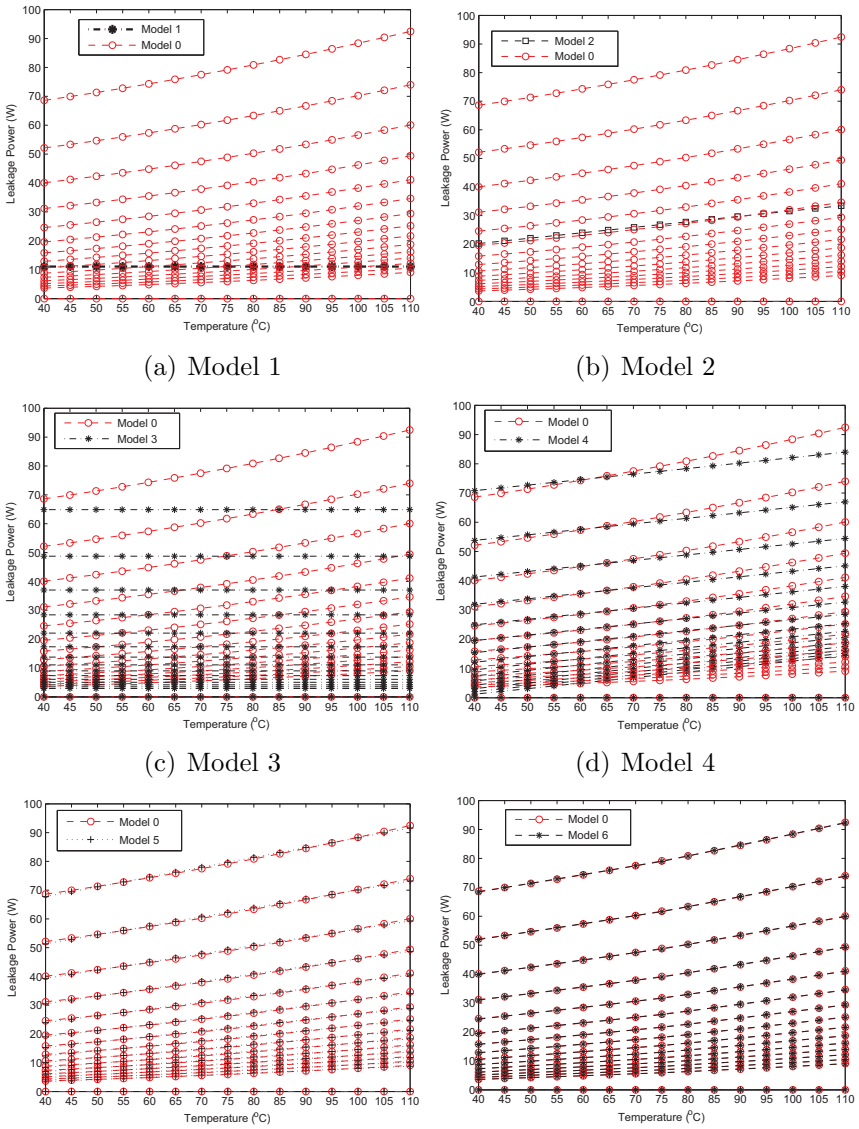


Figure 3.2: Estimated leakage power consumption by different leakage models under different temperature and supply voltage.

Table 3.2: Technical parameters of the circuit-level leakage model based on the 65nm IC technology

$I_{leak} = I_s \cdot (\mathcal{A} \cdot T^2 \cdot e^{((\alpha \cdot V_{dd} + \beta)/T)} + \mathcal{B} \cdot e^{(\gamma \cdot V_{dd} + \delta)})$	
$\mathcal{A}$	1.1432e-12
$\mathcal{B}$	1.0126e-14
α	466.4029
β	-1224.74
γ	6.28153
δ	6.9094

The six leakage power models discussed in Section 3 were constructed. The constants in each models were determined based on the leakage power consumption calculated by using Model 0, at different temperatures (from 40 ^oC to 110 ^oC with step size of 5^oC) and supply voltage levels. The parameters we used in Model 0 is based on the data reported in [65] (as shown in Table 3.2). In Model 1, the constant C_0 is set as the leakage power at the ambient temperature. To obtain the constants C_0 and C_1 in Model 2, we first used linear approximation for leakage power consumption at each supply voltage v_i and obtained a pair of parameters of $C_0(i)$ and $C_1(i)$. We then took the average value as the C_0 and C_1. The constants in Model 3, 4 and 5 were determined by linear approximation methods based on the leakage power consumption at different temperature values and different supply voltage levels. In Model 7, we picked the middle point, $Tz = 75^oC$, as the conjunction temperature. We then used curve-fitting to determine the corresponding constants in equation (3.15).

3.3.2 Leakage Power Estimation

In the first set of experiments, we collected the estimated leakage power consumptions by each model at different supply voltages and temperatures. We used these results to compare with those achieved by Model 0, as plotted in Figure 3.2. In the meantime, we also collected the absolute estimation error (AEE) and relative estimation error (REE) of each model, with the average and maximal values summarized in Table 3.3.

Specifically, the absolute error (i.e. $AEE(M_i)$) and relative error (i.e. $REE(M_i)$) are defined as follows:

$$AEE(M_i) = |P_{leak}[M_i] - P_{leak}[M_0]|, \quad (3.17)$$

$$REE(M_i) = \frac{AEE(M_i)}{P_{leak}[M_0]}, \quad (3.18)$$

where $P_{leak}[M_i]$ is the leakage power calculated by using Model i. The average and maximal values in Table 3.3 were obtained among all k different running modes. That is,

$$AEE(Avg) = \frac{\sum_{i=1}^{k} AEE(M_i)}{k} \quad (3.19)$$

$$AEE(Max) = \max_{i=1}^{k} AEE(M_i) \quad (3.20)$$

Table 3.3: The absolute (AEE) and relative (REE) estimation errors of leakage power consumption by different leakage models.

Model #	M1	M2	M3	M4	M5	M6
AEE(Avg)	28.6	16.1	7.5	1.4	0.24	0.06
AEE(Max)	81	60	27.6	7	0.84	0.2
REE(Avg)	55%	99%	33%	9.5%	1.3%	0.3%
REE(Max)	200%	450%	67%	65%	7.0%	1.5%

From Figure 3.2 and Table 3.3, we can see that leakage models without considering the leakage/temperature dependency or the leakage/supply voltage dependency can lead to significant errors. As illustrated in Figure 3.2(a) and 3.2(b), Model 1 and 2 intend to estimate leakage power consumption by using one single line instead of a group of lines, as other models, and thus cause large estimation errors. Even though Model 3 takes the leakage/supply voltage dependency into account, it ignores the leakage/temperature dependency. The estimation errors are still substantial. Therefore, these three models can be applied only when the supply voltages and

temperature vary within a relatively small range. For example, when we limited the processor supply voltage range between $[1.05, 1.1]V$, the maximum REE by model 3 can be cut to less than 8%.

When considering both temperature and supply voltage dependency, the accuracy of leakage models (e.g. Model 4, 5, and 6) can be dramatically improved as shown in Figure 3.2(d), 3.2(e), 3.2(f) and Table 3.3. Even though at some extreme cases, the maximum relative estimation error of Model 4 (i.e. 65%) is close to that of Model 3 (i.e. 67%), Model 4 is still considered as a much more accurate model than Model 3 for average cases, as shown in Table 3.3 and Figure 3.2(d). When we further limited the supply voltage level between [0.85-1.05]V, the maximum relative error is reduced to 16% and average error becomes 4.3%. Model 5 is the most accurate single linear approximation model according to our experimental results. We found that the maximal relative estimation error appeared at the lowest supply voltage and temperature. This is because the absolute values of the leakage power at these points are small. And a small difference in absolute value can cause a large relative error. To further improve the accuracy, using the piece-wise linear leakage model (i.e. Model 6) is a viable solution. As shown in Figure 3.2(f), the 2-segment piece-wise linear model almost perfectly matches the non-linear leakage power. The average relative error is 0.3% and the maximum relative error is only 1.5%.

3.3.3 Peak Temperature Estimation

To study how different leakage models may affect the thermal-aware system-level analysis, we conducted the second set of experiments. We simulated the scenario when the processor runs at a constant speed long enough until its temperature becomes stable (i.e. temperature variance within 0.001^0C). We collected the peak temperatures estimated based on each model under different supply voltages. The results are

Table 3.4: The absolute (AEE) and relative (REE) estimation errors of peak temperature by different leakage models.

Model #	M 1	M 2	M 3	M 4	M 5	M 6
AEE(Avg)	15	14	9	3.6	1.19	0.57
AEE(Max)	59	47	16	6	1.8	0.9
REE(Avg)	18%	27%	6.6%	3.4%	2.9%	1.5%
REE(Max)	50%	40%	16%	11%	4.8%	2.4%

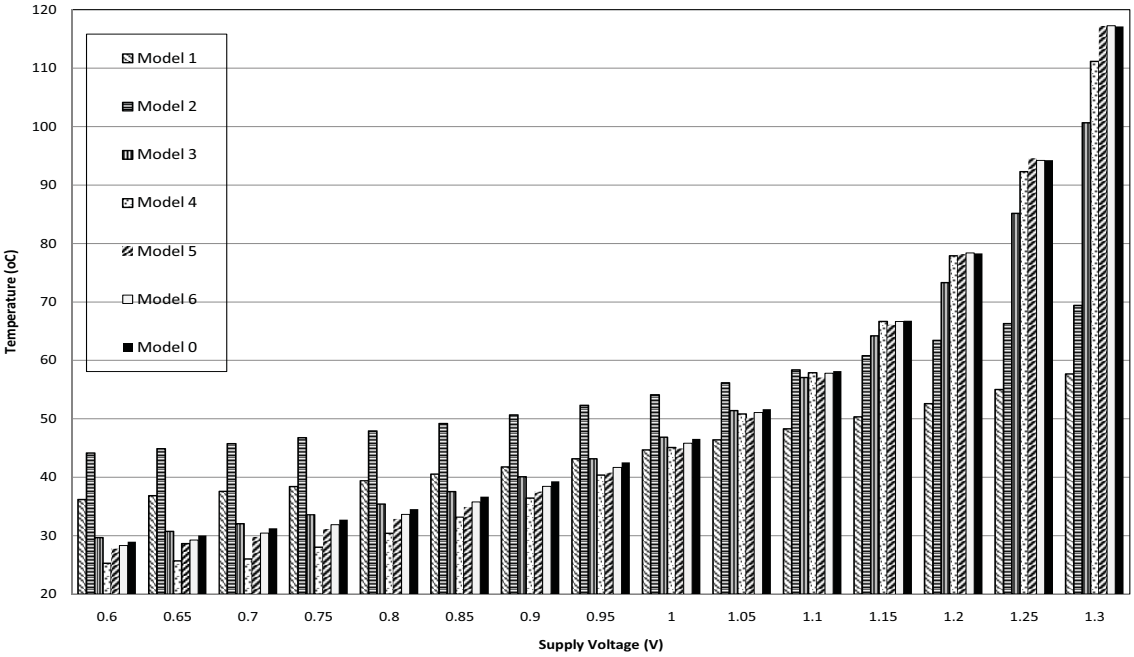


Figure 3.3: Estimated peak temperature for different leakage models

depicted in Figure 3.3. Similarly, we collected the absolute estimation error (AEE) and relative estimation error (REE) of each model and filled in Table 3.4.

As we can see in Figure 3.3, the peak temperature values calculated based on different leakage models demonstrate dramatic differences. When modeling the leakage as a constant, Model 1 can lead to a temperature discrepancy of 15^oC in average, and as much as 59^oC. Even though Model 2 and Model 3 take into account the leakage/temperature and leakage/supply voltage, respectively, the peak temperature discrepancies are still 14^oC and 9^oC in average, and can be as high as 47^oC and 16^oC, respectively. It is reported that a 10^oC increase in temperature can result in as

much as 50% reduction in the component's life span [113]. Therefore, the large error margins by Model 1, Model 2, and Model 3, seem to make them inappropriate in system-level thermal analysis. On the other hand, the estimated peak temperatures from Model 4, 5, and 6 can match those obtained by Model 0 with much less errors, as shown in Figure 3.3. The absolute error by Model 4 is 3.6^oC in average, and 6^oC at most. The results by Model 5 and Model 6 are very close to Model 0, with less than 1.8^oC of absolute error. Our experimental results strengthen the critical role that the leakage power plays in the system-level analysis. These results also highlight the fact that, in deep sub micron domain, it is not only important but necessary to take the leakage/temperature and leakage/supply voltage dependency into consideration for system-level power/thermal-aware design.

3.3.4 Our System-Level Power / Thermal Model

Based on the results of our empirical study, the leakage Model 5 and Model 6 clearly outperform the rest of the models under test, i.e. Model 1 to Model 4. Although Model 6 does have some advantages over Model 5 in terms of leakage power and steady state temperature estimation, the complexity of the piece-wise linear equation makes it (Model 6) cumbersome to be used in our system-level analysis, especially when we want to formally establish some theorems. Considering the fact that Model 5 has already captured both the leakage/temperature and leakage/supply voltage dependencies while still achieves reasonably good accuracy, throughout this research (unless otherwise specified), we adopt Model 5 as our leakage power model.

Therefore, based on our previous discussion in Section 3.1, the total power consumption of a processor when running at mode k can be formulated as

$$P(k) = C_0(k)v_k + C_1(k) \cdot Tv_k + C_2 v_k^3. \tag{3.21}$$

Based on equation (3.2) and (3.21), when a processor running in mode k, the temperature dynamic can be formulated as

$$\frac{dT(t)}{dt} = A(k) - BT(t), \tag{3.22}$$

where

$$A(k) = a(C_0(k)v_k + C_2v_k^3), \tag{3.23}$$

$$B(k) = b - aC_1(k)v_k. \tag{3.24}$$

Given an interval $[t_0, t_e]$, let the starting temperature be T_0, by solving equation (3.22), the ending temperature can be formulated as below:

$$\begin{aligned} T_e &= \frac{A(k)}{B(k)} + (T_0 - \frac{A(k)}{B(k)})e^{-B(k)(t_e - t_0)} \\ &= G(k) + (T_0 - G(k))e^{-B(k)(t_e - t_0)}, \end{aligned} \tag{3.25}$$

where

$$G(k) = \frac{A(k)}{B(k)}. \tag{3.26}$$

In equation (3.25), by letting $(t_e - t_0) \to \infty$, we have T_e equals $G(k)$, which is called the steady state temperature of the kth speed level. For an initial temperature T_0, if the steady state temperature of the running processor speed level is greater than T_0, the temperature will increase, or, decrease otherwise.

In what follows, we use A_k, B_k and G_k to denote $A(k)$, $B(k)$ and $G(k)$, respectively when there is no confusion. Equation (3.21) to equation (3.26) form the basis of our system-level power and thermal analysis with the leakage/temperature/supply voltage interdependency taken into account.

3.4 Summary

In this chapter, we present the system-level models we used in this research. Specifically, we study a large spectrum of leakage power models that can account for the leakage/temperature dependency, and in the meantime, greatly simplify the complex non-linear relationship implied by previous circuit-level research. We analyze and compare the trade-off between the complexity and accuracy for these models empirically. Our experimental results strengthen the critical role that the leakage plays in the system-level analysis. More importantly, our results highlight the fact that it is vital to take the leakage/temperature and leakage/supply voltage dependency into consideration for high-level power and thermal aware design. Based on our system-level leakage/temperature model, we are able to capture the temperature dynamic as discussed in Section 3.3. These models and formulations form the basis of our study on power/thermal-aware scheduling problems, which will be detailed in following chapters.

CHAPTER 4

SCHEDULING FOR PEAK TEMPERATURE MINIMIZATION

In the previous chapter, we have studied how to model the processor's power consumption, thermal dynamic as well as the crucial leakage/temperature dependency, at the system-level. Based on these models, in this chapter, we investigate how to apply real-time scheduling techniques to solve the peak temperature minimization problem.

The rest of this chapter is organized as follows. Section 4.1 discusses the related work in the field of temperature-aware scheduling, including a brief introduction to the *M-Oscillating* algorithm as proposed in [28], that oscillates high and low processor speeds to minimize the peak temperature of a processor when executing a periodic task set. Section 4.2.2 introduces our non-negligible transition overhead model and how to incorporate the non-negligible overhead into the *M-Oscillating* scheduling algorithm. In section 4.3, we present a searching algorithm to find the optimal m that can lead to the minimized peak temperature. Experimental results are presented in Section 4.4, and Section 4.5 concludes this chapter.

4.1 Related Work

In this chapter, we study the problem on how to apply the real-time scheduling technique to minimize the peak operating temperature of a processor when executing a periodic hard real-time task set. As closely related research, there are large number of literature published recently on how to optimize the operating temperature of a processor by using real-time scheduling techniques. Specifically, in [14, 29], the researchers aim to identify the upper bound of the maximum temperature. While some others (e.g. [14, 34, 26, 94, 111]) intend to minimize the peak temperature or to guarantee that the temperature does not exceed a given maximum temperature

limit when scheduling a task set or a single copy of a task graph. In [69], a thermal-aware scheduling algorithm with stochastic workload is presented to effectively avoid thermal emergencies by reducing peak operating temperature. In [61], Kumar et al. derive a stop-and-go algorithm to schedule a task graph with *just enough* idle period so that the peak temperature is minimized while a given *makespan* constraint can be guaranteed. Jayaseelan et al. [56] study how to appropriately arrange the execution sequence of a task set consisting of tasks with different power and thermal profiles to minimize the peak temperature.

As we discussed in Chapter 3, the critical leakage/temperature dependency plays an important role in the study of thermal-aware scheduling problems. Some researchers (e.g. [15, 117]) apply equation (2.2) directly to capture the leakage/temperature dependency in scheduling algorithm development. There are also a number of other approaches that formulate a temperature-constrained scheduling problem as a convex optimization problem [73, 83, 26]. Even though the leakage/temperature dependency (equation (2.2)) may be incorporated into the convex optimization formulation [73], its computational complexity is very high. Therefore, these approaches can only work at system-level when the design solution space is relatively small.

Efforts have been made to simplify the leakage/temperature dependency. Liu et al. [72] show that using the linear approximation is an effective way to accurately estimate the leakage power over the common operating temperature range of real-time ICs. A number of exiting works (such as [31, 42, 27]) adopt simple leakage/temperature dependency models that assume the leakage current changes linearly *only* with temperature. However, as shown in Chapter 3, the leakage models ignoring the effect of supply voltage can lead to results deviated far away from the actual values.

By incorporating both the leakage/temperature and leakage/supply voltage de-

pendency into analysis. Chaturvedi et al. [28] have proposed a novel scheduling technique, namely *M-Oscillating*, that oscillates between the high and low processor speed to minimize the peak temperature when executing a periodic task set. In this approach, however, the timing penalty that is associated with a processor's mode switching is ignored for simplicity. The apparent discrepancy between this assumption and practical platforms may potentially prevent this method from being useful in real world computing systems.

In this chapter, we propose to extend the *M-Oscillating* by incorporating a more realistic non-negligible transition overhead model into the algorithm.

Based on our system models presented in Chapter 3, the formal problem definition of our research topic in this chapter can be given as follows:

Problem 4.1.1. *Given a hard real-time task set with period p and worst case execution time c, develop a feasible schedule such that the peak temperature can be minimized when the processor reaches the thermal steady state.*

4.2 Peak Temperature Minimization

In this section, we first introduce the concept of the original *M-Oscillating* technique for a processor with negligible transition overhead. We next present how to incorporate the non-negligible transition overhead into this approach.

4.2.1 The Concept of *M-Oscillating*

The key idea of the *M-Oscillating* algorithm is shown in Figure 4.1. Given a two-speed schedule (S_1 and S_2, running for t_1 and t_2 seconds, respectively), an *M-Oscillating* schedule divides the high speed interval and the low speed interval evenly into m sections and run a processor with the low speed and high speed alternatively. Apparently, an *M-Oscillating* schedule will complete the same workload as the original

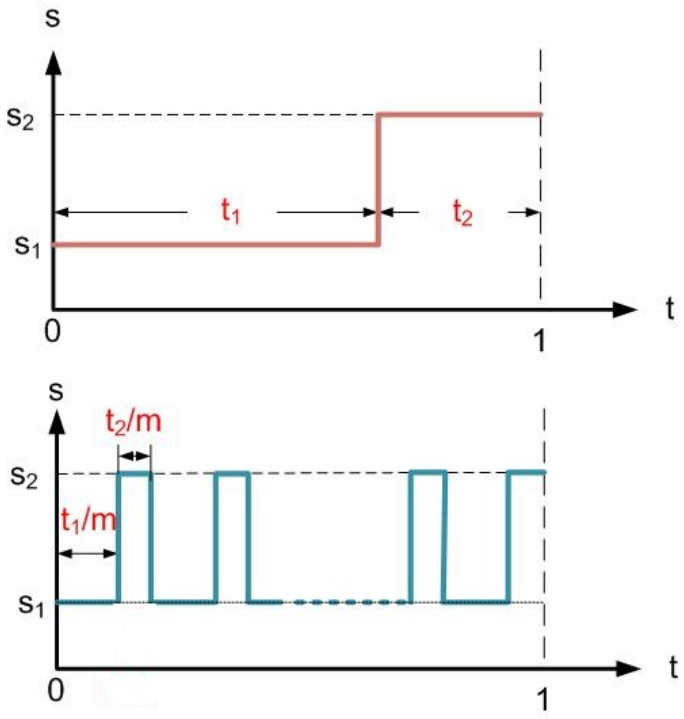
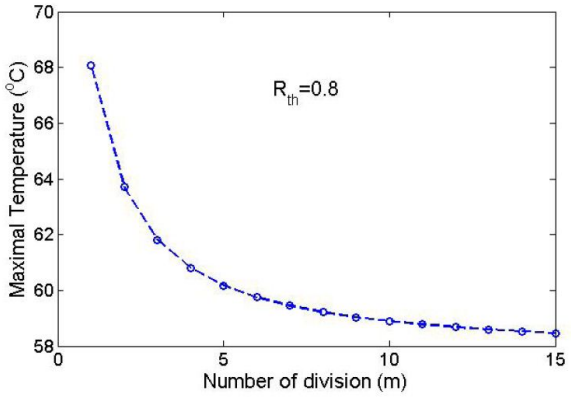


Figure 4.1: A two-speed schedule and its corresponding *M-Oscillating* schedule.

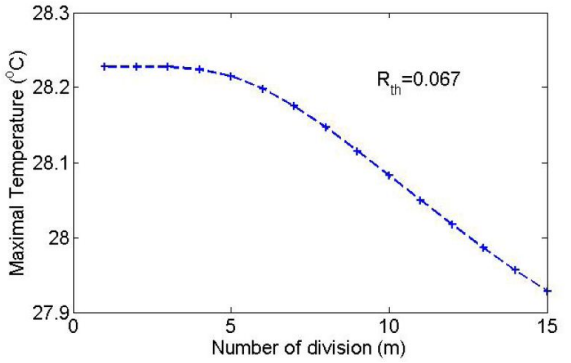
two-speed schedule in one period and thus guarantee the deadline. At the same time, the maximum temperature can be significantly reduced based on experimental results provided in [28].

According to [28], by dividing the high speed interval and the low speed interval each into m equal sections and running them alternatively, an *M-Oscillating* schedule can always reduce the maximum temperature when a processor reaches its thermal steady state. The larger the m is, the lower the maximum temperature becomes.

Simulation results are provided, in Figure 4.2 to validate this conclusion and to prove the effectiveness of the *M-Oscillating* algorithm in terms of peak temperature reduction. In this example, a task with 1000 seconds of execution time are executed by two supply voltages, i.e. $v_1 = 0.85V$ and $v_2 = 1.15V$. The maximal temperature of the processor under different cooling options was recorded and shown in Figure 4.2. As shown in Figure 4.2, the maximal temperature, indeed, drops monotonically



(a) $R_{th} = 0.8$



(b) $R_{th} = 0.067$

Figure 4.2: In an ideal *M-Oscillating* schedule, the peak temperature monotonically decreases with number of divisions

as m increases.

Note that, in [28], the conclusion and its proof are contingent upon several assumptions. One of them assumes that no timing overhead can be incurred during a processor's mode switching, which is not true in the real world scenario. When the overhead is non-negligible, conceivably, there is an optimal value of m to balance the impact of the transition overhead and the potential of *M-Oscillating* algorithm in peak temperature reduction. How to identify this optimal value by incorporating the non-negligible transition overhead model is an interesting problem and will be discussed in the following subsection.

4.2.2 The *M-Oscillating* Considering Transition Overhead

In real world scenarios, when a processor switches its speed level, there is a short time interval (on the order of hundreds of microseconds [27]) during which the clock signal is halted. As a result, the amount of time a processor spends on useful computation is reduced. To compensate this performance loss during the clock halting period, one intuitive way is to increase the speed level of a processor. However, this is not always feasible (e.g. when the high speed is the maximum processor speed). Therefore, in this work, we only change the duration of high and low speed level subject to a workload constraint, i.e. extending the high speed interval and reducing the low speed interval. By doing the workload compensation, an upper bound of m can be identified, beyond which the predefined workload cannot be completed.

We assume that the clock signal will be halted for a short time interval τ during each speed transition. In contrast to the ideal-case two-speed schedule shown in Figure 4.1, the non-ideal-case two-speed schedule is plotted in Figure 4.3 (A). Apparently, in the non-ideal-case, a performance loss of $(S_1 + S_2) \cdot \tau$ is incurred due to *one* speed transition. Thus, in order to counteract this performance loss, one has to extend the high speed interval while reduce the low speed interval by the same time frame δ, as shown in Figure 4.3 (B). The amount of time, i.e. δ, that needs to be taken away from low speed interval can be calculated as

$$\delta = \frac{(S_1 + S_2) \cdot \tau}{S_2 - S_1}. \tag{4.1}$$

From equation (4.1), one can easily notice that δ is a positive number. Therefore, the number of transition m cannot be arbitrarily increased since the low speed duration t_1 in the ideal-case two speed schedule has to be sufficiently large to accommodate m speed transitions. Thus, the maximum allowable m can be calculated by letting $m \cdot (\delta + \tau) \leq t_1$. Then, we have

$$m \leq \frac{t_1}{\delta + \tau}.$$

Obviously, the term $t_1/(\delta + \tau)$ is not necessarily an integer. Therefore, in order to ensure the workload constraint, we set the upper bound of m as

$$m_{Max} = \lfloor \frac{t_1}{\delta + \tau} \rfloor.$$

Based on equation (4.1), the modified non-ideal-case two-speed schedule can be found and the reduced low speed interval t_1' and extended high speed interval t_2' can be calculated by

$$t_1' = t_1 - \tau - \delta,$$

$$t_2' = t_2 - \tau + \delta.$$

By adopting the similar concept, we can find the corresponding non-ideal-case *M-Oscillating* schedule for a given ideal-case two-speed schedule. The procedure is shown the Lemma 4.2.1.

Lemma 4.2.1. *Given an ideal-case two speed schedule with low speed S_1 and high speed S_2 running for t_1 and t_2, respectively. For any value of* m *from 1 to m_{Max}, the corresponding non-ideal-case* M-Oscillating *schedule can guarantee the same workload if the adjusted low speed sub-interval t_1^m and high speed sub-interval t_2^m are calculated by equation (4.2) and (4.3).*

$$t_1^m = \frac{t_1}{m} - \tau - \delta \tag{4.2}$$

$$t_2^m = \frac{t_2}{m} - \tau + \delta \tag{4.3}$$

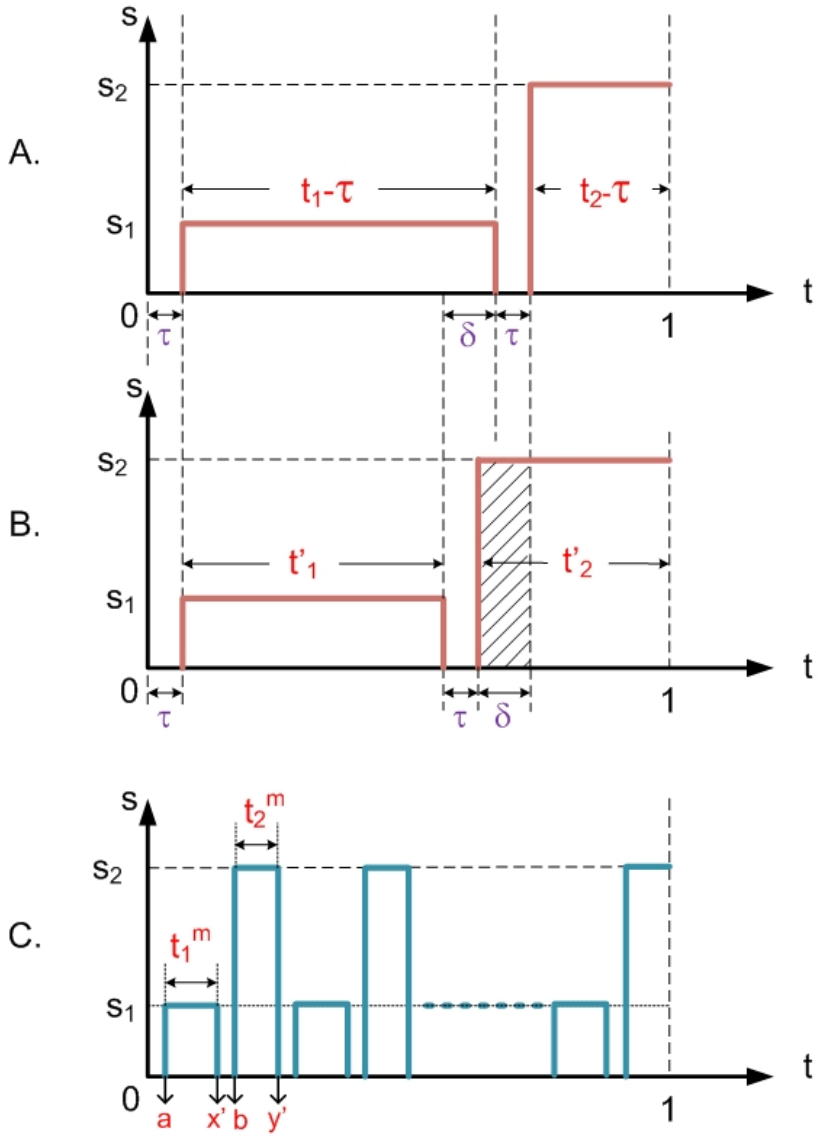


Figure 4.3: A non-ideal two-speed schedule and its corresponding *M-Oscillating* schedule

We next study the thermal characteristic of the non-ideal-case *M-Oscillating* schedule. Previous analysis reveals that the nature of our method to deal with the non-negligible transition overhead is to increase high speed duration while decrease low speed duration. Thus, the performance of a non-ideal-case *M-Oscillating*

technique can be degraded in terms of peak temperature reduction. Intuitively, the maximal temperature of a non-ideal-case *M-Oscillating* schedule is a unimodal function of m. The relationship is simple such that when m is small, the schedule works as expected by running low and high speed intermittently with slightly extended high speed interval to compensate the performance loss caused by transition overhead. Before m reaches certain value, e.g. m_{opt}, the maximal temperature will monotonically decrease until a minimized peak temperature is achieved at m_{opt}, beyond which the temperature reduction contributed by high-low speed oscillating is amortized by excessive extension of high speed interval, so that the maximal temperature will start to escalate.

Now the problem becomes how to identify the optimal value of m that leads to the lowest peak temperature. Consider the example illustrated in Figure 4.3 (C). Based on [90], when the temperature reaches the stable status, the maximal temperature of a non-ideal-case *M-Oscillating* schedule, i.e. $T^{NI}_{max}(\tilde{S}(m,t))$, can be expressed as

$$T^{NI}_{max}(\tilde{S}(m,t)) = T^{\infty}_{y'} = T_{y'} + \frac{T_{y'}}{1-K_{y'}}K_{y'}, \tag{4.4}$$

where $T_{y'}$ can be expressed as a function of m based on equation (3.25) and can be calculated iteratively once the temperature information of previous speed transition points, i.e. T_a, $T_{x'}$ and T_b, are available. Specifically, we have

$$\begin{aligned} T_{y'} &= G_2 + (T_b - G_2)e^{-B_2 t_2^m} \\ T_b &= G_0 + (T_{x'} - G_0)e^{-B_0 \tau} \\ T_{x'} &= G_1 + (T_a - G_1)e^{-B_1 t_1^m} \\ T_a &= G_0 + (1 - e^{-B_0 \tau}) \end{aligned} \tag{4.5}$$

where t_1^m and t_2^m are obtained from equation (4.2) and (4.3), respectively. $K_{y'}$ can be

computed by

$$K_{y'} = e^{-(B_1 t_1^m + B_2 t_2^m + 2B_0\delta)}, \tag{4.6}$$

where δ is defined in equation (4.1).

4.3 Searching Algorithm for the Optimal m

Once the formula of maximal temperature is available, one straightforward way to find the optimum m is by setting the first-order derivative of equation (4.4) equal to zero and solving for m. However, it can be a challenging problem not only for the complexity of the equation, but also because the discrete nature of equation (4.4) (the solution of the optimum m is not necessarily an integer). Instead, we adopt a searching algorithm [86] that can locate optimum m in linear time based on equation (4.4).

A systematic way to reduce the size of the *interval of uncertainty*, i.e. the interval that include the optimum point, is to randomly choose two point m_1 and m_2 ($m_1 < m_2$) between the initial *interval of uncertainty*, i.e. 1 to m_{Max}, and evaluate the value of $T_{max}^{NI}(\tilde{S}(m_1, t))$ and $T_{max}^{NI}(\tilde{S}(m_2, t))$, respectively. Then, three possible scenarios could happen as illustrated in Figure 4.4. If $T_{max}^{NI}(\tilde{S}(m_1, t)) < T_{max}^{NI}(\tilde{S}(m_2, t))$ (Figure 4.4 (A)), the minimum of the function could lie between 1 and m_1 or m_1 and m_2. However, the m_{opt} cannot occur between the interval m_2 and m_{Max}. Hence, the interval between m_2 and m_{Max} can be eliminated so that we only need to choose two points between interval 1 and m_2 for the next round of evaluation. Similarly, the interval between 1 and m_1 can be removed if the scenario depicted in Figure 4.4 (B) happens. And if the scenario in Figure 4.4 (C) occurs, both intervals from 1 to m_1 and the one from m_2 to m_{Max} can be eliminated from the current *interval of uncertainty*.

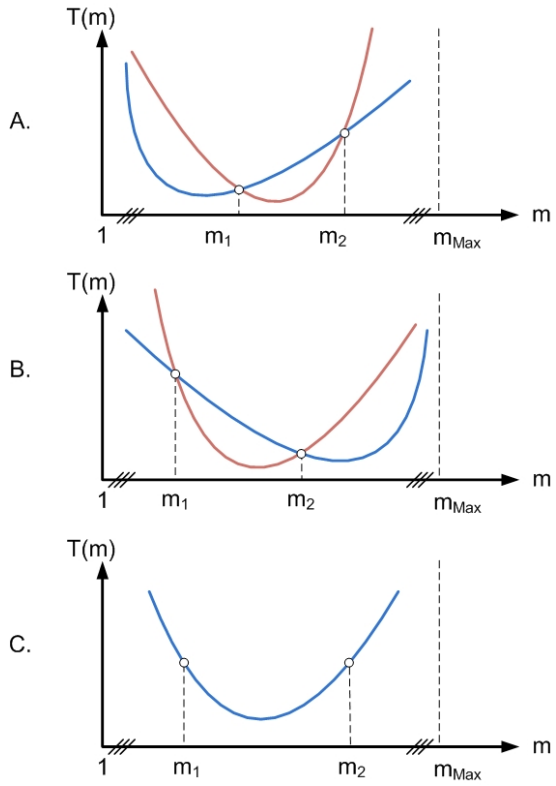


Figure 4.4: Searching for optimum m

Therefore, to obtain the optimum m, we can iteratively eliminate the *interval of uncertainty* of m_{opt}. The procedure will be stopped until the bound is narrow enough to locate a single value of m.

The algorithm is summarized in Algorithm 1. Initially, the *interval of uncertainty* is set between 1 and the upper bound of m. We choose two points, i.e. Lb and Ub (in step 4 and 5, respectively), in a way that they can equally divide the initial interval into three segments such that after each iteration the *interval of uncertainty* can be reduced by at least one third. Thus, new *interval of uncertainty* boundaries, i.e. L and R, are updated and new evaluation points are calculated during each iteration. The loop will stop until the *interval of uncertainty* is small enough to locate single integer value which is the best m we choose.

Algorithm 1 Searching for optimum m

Input: the upper bound of m: m_{Max};
Initialize uncertainty interval: $L = 1$, $R = m_{Max}$;
while (1) **do**
  $Lb = \lfloor L + 1/3 \times (R - L) \rfloor$;
  $Ub = \lfloor L + 2/3 \times (R - L) \rfloor$;
  if $T^{NI}_{max}(\tilde{S}(Lb, t)) < T^{NI}_{max}(\tilde{S}(Ub, t))$; **then**
    $L = L$;
    $R = Ub$;
  else if $T^{NI}_{max}(\tilde{S}(Lb, t)) < T^{NI}_{max}(\tilde{S}(Ub, t))$; **then**
    $L = Lb$;
    $R = R$;
  else if $T^{NI}_{max}(\tilde{S}(Lb, t)) == T^{NI}_{max}(\tilde{S}(Ub, t))$; **then**
    $L = L$;
    $R = Ub$;
  end if
  if $R - L \leq 2$ **then**
    Break;
  end if
end while

4.4 Empirical Studies

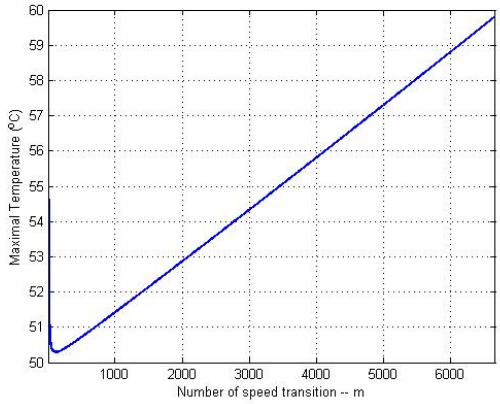
In this section, we use experiments to test the effectiveness of the proposed non-ideal-case *M-Oscillating* technique. At the same time, we also want to validate our assumption that the maximal temperature is a unimodal function of m. We generate an ideal two speed schedule, like the one shown in Figure 4.1. We set $t_1 = t_2 = 1000$ seconds, and the two speeds S_1 and S_2 are 0.8 and 1.0, respectively (normalized). Here, we conducted three tests by assuming the transition clock halting time τ equals to 0.1, 1 and 10 seconds, in order to see the thermal characteristics of the non-ideal-case *M-Oscillating* schedules when transition overheads are set to different magnitudes compared with the total duration of the schedules.

By using the method introduced in Chapter 3, we can derive the corresponding non-ideal-case *M-Oscillating* schedule and find its maximal temperature as we increase m from 1 to m_{Max}. The results are plotted in Figure 4.5, which clearly show

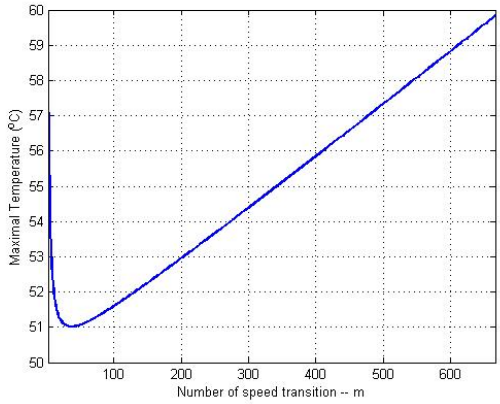
that there indeed exists an optimum m at which the lowest maximal temperature can be achieved. Generally speaking, when transition overhead is small, a lower maximal temperature can be achieved because less transition overhead potentially increases the upper bound of m which in turn lets us fully exploit the advantage of frequent speed transition, whereas larger transition overhead actually decreases the number of possible transition and has to significantly extend the high speed duration in order to compensate large performance loss during transitions.

4.5 Summary

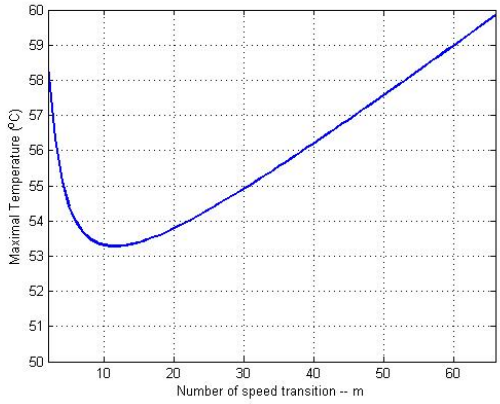
With the continuous scaling of semiconductor technology, the interdependency of temperature and leakage exacerbates not only the power/energy minimization problem but also the thermal management problem. In this chapter, we incorporate the leakage/temperature dependency and non-negligible transition overhead into the real-time scheduling analysis that aims at minimizing the peak temperature. We present a novel scheduling technique that can effectively reduce the peak temperature when executing a hard realtime periodic task set. A fast searching algorithm is adopted to efficiently find the solution.



(a) $\tau = 0.01s$



(b) $\tau = 0.1s$



(c) $\tau = 1s$

Figure 4.5: Maximal temperature of the non-ideal-case *M-Oscillating* schedules with different transition overhead and divisions (m)

CHAPTER 5

SCHEDULING FOR ENERGY MINIMIZATION

In previous chapter, we present a novel scheduling technique to reduce the peak temperature of a processor when executing a periodic task set under the thermal steady state. In this chapter, we investigate how to apply real-time scheduling techniques to solve the problem on how to minimize the overall energy consumption of a hard real-time system. The complexity of the problem lies in the fact that the leakage energy consumption depends on both supply voltage level and the temperature. For instance, two identical speed schedules may consume dramatically different energy simply because their initial temperatures are different. Therefore, how to accurately and efficiently estimate the energy consumption of various candidate speed schedules is the key to solve the energy optimization scheduling problem. To this end, we develop an effective closed-form energy estimation method with the leakage/temperature dependency taken into consideration.

Based on the proposed energy estimation method, we further develop two scheduling algorithms to minimize the overall energy consumption. The first algorithm is an off-line algorithm, targeting at real-time systems consisting of a set of periodic tasks with the same periods and deadlines. The second algorithm is an on-line algorithm, which is intended for more general real-time systems consisting of multiple sporadic tasks.

The rest of this chapter is organized as follows. Section 5.1 discusses a number of related works. Section 5.2 provides several motivational examples. In Section 5.3, we show how to derive the proposed energy estimation equation, based on which our overall energy minimization scheduling algorithms are presented in Section 5.4. Experimental results are discussed in Section 5.5. Section 5.6 concludes this chapter.

5.1 Related Work

Recently, there are a number of articles ([16, 15, 50, 110]) published on minimizing the overall energy consumption with the leakage/temperature dependency taken into consideration. Specifically, Yang et al. [110] develop a quadratic leakage model to simplify the leakage/temperature dependency. Based on this model, they propose the *Pattern-based* approach to schedule a periodic task and reduce the energy consumption when a processor reaches the thermal steady state. This approach mainly focuses on a processor without *DVS* capability. It reduces the energy consumption by periodically switching a processor between the active and dormant modes. Bao et al. [16] employ a piece-wise linear (PWL) leakage model to capture the interdependency between the leakage, temperature and supply voltage. They propose a method to distribute idle intervals judiciously when schedule a task graph such that the temperature of a processor can be effectively "cooled down" to reduce the leakage energy consumption.

While a schedule developed off-line may be more effective to provide certain guarantee, an on-line schedule can be more effective in saving energy. Moreover, for a more complicated real-time system, such as a real-time system consisting of multiple sporadic tasks scheduled by the *EDF* policy, the temperature rarely reaches the thermal steady state. In such a scenario, an on-line scheduling algorithm becomes more desirable.

Several works are published to address the on-line energy optimization problems. For example, Yuan et al. [116] introduce a simple heuristic to turn on or off a processor based on its workload and temperature. This approach directly employs the circuit-level leakage model ([65, 8]) to capture the interdependency between the leakage and the temperature. However, as a result of the complexity from the nonlinear and high order terms of the leakage model, this approach can only be applied for

soft real-time systems. Bao et al. [15] propose an on-line temperature-aware *DVS* technique to minimize the energy consumption when scheduling a task graph. This approach requires extensive off-line static analysis to generate a *lookup table* (LUT) for each task under various starting temperature, so that the scheduler can adjust the processor speed accordingly to minimize energy consumption on-line. There are also other on-line thermal-aware approaches such as [60, 115, 52, 119, 27]. They focus more on peak temperature reduction or throughput maximization under a given peak temperature constraint.

In this chapter, we are interested in the thermal-aware overall energy minimization scheduling problem. By incorporating the interdependency between leakage and temperature, we derive an closed-form energy estimation method to find the potential energy consumption of a candidate schedule. Then, based on the energy estimation method, two scheduling algorithms are proposed. The first one is an off-line approach which is developed to optimize the energy consumption of a single periodic task under the thermal steady state. The second method is an on-line energy minimization scheduling algorithm for a hard real-time system running multiple sporadic tasks. To the best of our knowledge, we are not aware of any other thermal-aware scheduling techniques that can guarantee the hard real-time deadlines for an aperiodic task set.

Based on the system models presented in Chapter 3, the formal problem definitions of this chapter can be given as follows:

Problem 5.1.1. *Given a hard real-time task set with period p and worst-case execution time c, develop a feasible schedule off-line such that the overall energy consumption is minimized when a processor reaches the thermal steady state while at the same time ensuring the peak temperature constraint T_{max}.*

Problem 5.1.2. *Given a hard real-time task set Γ consisting of N sporadic tasks $\{\tau_1, \tau_2, ..., \tau_n\}$, develop an on-line speed schedule under the* EDF *policy such that the*

overall energy consumption is minimized.

5.2 Motivational Examples

In this section, we use an example to illustrate the concepts of different energy calculation methods. We show why an energy estimation method with low computational complexity and good accuracy is highly desirable in scheduling algorithm development.

5.2.1 *M-Oscillating* for Overall Energy Reduction

First, we use a simple example to demonstrate the concept of the proposed algorithm as well as its advantages. Let us consider a sporadic task set [1] consisting of two tasks, i.e. $\tau_1 \equiv \{0, 5, 13, 13\}$ and $\tau_2 \equiv \{2, 3, 6, 6\}$, which are scheduled according to the *EDF* policy. A simple naive approach, as shown in Figure 5.1(a), is to execute a task with the maximal speed and turn to sleep mode when finish execution. Another approach, i.e. the Online DVS (*OLDVS*) approach [64], as illustrated in Figure 5.1(b), is to use two neighboring speeds of the *constant speed* (dotted lines), i.e. the speed that can finish the workload exactly at the deadline. The third approach, i.e. the *Pattern-based* approach [110], as illustrated in Figure 5.1(c), uses a single speed to run the task. This approach lets the processor's active state to be scattered into a number of intervals so that the processor can have a chance to cool down. The forth approach, *M-Oscillating*, as illustrated in Figure 5.1(d), chooses the same speed levels as the *OLDVS* approach. It then divides the high speed and the low speed interval each into m segments and oscillates between the two speeds to execute a task.

Table 5.1 shows the normalized energy consumption (w.r.t. the overall energy

[1] A sporadic task τ_i is specified by $\{a_i, c_i, d_i, p_i\}$, where a_i represents the arrival time of τ_i, c_i is the *worst-case execution time* of τ_i under the maximum clock frequency, d_i is the relative deadline of τ_i with respect to its arriving time and p_i is the *minimum inter-arrival time* between any two consecutive job of τ_i. In this chapter, we refer p_i as the period of τ_i.

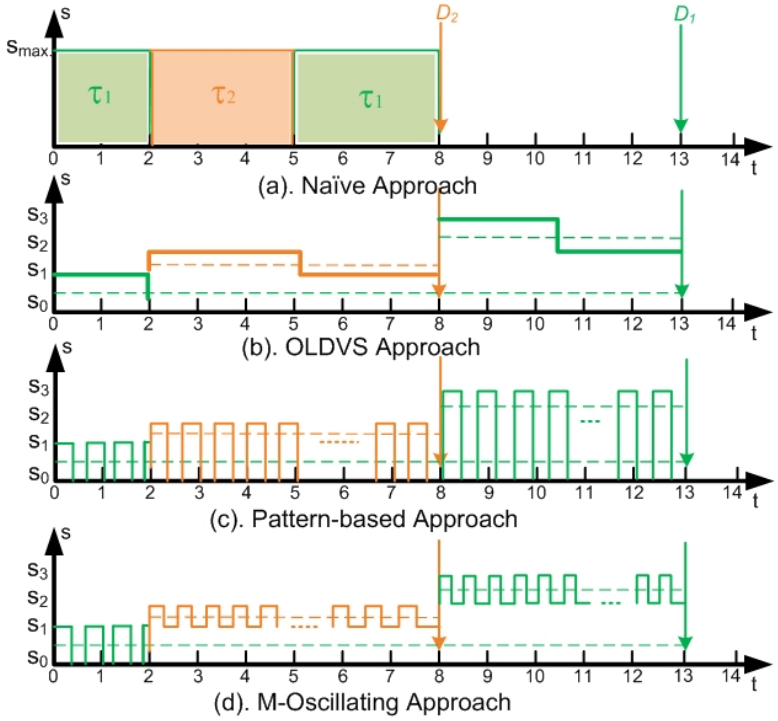


Figure 5.1: Different speed scheduling approaches under EDF policy

Table 5.1: Normalized energy consumptions of four scheduling approaches

Approach	E_{dyn}	↘	E_{leak}	↘	E_{total}	↘
Pattern	0.31	-	0.69	-	1	-
OLDVS	0.28	9.6%	0.66	4%	0.94	6%
M-Osc.	0.28	9.6%	0.55	20%	0.83	17%
	E_{dyn}	↗	E_{leak}	↗	E_{total}	↗
Naive	1.38	440%	3.56	510%	4.94	494%

of the *Pattern-based* approach) based on a processor model built based on the 65nm technology (more details can be found in Section 6.5). We can see that in this example, the naive approach uses only the maximal speed level for task execution (no DVS), and as a result, the energy consumption is several times higher than the other three approaches. As shown in Table 5.1, compared with the *Pattern-based* approach, the *OLDVS* saves 6% of the overall energy consumption. The energy saving comes from two aspects. First, using two neighboring speeds instead of the single speed reduces the dynamic energy consumption. Second, the reduced dynamic power consumption directly translates into a lower temperature which also helps to save the leakage energy. Also, from Table 5.1 we can see that although both the *OLDVS* approach and the *M-Oscillating* approach consume the same amount of dynamic energy, the leakage energy consumed by different policies are quite different. Additional 11% energy can be saved by *M-Oscillating* approach as a result of the optimized temperature achieved by speed oscillating.

5.2.2 Fast and Accurate Energy Calculation is Needed

As we can see from the previous example, *M-Oscillating* is shown to be very effective to reduce the overall energy consumption. Then, the problem for us is how to judiciously choose the appropriate number of transitions, with timing and energy transition overheads in mind, to achieve the best energy efficiency. However, as leakage varies with temperature, one key challenge in solving this problem is how to calculate the overall energy consumption for a speed schedule accurately and efficiently.

Let us consider a speed schedule shown in Figure 5.2(a). The speed schedule, $S \equiv [A, B, C, D]$, consists of four intervals with each characterized by a speed level (s_1 to s_4) and a duration ($(t_1 - t_0)...(t_4 - t_3)$).

Since the leakage energy used to take only a small portion of the overall energy

consumption, some early works ([64, 112, 30, 57, 89]) simply ignore the leakage or assume it to be constant. However, as we discussed earlier, these models can result in significant estimation errors when leakage/temperature dependency is becoming increasingly prominent.

An alternative method, similar to that in [73], is to divide a interval into small sub-intervals (as shown in Figure 5.2(b)). Within each sub-interval, one can assume the temperature (and thus the leakage) remains constant. Based on this concept, to calculate the energy consumption of S, we can divide each interval into multiple sub-intervals with equal length, i.e. ρ. Then, the energy consumption of the *first* sub-interval, i.e. from t_0 to $t_0 + \rho$, can be calculated in the following steps,

$$\begin{aligned} E_{dyn} &= C_2 \cdot {S_3}^3 \cdot \rho, \\ E_{leak} &= s_3 \cdot \rho \cdot I_s \cdot (\mathcal{A} \cdot T_0^2 \cdot e^{((\alpha \cdot s_3 + \beta)/T_0)} + \mathcal{B} \cdot e^{(\gamma \cdot s_3 + \delta)}), \\ E_{total} &= E_{dyn} + E_{leak}. \end{aligned} \tag{5.1}$$

The dynamic energy component, which is independent to the temperature, can be calculated based on our dynamic power model introduced in Chapter 3, while the leakage energy consumption is obtained by using circuit-level model (equation (2.2)). The temperature within this sub-interval is assumed to be constant, which is the initial temperature T_0. By adding the dynamic and leakage energy, we can obtain the overall energy consumption of the *first* sub-interval. Then, by applying equation (3.25), the ending temperature of the *first* sub-interval, i.e. the temperature at $(t_0 + \rho)$, can be updated and then used to calculate the leakage energy consumption of the next sub-interval. Note that in this example, $\frac{(t_4 - t_1)}{\rho}$ rounds of calculation are required to get the overall energy consumption of the speed schedule S. With equation (2.2), this

method can achieve relatively accurate energy estimation as long as the sub-interval (ρ) is sufficiently small. However, the computational cost can be very sensitive to the accuracy of this approach.

Assume that we want to compare the energy consumption of the schedule S with another schedule $S' \equiv [D, C, B, A]$, and find the better one in terms of energy reduction. Even though S and S' have identical dynamic energy consumption, the leakage energy consumptions are totally different simply because the temperature curve produced by running S and S' are dramatically different. Note that, the overall energy consumptions are different even for the same schedule starting with different initial temperatures. Therefore, we have to again, go through $\frac{(t_4-t_1)}{\rho}$ steps to find the corresponding energy consumption of S'. Using this method can be very time consuming if the interval of the schedule being evaluated is relatively long. It is computationally expensive even for an off-line approach, not to mention incorporating it into the development of on-line scheduling algorithms. Evidently, an accurate energy estimation method with low computation cost is highly desirable.

By observing equation (3.2), we notice that the relationship between the overall power consumption and the corresponding system temperature can be described by a differential equation. Intuitively, if the temperature information of a given schedule interval is obtained, the time varying power consumption and thus (by calculating the integral) the overall energy consumption can be calculated analytically. Based on this concept, the energy consumption of a given interval can be obtained conveniently in one step if the initial temperature is provided.

In the sections that follow, we first introduce our method to calculate the energy consumption for a given schedule. We then present an off-line and an on-line scheduling method to minimize the energy consumption.

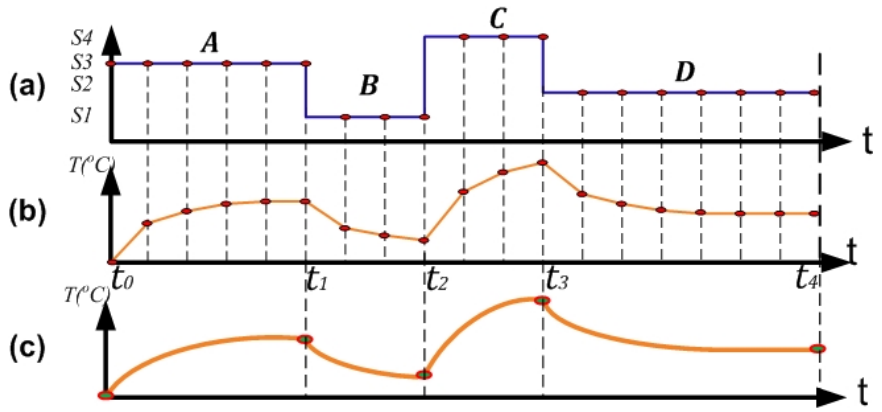


Figure 5.2: A speed schedule and its temperature curve

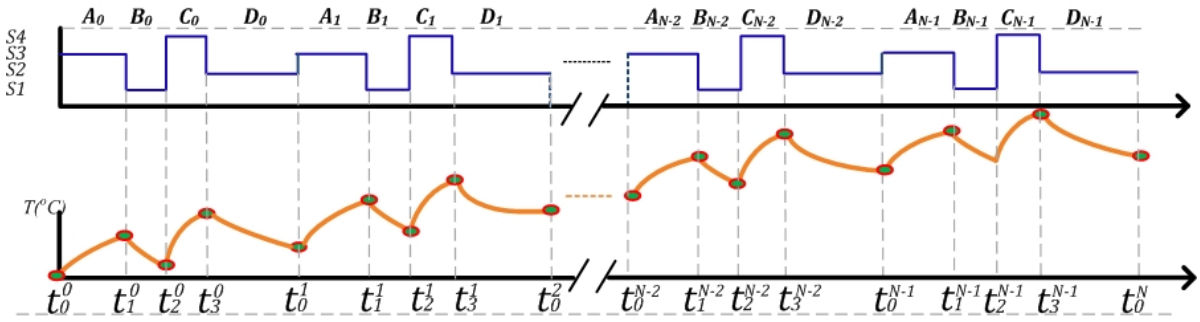


Figure 5.3: A periodic schedule and its temperature curve

5.3 Energy Estimation Equation

From the motivational example above, we can see that it is critical to develop a method that can rapidly and accurately estimate the energy consumption of a candidate schedule. As one of the major components of overall energy consumption, the leakage energy, as well as its strong dependency with temperature, have made the energy estimation a non-trivial task. In this section, we first discuss how to calculate the energy consumption of a single speed interval. Then, we discuss how to calculate the energy consumption of a periodic schedule.

5.3.1 Energy Calculation for Single Speed Interval

Without losing of generality, let us consider running a processor in mode k, i.e. applying the kth speed level, during the interval $[t_0, t_e]$. Based on our system model, the temperature change is described by equation (3.2). By integrating both sides of equation (3.2), we have

$$\begin{aligned} \int_{t_0}^{t_e} \frac{dT(t)}{dt} \cdot dt &= a \cdot \int_{t_0}^{t_e} P(t)dt - b \cdot \int_{t_0}^{t_e} T(t)dt, \\ T(t_e) - T(t_0) &= a \cdot E(t_0, t_e) - b \cdot \int_{t_0}^{t_e} T(t)dt, \\ E(t_0, t_e) &= 1/a \cdot (T(t_e) - T(t_0) + b \cdot \int_{t_0}^{t_e} T(t)dt), \end{aligned} \tag{5.2}$$

where $T(t_0)$ and $T(t_e)$ denote the starting and ending temperature of interval $[t_0, t_e]$ and $E(t_0, t_e)$ represents the energy consumed during this interval. Note that in equation (5.2), $T(t_e)$ can be analytically calculated from equation (3.25) that

$$T(t_e) = G(k) + (T(t_0) - G(k))e^{-B(k)(t_e - t_0)}. \tag{5.3}$$

Also, the integral term $\int_{t_0}^{t_e} T(t)dt$ can be expressed as,

$$\begin{aligned} &\int_{t_0}^{t_e} T(t)dt = \int_{t_0}^{t_e} G(k)dt + \int_{t_0}^{t_e} (T_0 - G(k)) \cdot e^{-B(k)(t - t_0)}dt \\ &= G(k) \cdot (t_e - t_0) + 1/B(k) \cdot (G(k) - T_0)(e^{-B(k)(t_e - t_0)} - 1). \end{aligned} \tag{5.4}$$

Replacing $T(t_e)$ and the integral term $\int_{t_0}^{t_e} T(t)dt$ in equation (5.2) with equation (5.3) and (5.4), the overall energy consumed within the interval $[t_0, t_e]$ can be

formulated by a closed-form equation

$$\begin{aligned} E(t_0, t_e) &= 1/a \cdot (G(k) + (T(t_0) - G(k))e^{-B(k)(t_e - t_0)} \\ &- T_0 + b \cdot (G(k) \cdot (t_e - t_0) \\ &+ 1/B(k) \cdot (G(k) - T(t_0))(e^{-B(k)(t_e - t_0)} - 1))). \end{aligned} \tag{5.5}$$

From equation (5.5), we can see that in order to obtain the energy consumption of a given speed interval, we only need to know the duration of the interval $(t_e - t_0)$, the initial temperature $(T(t_0))$ and the current processor speed level (k), which determines the value of $G(k)$ and $B(k)$, according to our system models.

With equation (5.5), we can quickly calculate the energy consumption of a given speed schedule. Recall the example shown in Figure 5.2. Now, by applying equation (5.5), the overall energy consumption of the schedule S can be formulated as

$$E_{total} = E(t_0, t_1) + E(t_1, t_2) + E(t_2, t_3) + E(t_3, t_4). \tag{5.6}$$

It is not difficult to see that by applying equation (5.5), the proposed energy calculation method (Figure 5.2(c)/equation (5.6)) has significantly lower computational cost than the energy estimation method in [73] (Figure 5.2(b)/equation (5.1)). The number of steps required to calculate the energy consumption of a given schedule is equal to the number of intervals a schedule has, i.e. 4, in this example. The computational overhead reduction is achieved from two aspects. First, by using the proposed closed-form energy equation, there is no need to divide one speed interval into several small sub-intervals. Second, for each interval being evaluated, instead of calculating the dynamic and leakage energy separately, we can obtain the overall energy consumption in one step.

5.3.2 Energy Calculation for Periodic Schedule

Let us consider a periodic schedule (as shown in Figure 5.3) and we want to calculate the energy consumption when running this schedule for N periods. Unless reaching the thermal steady state, the temperature curves of different periods are different. As a result, the (leakage) energy consumption within each period is also different. We can certainly employ the closed-form equation introduced above to calculate the total energy consumption, which is far more efficient than using the approach proposed in [73]. However, even if the closed-form energy equation is applied, we still have to find the starting/ending temperature at each of the 4N intervals and calculate the corresponding energy consumption within each interval individually. The computational cost can still be very high if N is large. In what follows, we introduce another energy estimation method that works more efficiently to deal with a periodic schedule.

Consider the example shown in Figure 5.3 that a periodic schedule $\tilde{S} \equiv [A, B, C, D]$ is running with an arbitrary starting temperature. Without losing of generality, we assume that the schedule $\tilde{S}$ repeats for a total number of N periods (or copies). We use $T(t_\beta^\alpha)$ to denote the starting temperature of the βth interval in the αth copy of the periodic schedule $\tilde{S}$. Also, in this example, we assume that for the interval A, B, C and D, the processor runs in the speed level S_3, S_1, S_4 and S_2, respectively and we have $S_4 > S_3 > S_2 > S_1$.

To obtain the overall energy consumption of $\tilde{S}$, we apply equation (5.2) to each

speed segment. Then, we have

$$\begin{aligned}
E(t_0^0, t_1^0) &= \frac{1}{a}(T(t_1^0) - T(t_0^0) + b\int_{t_0^0}^{t_1^0} T(t)dt), \\
E(t_1^0, t_2^0) &= \frac{1}{a}(T(t_2^0) - T(t_1^0) + b\int_{t_1^0}^{t_2^0} T(t)dt), \\
E(t_2^0, t_3^0) &= \frac{1}{a}(T(t_3^0) - T(t_2^0) + b\int_{t_2^0}^{t_3^0} T(t)dt), \\
E(t_3^0, t_0^1) &= \frac{1}{a}(T(t_0^1) - T(t_3^0) + b\int_{t_3^0}^{t_0^1} T(t)dt), \\
&\cdots \\
E(t_2^{N-1}, t_3^{N-1}) &= \frac{1}{a}(T(t_3^{N-1}) - T(t_2^{N-1}) + b\int_{t_2^{N-1}}^{t_3^{N-1}} T(t)dt), \\
E(t_3^{N-1}, t_0^N) &= \frac{1}{a}(T(t_0^N) - T(t_3^{N-1}) + b\int_{t_3^{N-1}}^{t_0^N} T(t)dt).
\end{aligned} \tag{5.7}$$

By summing up the above $4N$ equations we get

$$\begin{aligned}
E_{sum} &= 1/a \cdot (T(t_0^N) - T(t_0^1) + b \cdot (\sum_{x=0}^{N-1}\int_{t_0^x}^{t_1^x} T(t)dt \\
&+ \sum_{x=0}^{N-1}\int_{t_1^x}^{t_2^x} T(t)dt + \sum_{x=0}^{N-1}\int_{t_2^x}^{t_3^x} T(t)dt + \sum_{x=0}^{N-1}\int_{t_3^x}^{t_0^{x+1}} T(t)dt)),
\end{aligned} \tag{5.8}$$

where E_{sum} is the total energy consumption that we want to find. $T(t_0^N)$ and $T(t_0^0)$ are the final and starting temperature of the periodic schedule $\tilde{S}$, respectively. Each of the summation term is corresponding to the summation of all N integral terms for interval A, B, C and D.

Recall that in equation (5.4), if the kth speed level is applied during an interval, the corresponding integral term can be analytically expressed. $G(k)$ and $B(k)$ are

known *a priori* when a processor model is given. $T(t_0)$ is the initial temperature of the segment being evaluated. Similarly, from equation (5.8), it is not difficult to see that the overall energy consumption E_{sum} depends upon the starting temperature of each speed interval, i.e. $T(t_0^0)$, $T(t_1^0)$, $T(t_2^0)$, $T(t_3^0)$, ... ,$T(t_2^{N-1})$ and $T(t_3^{N-1})$.

To obtain these values, one straightforward way is to keep track of the entire schedule $\tilde{S}$ and calculate the temperature at those specific time instants. However, the complexity of this method can be extremely high if there are too many number of intervals. A more computationally efficient method is to formulate the temperature values analytically. According to [90], the differences between the ending temperature and the starting temperature of each single copy of the schedule form a geometric series. Specifically, we have

$$\begin{aligned} T(t_0^2) - T(_0^1) &= (T(t_0^1) - T(_0^0)) \cdot K_0, \\ T(t_0^3) - T(_0^2) &= (T(t_0^2) - T(_0^1)) \cdot K_0, \\ &\cdots \\ T(t_0^N) - T(_0^{N-1}) &= (T(t_0^{N-1}) - T(_0^{N-2})) \cdot K_0, \end{aligned} \tag{5.9}$$

where $K_0 = exp(-B(3)(t_1^0 - t_0^0) - B(1)(t_2^0 - t_1^0) - B(4)(t_3^0 - t_2^0) - B(2)(t_0^1 - t_3^0))$.

The significance of this observation is that we can obtain the temperature information within the qth period of the periodic schedule $\tilde{S}$ based on those within the first copy. For example, the starting temperature of the qth period of the schedule, i.e. $T(t_0^q)$, can be effectively calculated by

$$T(t_0^q) = T(t_0^0) + \frac{(T(t_0^1) - T(t_0^0)) \cdot (1 - K_0^q)}{1 - K_0}. \tag{5.10}$$

Similarly, other temperature information within the qth period can also be formulated as[2]:

$$\begin{aligned} T(t_1^q) &= T(t_1^0) + \frac{(T(t_0^1) - T(t_0^0))(1 - K_0^q)K_1}{1 - K_0}, \\ T(t_2^q) &= T(t_2^0) + \frac{(T(t_0^1) - T(t_0^0))(1 - K_0^q)K_2}{1 - K_0}, \\ T(t_3^q) &= T(t_3^0) + \frac{(T(t_0^1) - T(t_0^0))(1 - K_0^q)K_3}{1 - K_0}, \end{aligned} \tag{5.11}$$

where

$$\begin{aligned} K_1 &= exp(-B(1)(t_1^0 - t_0^0), \\ K_2 &= exp(-B(4)(t_1^0 - t_0^0) - B(1)(t_2^0 - t_1^0), \\ K_3 &= exp(-B(2)(t_1^0 - t_0^0) - B(1)(t_2^0 - t_1^0) - B(2)(t_3^0 - t_2^0)). \end{aligned} \tag{5.12}$$

By applying equation (5.10) and (5.11), we can see that the temperature of all speed transition instants can be obtained analytically after we get the temperature information within the first period of $\tilde{S}$, i.e. $T(t_0^0), T(t_1^0), T(t_2^0)$, $T(t_3^0)$ and $T(t_0^1)$.

Now let us first consider all "A" intervals in Figure 5.3. The starting temperature of the qth "A" interval has already been formulated in equation (5.10). Therefore, the summation of the initial temperature of all "A" intervals can be expressed as

$$\sum_{q=0}^{N-1} T(t_0^q) = N \cdot T(t_0^0) + \frac{T(t_0^1) - T(t_0^0)}{1 - K_0} \cdot (N - \frac{1 - K_0^N}{1 - K_0}). \tag{5.13}$$

Once the initial temperature of all "A" intervals are available, the summation

[2]The details regarding how to derive equation (5.10) and (5.11) can be found in [90].

term of the total N corresponding integrals (in equation (5.8)) can be formulated as

$$\sum_{x=0}^{N-1} \int_{t_0^x}^{t_1^x} T(t)dt = N \cdot G(3) \cdot ((t_1^0 - t_0^0) + \beta_A) - \beta_A \cdot \sum_{q=0}^{N-1} T(t_0^q), \tag{5.14}$$

where $\beta_A = 1/B(3) \cdot exp(-B(3)(t_1^0 - t_0^0) - 1)$.

Similarly, based on the equation group (5.11), the summations of the initial temperature of all "B" intervals, "C" intervals and "D" intervals are

$$\begin{aligned}
\sum_{q=0}^{N-1} T(t_1^q) &= NT(t_1^0) + \frac{T(t_0^1) - T(t_0^0)}{1 - K_0} K_1 (N - \frac{1 - K_0^N}{1 - K_0}), \\
\sum_{q=0}^{N-1} T(t_2^q) &= NT(t_2^0) + \frac{T(t_0^1) - T(t_0^0)}{1 - K_0} K_2 (N - \frac{1 - K_0^N}{1 - K_0}), \\
\sum_{q=0}^{N-1} T(t_3^q) &= NT(t_3^0) + \frac{T(t_0^1) - T(t_0^0)}{1 - K_0} K_3 (N - \frac{1 - K_0^N}{1 - K_0}).
\end{aligned} \tag{5.15}$$

Correspondingly, the summation of the integral terms of all "B" intervals, "C" intervals and "D" intervals are

$$\begin{aligned}
\sum_{x=0}^{N-1} \int_{t_1^x}^{t_2^x} T(t)dt &= NG(1)((t_2^0 - t_1^0) + \beta_B) - \beta_B \cdot \sum_{q=0}^{N-1} T(t_1^q), \\
\sum_{x=0}^{N-1} \int_{t_2^x}^{t_3^x} T(t)dt &= NG(4)((t_3^0 - t_2^0) + \beta_C) - \beta_C \cdot \sum_{q=0}^{N-1} T(t_2^q), \\
\sum_{x=0}^{N-1} \int_{t_3^x}^{t_0^{x+1}} T(t)dt &= NG(2)((t_0^1 - t_3^0) + \beta_D) - \beta_D \cdot \sum_{q=0}^{N-1} T(t_3^q),
\end{aligned} \tag{5.16}$$

where $\beta_B = 1/B(1) \cdot exp(-B(1)(t_2^0 - t_1^0) - 1)$; $\beta_C = 1/B(4) \cdot exp(-B(4)(t_3^0 - t_2^0) - 1)$ and $\beta_D = 1/B(2) \cdot exp(-B(2)(t_0^1 - t_3^0) - 1)$.

Therefore, the overall energy consumption of the periodic schedule $\tilde{S}$ after running

for N copies can be formulated as

$$\begin{aligned}\tilde{E}_{total} &= 1/a \cdot (\frac{(T(t_0^1) - T(t_0^0)) \cdot (1 - K_0^N)}{1 - K_0} \\ &+ b \cdot (\sum_{x=0}^{N-1} \int_{t_0^x}^{t_1^x} T(t)dt + \sum_{x=0}^{N-1} \int_{t_1^x}^{t_2^x} T(t)dt \\ &+ \sum_{x=0}^{N-1} \int_{t_2^x}^{t_3^x} T(t)dt + \sum_{x=0}^{N-1} \int_{t_3^x}^{t_0^{x+1}} T(t)dt)). \end{aligned} \tag{5.17}$$

Note that, in equation (5.17), given an initial temperature, i.e. $T(t_0^0)$, we only need to calculate the temperature of those speed transition points within the first period of $\tilde{S}$ to get the entire temperature information of the schedule. As a result, the overall energy consumption can be calculated efficiently. It is worth to mention that the accuracy of employing equation (5.17) to estimate energy consumption is contingent upon the accuracy of using linear approximated leakage model. In Section 6.5, we use experiments to quantitatively evaluate the accuracy and efficiency of our proposed energy estimation methods.

In the next section, we apply the proposed energy estimation methods to solve the overall energy minimization scheduling problems.

5.4 Our Energy Minimization Scheduling Algorithms

In the previous section, we have proposed a simple yet effective energy estimation method to calculate the overall energy consumption of a single speed interval. Then, by formulating the difference between the ending and the starting temperature of each consecutive copy of a periodic schedule as a geometric series, we further propose an efficient method to estimate the the overall energy consumption of a given periodic schedule.

In this section, based on the proposed energy estimation methods and the pro-

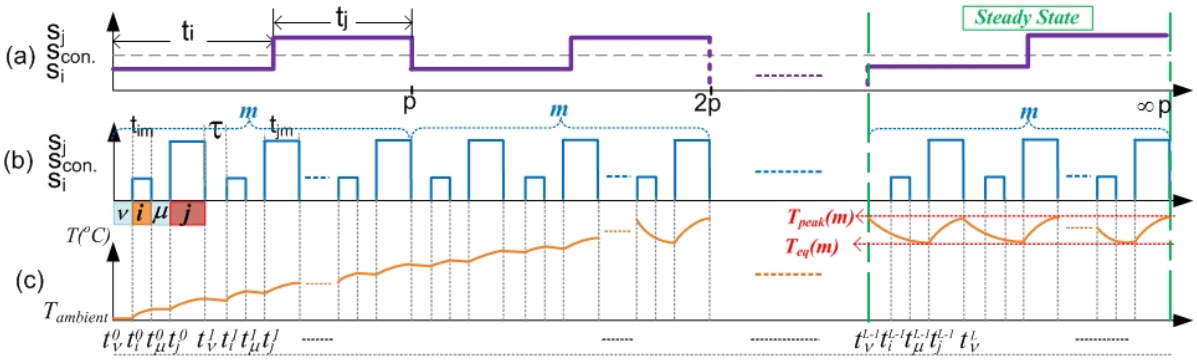


Figure 5.4: *M-Oscillating* and the corresponding temperature curve at thermal steady state

posed non-ideal-case *M-Oscillating* technique, two energy minimization scheduling algorithms are presented. The first one is targeting at reducing the overall energy consumption of a real-time system that consists of only one periodic task, while the second one can be applied for a more general case that a real-time system consisting of multiple sporadic tasks scheduled under the *EDF* policy.

5.4.1 Energy Minimization for Single Task under Thermal Steady State

In this sub-section, we propose an off-line scheduling approach to minimize the overall energy consumption of a hard real-time system under the thermal steady state. We assume that the task set consists of a set of periodic tasks with equal periods and deadlines. We can further assume such a system consists of only one periodic task with its deadline equal to its period.

Consider an *M-Oscillating* schedule and the corresponding temperature curve, which are depicted in Figure 5.4(b) and (c). Without losing of generality, we assume there are total m divisions and correspondingly, $2m$ times of mode switchings within one task period.

The processor is assumed to be turned into idle mode between each speed transition for a small time frame τ. To ease the presentation, we denote that the processor

is in mode μ during each low-high transition and in mode ν during each high-low transition even though they represent the same idle mode. Similar to the example we showed in Figure 5.3, in an *M-Oscillating* schedule, $T(t_M^\alpha)$ denotes the starting temperature of the processor mode M in the αth division.

As we start to run the processor with the *M-Oscillating* schedule from certain initial temperature, i.e. T_{amb}, in this case, the temperature will gradually increase. Eventually, when the heat generated by the processor matches the heat that can be removed by the heat sink, the processor is said to be in its thermal steady state, during which, the temperature will follow a periodic pattern instead of increasing indefinitely. Our first goal is to minimize the overall energy consumption at the thermal steady state.

From Figure 5.4(b) and (c), it is not difficult to see that under the thermal steady state, the processor reaches its peak temperature (T_{peak}) at the end of each high speed interval and then drops to a so-called equilibrium temperature, i.e. T_{eq}, after each idle period immediately following the low speed interval (μ interval). Note that, based on the previous proposed energy equation, the overall energy consumption of the *M-Oscillating* schedule under thermal steady state can be formulated once the temperature information at the mode switching instants are available. Then, for all possible value of m, we choose the one that results in the minimum overall energy consumption as our solution. One straightforward way to calculate the peak temperature for a given m is to trace the temperature change by using equation (3.25) from the T_{amb} until the temperature saturated at certain time instant and pick the highest one as the T_{peak}. However, this method can be very time consuming if the number of mode switchings, i.e. m, is large. One better solution is to treat the *M-Oscillating* schedule as a periodic schedule and formulate the temperature change analytically. Similar to equation (5.10), the temperature at time instant t_ν^L can be

formulated as

$$T(t_\nu^L) = T(t_\nu^0) + \frac{(T(t_\nu^1) - T(t_\nu^0)) \cdot (1 - K_0^L)}{1 - K_0}, \tag{5.18}$$

where $K_0 = exp(-B(j)t_{jm} - B(i)t_{im} - 2B_{idle}\tau)$ and $B_{idle} = B(\mu) = B(\nu)$. Then, the peak temperature of a given m can be obtained by setting $L \to \infty$. Since K_0 is always less than 1 [90], we have

$$\lim_{L\to\infty} T(t_\nu^L) = T(t_\nu^0) + \frac{T(t_\nu^1) - T(t_\nu^0)}{1 - K_0}. \tag{5.19}$$

Note that once T_{peak} is found, other temperature values at the mode transition points within the same segment are readily available based on equation (3.25). Therefore, the overall energy consumption in one period at the steady state can be formulated as

$$\begin{aligned} \tilde{E}(m)_{steady} &= m \cdot (E(t_\nu^{L-1}, t_i^{L-1}) + E(t_i^{L-1}, t_\mu^{L-1}) \\ &+ E(t_\mu^{L-1}, t_j^{L-1}) + E(t_j^{L-1}, t_\nu^L)) + 2m \cdot E_{sw}, \end{aligned} \tag{5.20}$$

where $E(t_i^{L-1}, t_\mu^{L-1})$ and $E(t_j^{L-1}, t_\nu^L)$ denote the energy consumption of the low and high speed interval, respectively. $E(t_\nu^{L-1}, t_i^{L-1})$ and $E(t_\mu^{L-1}, t_j^{L-1})$ correspond to the energy consumption of the idle period, and E_{sw} is the switching energy overhead and there are a total number of $2m$ transitions taking place.

Then, our problem is to minimize $\tilde{E}(m)_{steady}$ subject to $m \leq m_{Max}$. Because of the simplicity of both our energy estimation technique as well as the peak/equilibrium temperature calculation method, we can use simple exhaustive search to find the optimal value of m. The complete algorithm is shown in Algorithm 2.

Algorithm 2 Off-line M-Oscillating algorithm

1: **Input:** p, c, E_{sw}, S_i, S_j, τ,T_{max};
2: **Output:** E_{min} and m_{opt};
3: $E_{min} = \infty$ and $m_{opt} = 0$;
4: $m_{Max} = \lfloor \frac{t_i}{\delta+\tau} \rfloor$;
5: **for** m=1:1:m_{Max} **do**
6: $t_i^m = \frac{t_i}{m} - \tau - \delta$;
7: $t_j^m = \frac{t_j}{m} - \tau + \delta$;
8: Calculate $T_{peak}(m)$;
9: **if** $T_{peak}(m) \leq T_{max}$ **then**
10: Calculate $T_{eq}(m)$ and $\tilde{E}(m)$;
11: **if** $\tilde{E}(m)_{steady}(m) < E_{min}$ **then**
12: $E_{min} \leftarrow \tilde{E}(m)_{steady}(m)$
13: $m_{opt} \leftarrow m$
14: **end if**
15: **end if**
16: **end for**
17: **Return:** E_{min} and m_{opt}

5.4.2 Energy Minimization of Multiple Sporadic Tasks under the EDF Policy

In Section 5.4.1, we have introduced an effective energy minimization speed scheduling approach targeting at the thermal steady state. This approach is off-line in nature since we assume the task always takes its *WCET* and the processor reaches the thermal steady state. However, in real world applications, the actual execution time of a task can be only a small fraction of its *WCET*. A schedule developed off-line may provide certain guarantee, but an on-line schedule can be more effective in saving energy. Moreover, for a more complicated real-time system, such as a real-time system consisting of multiple sporadic tasks scheduled by the *EDF* policy, a processor rarely reaches the thermal steady state. In such a scenario, the off-line algorithms simply become not effective at all.

In this sub-section, we extend our method and develop an on-line scheduling algorithm for task sets with multiple tasks scheduled by the *EDF* policy.

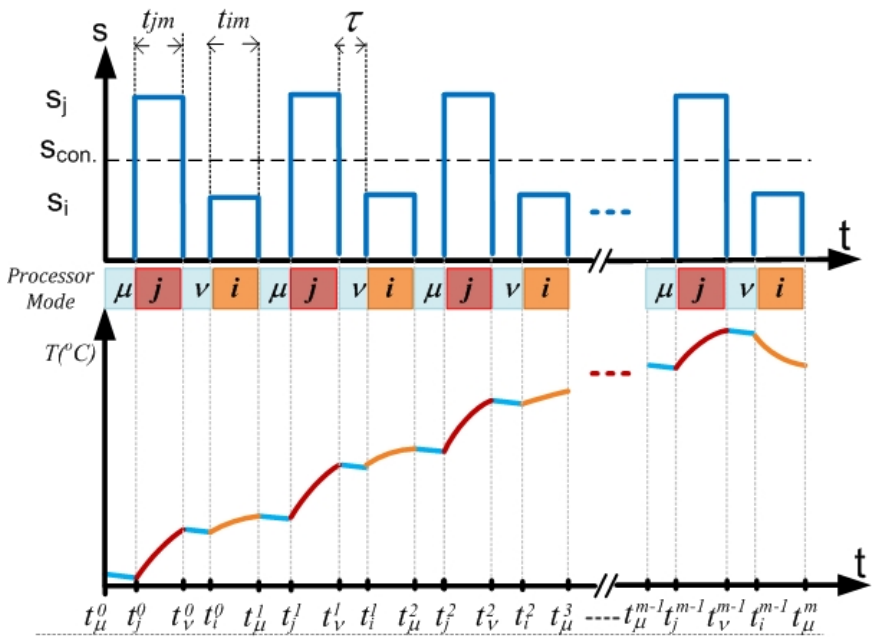


Figure 5.5: *M-Oscillating* and the corresponding temperature curve at transient stage

We still apply the proposed non-ideal-case *M-Oscillating* technique in our on-line scheduling method. One of the key challenges for the on-line scheduling algorithm development is to rapidly calculate the energy consumption of a candidate schedule during run-time. Therefore, in what follows, we first introduce how to formulate the energy consumption of an *M-Oscillating* schedule at the transient stage, i.e. before reaching the thermal steady state. Then, we discuss how to combine the *M-Oscillating* with the existing *EDF* policy to schedule multiple sporadic tasks.

Online *M-Oscillating* Energy Calculation without Steady State Temperature Information

The energy consumption of an *M-Oscillating* schedule during the transient stage can be formulated by using the same methodology as we proposed in Section 5.3.2. Consider an *M-Oscillating* schedule and the corresponding temperature curve in Figure 5.5. The *M-Oscillating* schedule can be considered as a periodic speed schedule

with one period consists of four speed intervals, namely, μ, j, ν and i. Therefore, the equations we derived in Section 5.3.2, i.e. equation (5.17), can be effectively applied to formulate the energy consumption of an *M-Oscillating* schedule during transient stage.

Assuming at certain scheduling point, there are a total number of m divisions and the current temperature is $T(t_\mu^0)$. Then, the overall energy consumption is

$$\begin{aligned}\tilde{E}(m) &= 1/a \cdot (\frac{(T(t_\mu^1) - T(t_\mu^0)) \cdot (1 - K_0^m)}{1 - K_0} \\ &+ b \cdot (\sum_{x=0}^{m-1} \int_{t_\mu^x}^{t_j^x} T(t)dt + \sum_{x=0}^{m-1} \int_{t_j^x}^{t_\nu^x} T(t)dt \\ &+ \sum_{x=0}^{m-1} \int_{t_\nu^x}^{t_i^x} T(t)dt + \sum_{x=0}^{m-1} \int_{t_i^x}^{t_\mu^{x+1}} T(t)dt)), \end{aligned} \tag{5.21}$$

where the four summation terms can be expressed as

$$\begin{aligned}\sum_{x=0}^{m-1} \int_{t_\mu^x}^{t_j^x} T(t)dt &= m \cdot G_{idle} \cdot (\tau + \beta_\mu) - \beta_\mu \cdot \sum_{q=0}^{m-1} T(t_\mu^q), \\ \sum_{x=0}^{m-1} \int_{t_j^x}^{t_\nu^x} T(t)dt &= m \cdot G(j) \cdot (t_{jm} + \beta_j) - \beta_j \cdot \sum_{q=0}^{m-1} T(t_j^q), \\ \sum_{x=0}^{m-1} \int_{t_\nu^x}^{t_i^x} T(t)dt &= m \cdot G_{idle} \cdot (\tau + \beta_\nu) - \beta_\nu \cdot \sum_{q=0}^{m-1} T(t_\nu^q), \\ \sum_{x=0}^{m-1} \int_{t_i^x}^{t_\mu^{x+1}} T(t)dt &= m \cdot G(i) \cdot (t_{im} + \beta_i) - \beta_i \cdot \sum_{q=0}^{m-1} T(t_i^q), \end{aligned}$$

where $\beta_j = 1/B(j) \cdot exp(-B(j)t_{jm} - 1)$; $\beta_i = 1/B(i) \cdot exp(-B(i)t_{im} - 1)$; $\beta_\mu = \beta_\nu = 1/B_{idle} \cdot exp(-B_{idle}\tau - 1)$ and $G_{idle} = G(\mu) = G(\nu)$.

Similarly, each of the summation terms (summation of the initial temperatures) on the right-hand-side of the above equations can be further calculated as

$$
\begin{aligned}
\sum_{q=0}^{m-1} T(t_\mu^q) &= m \cdot T(t_\mu^0) + \frac{T(t_\mu^1) - T(t_\mu^0)}{1 - K_0}(m - \frac{1 - K_0^m}{1 - K_0}), \\
\sum_{q=0}^{m-1} T(t_j^q) &= m \cdot T(t_j^0) + \frac{T(t_\mu^1) - T(t_\mu^0)}{1 - K_0} K1(m - \frac{1 - K_0^m}{1 - K_0}), \\
\sum_{q=0}^{m-1} T(t_\nu^q) &= m \cdot T(t_\nu^0) + \frac{T(t_\mu^1) - T(t_\mu^0)}{1 - K_0} K2(m - \frac{1 - K_0^m}{1 - K_0}), \\
\sum_{q=0}^{m-1} T(t_i^q) &= m \cdot T(t_i^0) + \frac{T(t_\mu^1) - T(t_\mu^0)}{1 - K_0} K3(m - \frac{1 - K_0^m}{1 - K_0}),
\end{aligned}
$$

where

$$
\begin{aligned}
K_0 &= exp(-B(j)t_{jm} - B(i)t_{im} - 2B_{idle}\tau), \\
K1 &= exp(-B_{idle}\tau), \\
K2 &= exp(-B(j)t_{jm} - B_{idle}\tau), \\
K3 &= exp(-B(j)t_{jm} - 2B_{idle}\tau).
\end{aligned}
$$

Note that, in equation (5.21), given an initial temperature, i.e. t_μ^0, we only need to calculate the temperature at the time instants within the first sub-interval, e.g. t_j^0, t_ν^0, t_i^0 and t_μ^1, to get the entire temperature information and as a result, the overall energy consumption of an *M-Oscillating* schedule with m divisions can be calculated efficiently.

Combining *M-Oscillating* with Online EDF

In this subsection, based on the proposed energy estimation method, we present our on-line energy minimization scheduling algorithm: "*On-line M-Oscillating*" (*OLMO*).

To guarantee the real-time constraint, we adopt the method in [64] to calculate the constant speed. According to [64], for the arriving task τ_i (either new arrival or resumed after preemption), at time instant t, the constant speed can be determined as follows.

We first calculate the *worst-case completion time* ($WCCT$) and *worst-case remaining time* ($WCRT$) for each arrived (but not finished) task. Three cases are considered:

- τ_i resumes after τ_k

  $WCCT_i = t + c_i$

  $WCRT_i = c_i$

  $WCRT_i = WCRT_j - s_j(t - l)$ /*l:previous context switch time*/

- τ_i preempts τ_j

  $WCCT_i = WCCT_i + WCCT_k - t_p$ /* τ_i is preempted at t_p*/

- otherwise

  $WCRT_i = c_i$

  if($d_k > d_i$ or $WCCT_i < t$), $WCCT_i = t + WCRT_i$

  else $WCCT_i = WCCT_k + c_i$

The required constant speed of τ_i can thus be calculated as

$$s_{con} = \frac{WCRT_i}{WCCT_i - t}. \tag{5.22}$$

Now the two neighboring speeds (S_i and S_j) are found based on the calculated S_{con} and the given processor model. To guarantee the same workload during the

period t_p, the low speed duration t_i and high speed duration t_j can be formulated as $t_j = t_p \times \frac{S_{con} - S_i}{S_j - S_i}$ and $t_i = t_p - t_j$.

To incorporate the non-negligible transition overhead, we use equation (4.2) and (4.2), derived in Chapter 4, to find the duration of the adjusted high speed (t_{jm}) and low speed (t_{im}) interval with respect to different m values.

At each scheduling point, given the initial temperature $T(t_0^0)$, the overall energy consumption, i.e. $\tilde{E}(m)$, can thus be formulated as a function of m (equation (5.21)). Our problem is therefore to minimize $\tilde{E}(m)$. We can use a sequential search to find the optimal value of m during the run-time. The detailed algorithm is presented in Algorithm 3.

As an on-line scheduling algorithm, the complexity of the proposed *OLMO* is a crucial factor to be considered. In our approach, the most time consuming process is the searching step for the optimum m. The complexity of the searching algorithm is $O(m_{Max})$, where m_{Max} is the theoretical upper bound of m, i.e. $m_{Max} = \lfloor \frac{t_i}{\delta + \tau} \rfloor$. The upper bound of m depends on the low speed duration of a given task, i.e. t_i, and the transition overhead τ. Based on the task and processor model used in this chapter, in most cases, the optimal m can be found within 20 iterations which would pose negligible computational overhead as evidenced later in Section 5.5.1.

In the next section, we use experiments to validate the accuracy of the proposed energy estimation methods and the performance of our scheduling algorithms.

5.5 Experimental Results

In this section, we use experiments to evaluate our methods. We first examine the accuracy and efficiency of the proposed energy estimation equation. Then, the performance of the proposed energy minimization scheduling algorithms will be evaluated and discussed.

Algorithm 3 On-Line M-Oscillating (OLMO) Algorithm

while context switch to task τ_i at time t **do**
  Calculate the constant speed S_{con} by Eq. (5.22);
  Find two nearest neighboring speeds S_i and S_j of the constant speed;
  Calculate ideal low/high speed duration t_i and t_j
  Identify upper bound of m:
  if $S_i \neq idle$ **then**
    $m_{Max} = \lfloor \frac{t_i}{\delta+\tau} \rfloor$
  else
    $m_{Max} = \lfloor \frac{t_i}{2\tau} \rfloor$
  end if
  Assign: $E_{min} = \infty$ and $m_{opt} = 0$
  for m=1:m_{Max} **do**
    if $S_i \neq idle$ **then**
      $t_{im} = \frac{t_i}{m} - \tau - \delta$;
      $t_{jm} = \frac{t_j}{m} - \tau + \delta$;
    else
      $t_{im} = \frac{t_i}{m}$
      $t_{jm} = \frac{t_j}{m}$
    end if
    Calculate $\tilde{E}(m)$; Eq. (5.17)
    if $\tilde{E}(m) < E_{min}$ **then**
      $E_{min} \leftarrow \tilde{E}(m)$
      $m_{opt} \leftarrow m$
    end if
  end for
  Return: $\{S_i, S_j, t_{im}, t_{jm}, m_{opt}\}$
end while

5.5.1 Accuracy and Efficiency of Energy Estimation Technique

In this subsection, we study the accuracy and efficiency of our energy estimation methods. We adopted the processor model which is detailed in Chapter 3. The ambient temperature is set to 25^oC. The timing penalty of speed transition τ is set to be 5ms and the transition energy overhead E_{sw} is 0.01J [110].

First, we tested the performance of our energy estimation method for a single speed interval, e.g. equation (5.5). We let the processor running in intervals with different lengths and by using different modes. We then compared the overall energy consumption and computational costs achieved by using equation (5.5) with the method proposed in [73]. To ensure the accuracy of energy calculation, we used a very small time interval, e.g. 0.01s, as the step size of the method in [73]. The energy consumption of each test case and the corresponding CPU time are summarized in Table 5.2. The results obtained by using the method in [73] are labeled as "REAL", while ours are labeled as "PROPOSED". The relative errors and speedups of our approach under different test cases are also listed in the table.

Our experimental results clearly show that the proposed energy estimation method (for single speed interval) is very accurate and computationally efficient. As can be seen in Table 5.2, the energy consumption calculated by using our proposed method closely matches the one obtained by using the approach in [73], with the maximum relative error no more than 2.7% among all test cases. The computational cost, on the other hand, is significantly reduced, i.e. up to 177X or over 57X of speedup in average for our test cases. We can also see from Table 5.2 that the longer the interval is, the more efficient our method can be. The reason is that the computational cost of our method is independent to the length of the interval, whereas for the method in [73], the CPU time is very sensitive to the interval length since it needs to divide the entire speed interval into many small sub-intervals. Moreover, Table 5.2 also indicates

that in general, the relative estimation errors of our method decrease as we increase the supply voltage levels. This is because for the same interval length, higher supply voltage level results in larger absolute value of energy consumption, which increases the denominator when we calculated the relative errors.

Table 5.2: Accuracy and efficiency of energy estimation equation for single interval

PERIOD(s)	V_{dd} (v)	REAL (J)	PROPOSED (J)	ERROR	Ave.TIME-REAL (s)	Ave.TIME-PROP(s)	Ave.SPEEDUP (X)
5	0.6	29.66	28.86	2.7%	0.028	0.0025	11X
	0.8	68.39	68.23	2.3%			
	1.0	152.75	154.74	1.3%			
	1.2	387.49	392.73	1.3%			
10	0.6	59.37	57.79	2.6%	0.038	0.0027	14X
	0.8	139.13	136.10	2.1%			
	1.0	306.45	310.56	1.2%			
	1.2	780.20	790.68	1.3%			
20	0.6	118.79	115.66	2.6%	0.064	0.0026	24X
	0.8	278.49	273.53	2.0%			
	1.0	623.74	631.80	1.3%			
	1.2	1.57×10^3	1.59×10^3	1.3%			
50	0.6	297.04	289.26	2.7%	0.17	0.0028	60X
	0.8	696.56	681.83	2.1%			
	1.0	1.56×10^3	1.58×10^3	1.2%			
	1.2	3.92×10^3	3.97×10^3	1.3%			
100	0.6	594.13	578.60	2.7%	0.48	0.0027	177X
	0.8	1.39×10^3	1.36×10^3	2.1%			
	1.0	3.16×10^3	3.12×10^3	1.3%			
	1.2	7.84×10^3	7.95×10^3	1.3%			

Table 5.3: Transient state energy calculation equation evaluation

Deadline(s)	Ave.m_{max}	A.C.T.([73])	A.C.T.(SIE)	A.C.T(PSE)	Ave.Speedup (PSE vs.SIE)	A.R.E (PSE vs.[73])
5	63	4.68s	0.034s	0.00092s	36X	4.1%
10	117	9.63s	0.093s	0.0019s	49X	3.4%
20	248	18.38s	0.30s	0.0037s	81X	2.8%
50	625	46.2s	1.81s	0.014s	129X	1.9%
100	1180	108s	6.03s	0.03s	210X	1.1%

We next used experiments to examine the performance of the proposed on-line energy estimation method, i.e. equation (5.21). We generated five groups of tasks with different deadlines (D), e.g. 5s, 10s, 20s, 50s and 100s. Each group consists of 1000 tasks with randomly generated $WCET$ varying from $0.5D$ to $0.95D$. For each task, we calculated the energy consumption of an *M-Oscillating* schedule with m increasing from 1 to m_{Max}. We assume that the initial temperature is equal to the ambient temperature. Three approaches were used to obtain the energy consumptions.

- The proposed online periodic schedule energy estimation method, i.e. equation (5.21) in Section 5.4.2. We refer this method as the *PSE* method.
- The single interval based energy calculation method. Specifically, we use equation (5.5) derived in section 5.4.1 to calculate the energy consumption of each interval and apply equation (3.25) to trace the temperature. We call it the *SIE* method.
- The energy calculation method similar to the one proposed in [73].

Based on previous discussion, for a single interval, calculating the energy by using equation (5.5) has already demonstrated significant speedup over the method in [73], while the latter one has better performance in terms of accuracy (as shown in Table 5.2). Therefore, in this experiment, to evaluate the accuracy of the *PSE* method, we compared the energy consumption calculated by the *PSE* method with the ones obtained by the method in [73]. The *SIE* method is considered as the baseline approach to show the additional speedup achieved by the *PSE* method. The average CPU time (A.C.T.) of each task group achieved by using different approaches as well as the average relative error (A.R.E) and speedup of the *PSE* method were recorded and summarized in Table 5.3

As can be seen from Table 5.3, our experimental results show the superiority of the *PSE* method. On one hand, the energy consumption calculated by using the *PSE* method well matches the one obtained by the approach in [73] with the relative error kept less than 4.1%. The CPU time, on the other hand, is significantly reduced compared with the *SIE* method, i.e. up to 210X or over 94X of speedup in average for different task groups. The speedup of the *PSE* method over the *SIE* method comes from the following facts. For each valid m, the *PSE* method can efficiently obtain the potential energy consumption based on the current temperature reading and four simple temperature calculations, i.e. the ending temperatures of the 4 intervals within the first period of an *M-Oscillating* schedule. The *SIE* method, on the other hand, needs to calculate the energy consumption of a schedule interval by interval. Therefore, $4 \times m$ times of energy calculation are required to find the overall energy consumption. As m becomes larger, the computational overhead increases significantly. Moreover, the *SIE* method needs the temperature information at every speed transition point as the input. Thus, a complete calculation of temperature information is required for the entire schedule under each m, which can be extremely time consuming. This experiment shows the fact that the accuracy of the *PSE* method is guaranteed while the computational overhead is significantly reduced, which is crucial for our on-line scheduling algorithm.

5.5.2 Energy Minimization under Thermal Steady State

To study the energy saving performance of our off-line scheduling algorithm targeting at a processor when reaching the thermal steady state, we compared our approach with two existing methods,

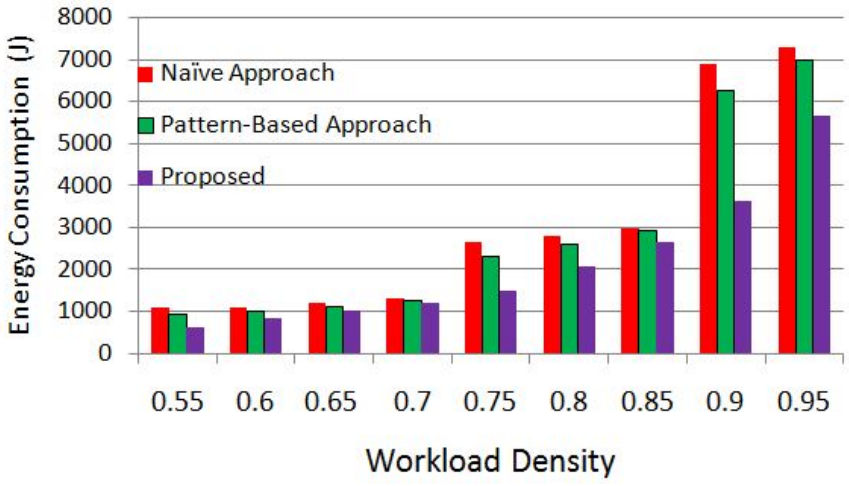
- A simple naive approach that executes the task with the maximal speed and turns to sleep mode when finish execution.

- The *Pattern-based* approach [110] that uses a single speed to run the task. This approach lets the processor's active state to be scattered into a number of intervals so that the processor can have a chance to cool down.

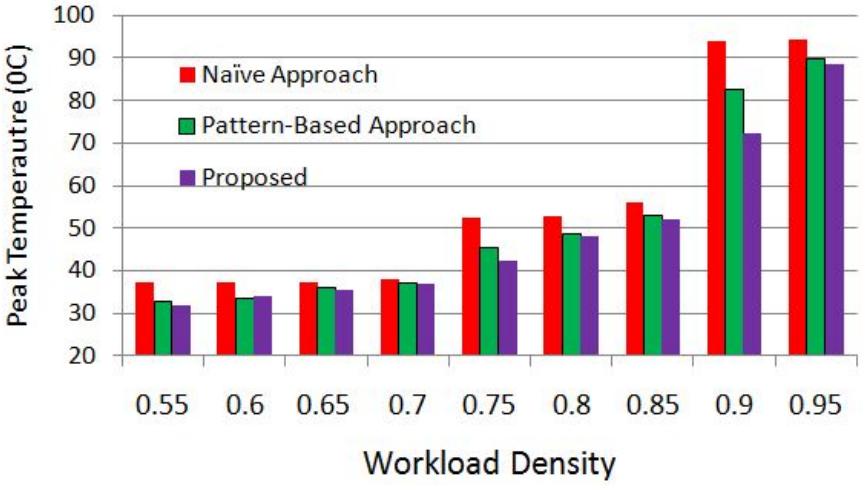
The periods of the tasks are set to 100s. The worst-case execution times are randomly generated with workload density (c/p) varying from 55% to 95% with 5% increment. The reason we do not further reduce the workload density is that when $c/p < 55\%$, the low speed mode used by the *Pattern-based* approach and the proposed technique becomes the same. The temperature constraint is set to be $T_{max} = 85^oC$. The energy consumption of each task is calculated and plotted in Figure 5.6(a).

From Figure 5.6(a), we can see that while both the *Pattern-based* approach and the proposed technique can achieve significant energy reduction (compared with naive approach), the proposed method consistently outperforms the *Pattern-based* approach under different test scenarios. Compared with the naive approach, on average, 26.5% energy can be saved by using our method versus 13.2% by applying the *Pattern-based* approach. Compared with the *Pattern-based* approach, the energy reduction of our method achieves from two aspects. (i) By using the speed oscillating, the leakage energy is reduced as the result of the reduced temperature. (ii) By applying the two neighboring speeds instead of the maximum speed combined with the idle mode, the dynamic energy consumption can be saved as well.

We also compared the peak temperature of each test case and plotted in Figure 5.6(b). The peak temperature achieved by the proposed approach can be at most 21.8^oC and 11.4^oC (or 6.4^oC and 2.1^oC in average) lower than the naive approach and the *Pattern-based* approach, respectively. That means our approach can be even more effective when the peak temperature constraint becomes tighter.



(a) Energy Vs. Workload



(b) Temperature Vs. Workload

Figure 5.6: Performance comparison of three different scheduling approaches in terms of energy minimization and peak temperature reduction

5.5.3 Online Energy Minimization under the EDF Policy

We next evaluated the performance of our proposed *OLMO* algorithm by comparing with the *Pattern-based* approach [110] and the *OLDVS* approach [64].

The task model used in this experiment is set as follows. The utilization (u_i) of each task (τ_i) is uniformly distributed between $[0.02, 1.00]$. The minimum inter-arrival

time (P_i) of τ_i also has a uniform distribution, with a range of $[50, 500]$. To simulate the actual execution time (e_i), we define the *actual-to-worst ratio* ($e2E\_ratio$), as the ratio of e_i over E_i, which is generated randomly for each job with uniform distribution between $[0, 1]$. Then, for each job of τ_i, e_i can be obtained by $e_i = e2E\_ratio \cdot E_i$. The system utilization (workload density) is set between [0.5,1.0] with 5% increment. For each test case, $1,000$ tasks are generated and bounded by the corresponding system utilization. The overall energy consumed by the three different approaches are collected and then normalized to the energy consumption of the *Pattern-based* approach.

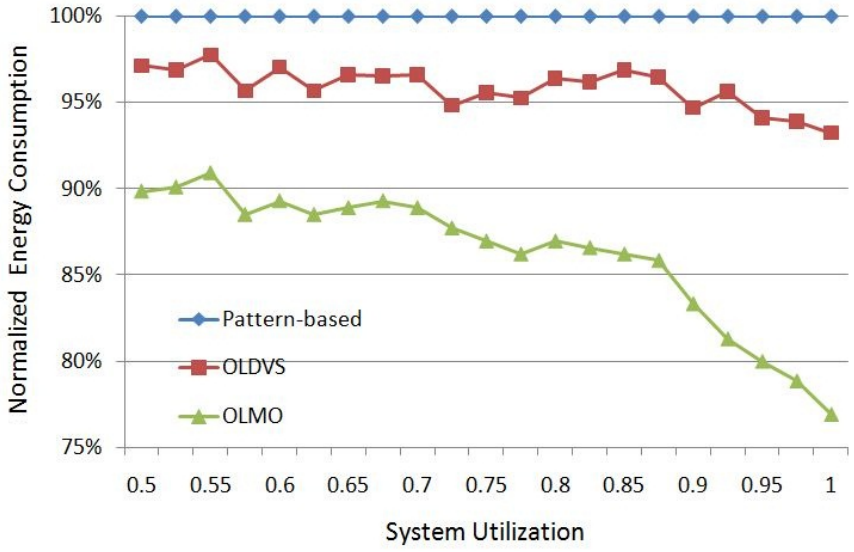


Figure 5.7: Normalized energy consumption of three scheduling approaches

The results are plotted in Figure 5.7, which clearly show that the *OLMO* significantly outperforms the other two existing approaches. Compared with the *Pattern-based* approach, the *OLMO* can save more than 10% energy on average at low system utilization range, while up to 20% energy can be saved when system utilization is high (11% and 23% energy saving when system utilization equal to 0.5 and 1.0, respectively and 14% energy saving on average). Compared with *OLDVS*, *OLMO* achieves

10% energy saving on average, and the energy saving also increases with the increment of system utilization. Compared with both the *Pattern-based* approach and the *OLDVS*, the proposed *OLMO* can significantly reduce the overall energy consumption, particularly under high system utilization range. This is because the processor tends to choose higher speed levels when the system utilization is high. As a result, the absolute values of the energy consumption as well as the temperature are high. The *M-Oscillating* can achieve significant temperature reduction, especially when the system stays in high utilization range. The minimized temperature curve directly translates into a reduced leakage energy consumption which is another contributor of our energy saving.

5.6 Summary

In this chapter, we investigate how to apply scheduling techniques to reduce the overall energy consumption of a hard real-time system. With the inter-dependency between leakage, temperature and supply voltage taken into consideration, two approaches are proposed in this chapter. The first one is an off-line approach that focuses on how to minimize the energy of a single periodic task under the thermal steady state. Next, we develop an on-line approach to reduce the overall energy consumption of a hard real-time system scheduled according to the *EDF* policy. Our experimental results demonstrate that the proposed energy estimation method can achieve significant speedup compared with existing approaches while maintaining good accuracy. In addition, the results from large number of test cases show that, both the on-line and the off-line approaches we proposed in this chapter outperform the existing works consistently.

CHAPTER 6

SCHEDULING FOR THROUGHPUT MAXIMIZATION

In Chapter 4 and 5, we investigate how to apply the *M-Oscillating* concept to solve the peak temperature reduction and the energy minimization problem, respectively. In this chapter, we are interested in studying how to maximize the throughput of a periodic real-time system under a given peak temperature constraint. We assume that different tasks, in our system, may have different power and thermal characteristics. Two scheduling approaches are presented in this chapter. The first one is built upon processors that can be either in active or sleep mode. By judiciously selecting tasks with different thermal characteristics as well as alternating between a processor's active and sleep mode, the sleep period required to cool down a processor is kept at a minimum level and as the result, the throughput is maximized. We further extend this approach for processors with dynamic voltage/frequency scaling (DVS) capability.

The rest of this chapter is organized as follows. Section 6.2 introduces the system models followed by the motivational examples in Section 6.3. Our proposed scheduling algorithms are discussed in Section 6.4. Experimental results are presented in Section 6.5. And Section 6.6 concludes this chapter.

6.1 Related Work

The desire of high-performance computing system together with the awareness of negative effects of high system temperature have driven researchers into the study of the thermal-aware throughput maximization problem. Significant efforts have been spent to solve the thermal-constrained throughput maximization problem for single processor platforms, e.g. [108, 91, 27, 118, 119], as well as multi-processor platforms, e.g. [47, 48, 71, 121, 75, 122, 104, 35].

For single processor platforms, Wang et al. [108] introduce an effective two-speed reactive scheme that runs the processor at the maximum speed until it reaches the temperature threshold, then uses an equilibrium speed to maintain the temperature. Quan et al. [91] propose a closed-form formula for feasibility analysis for a given peak temperature threshold. For multi-processor platforms, Liu et al. [71] propose a thermal-aware job allocation algorithm, which assigns hot tasks to cores close to the heat sink by taking advantage of its better heat removing capability. And cool tasks are assigned to cores far away from heat sink. By doing so, better thermal condition can be achieved such as low peak temperature and less temperature variations. In [121], Zhou et al. present a task scheduling technique based on the observation that the vertically adjacent cores have strong thermal correlation. This method then jointly considers vertically adjacent cores and forms them into super cores. Lung et al. [75] introduce a fast thermal simulation method, based on which, a task allocation algorithm is proposed by assigning a given task to a core that can result in a minimized peak temperature. In [35], Coskun et al. present a thermal-aware scheduling method called Adapt3D, which takes the thermal history into account to reduce the number of hot spot occurrences. Most of these works do not consider the leakage/temperature dependency in the analysis. They either ignore the leakage power consumption or simply treat it as constant.

In this chapter, we are interested in the problem of how to apply the scheduling technique to maximize the throughput of a real-time system under a given peak temperature constraint. There are some closely related works that have considered the leakage/temperature dependency when solving throughput maximization problems, e.g. [27, 118, 119]. Specifically, Chantem et al. [27] propose to run real-time tasks by frequently switching between the two speeds which are neighboring to a constant speed. This work, however, targets only at the scenario when the processor reaches

its thermal steady state. Zhang and Chatha [118] present a pseudo-polynomial time speed assignment algorithm (based on the dynamic programming approach), followed by a scheme to minimize the total execution latency. To guarantee the peak temperature constraint, this approach requires that the ending temperature of each period not exceeding the starting temperature, which can be very pessimistic. In addition, the two approaches mentioned above assume that all tasks have the same power characteristics, i.e. tasks consume the same amount of power as long as they run at the same processor speed, which might not be true in real-world scenario. As shown in [56], the power and thus the thermal characteristics of different real-time tasks can be significantly different. With this fact in mind, Jayaseelan and Mitra [56] introduce a method to arrange the task execution sequence to minimize the peak temperature. Zhang and Chatha [119] further develop several algorithms to maximize the throughput of a real-time system by sequencing the task execution of processors with and without dynamic voltage/frequency scaling (DVS) capability. Both works ([56] and [119]) assume that the processor can only change its speed at the boundary of task execution.

In what follows, we first introduce the system models we used in this chapter. Then, we present our scheduling algorithms for processors without and with DVS feature, respectively.

6.2 Preliminaries

In this section, we briefly introduce the system models used in this chapter, followed by the problem definition. We use the same processor model and thermal model as we introduced earlier in Chapter 3. The task and power consumption, however, are modeled in a more practical fashion, which are detailed below.

Real-Time Tasks: The task model considered in this chapter is a periodic task

set consisting of independent heterogeneous tasks. The heterogeneous nature of the tasks are manifested in a way that the power consumption of different tasks vary significantly even running under the same speed level and at the same temperature. This is because the power consumptions are strongly depending on the circuit activities [73] and the usage patterns of different functional units when executing different tasks. Specifically, we introduce a parameter μ, called *activity factor*, to capture different switching activities of different tasks. For a given task, the *activity factor* μ (ranging between $(0, 1]$) defines how intensively the functional units have been used. For common benchmark tasks, the *activity factors* can be obtained by using architectural-level power analysis tools such as *Wattch* [23]. Similarly, we also define a *leakage factor* δ, which can be used as the scaling factor of the leakage power of a processor when running different tasks.

Power Model: With the *activity factor* taken into consideration, the dynamic power consumption of a processor when executing a task τ_i at the kth speed level can be formulated as

$$P_{dyn}(i, k) = \mu_i C_2 v_k^3, \tag{6.1}$$

where v_k is the supply voltage level, C_2 is a constant and μ_i is the *activity factor* of the task τ_i.

Similarly, if we consider the *leakage factor*, the leakage power of a processor when executing the task τ_i can be effectively estimated as

$$P_{leak}(i, k) = \delta_i(C_0(k)v_k + C_1(k)Tv_k), \tag{6.2}$$

where δ_i is the *leakage factor*, $C_0(k)$ and $C_1(k)$ are constants. Therefore, the overall power consumption when executing the task τ_i at the kth speed level can be modeled

as

$$P(i,k) = \delta_i(C_0(k)v_k + C_1(k)Tv_k) + \mu_i C_2 v_k^3. \tag{6.3}$$

Accordingly, the temperature dynamic when executing the task τ_i can be formulated as

$$\frac{dT(t)}{dt} = A(i,k) - B(i,k)T(t), \tag{6.4}$$

where $A(i,k) = a(\delta_i C_0(k)v_k + \mu C_2 v_k^3)$ and $B(i,k) = b - a\delta_i C_1(k)v_k$ (we use B_s for $B(sleep)$). Hence, for a given time interval $[t_0, t_e]$, if the initial temperature is T_0, by solving equation (6.4), the ending temperature after executing the task τ_i can be formulated as:

$$\begin{aligned} T_e &= \frac{A(i,k)}{B(i,k)} + (T_0 - \frac{A(i,k)}{B(i,k)})e^{-B(i,k)(t_e - t_0)} \\ &= T_{ss}(i,k) + (T_0 - T_{ss}(i,k))e^{-B(i,k)(t_e - t_0)}, \end{aligned} \tag{6.5}$$

where $T_{ss}(i,k)$ is the steady state temperature of the task τ_i at the kth speed level. For a given task, if $T_{ss}(i,k) > T_{max}$ (the maximal temperature limit), we call it a *hot task*, or *cool task* otherwise. Apparently, for the sleep mode, we have $T_{ss} = T_{amb}$.

Based on the models introduced above, the throughput of a real-time system can be maximized when the latency of executing a task in one period is minimized. Then, our research problem can be formulated as follows.

Problem 6.2.1. *Given a tasks set* $\Gamma = \{\tau_1(t_1, \mu_1, \delta_1), \tau_2(t_2, \mu_2, \delta_2), ..., \tau_n(t_n, \mu_n, \delta_n)\}$, *where* t_i, μ_i *and* δ_i *are the execution time, activity factor and the leakage factor of the task* τ_i, *respectively. Develop a feasible schedule such that the latency of executing one iteration of* Γ *is minimized under the thermal steady state while ensuring a given peak temperature constraint* T_{max}.

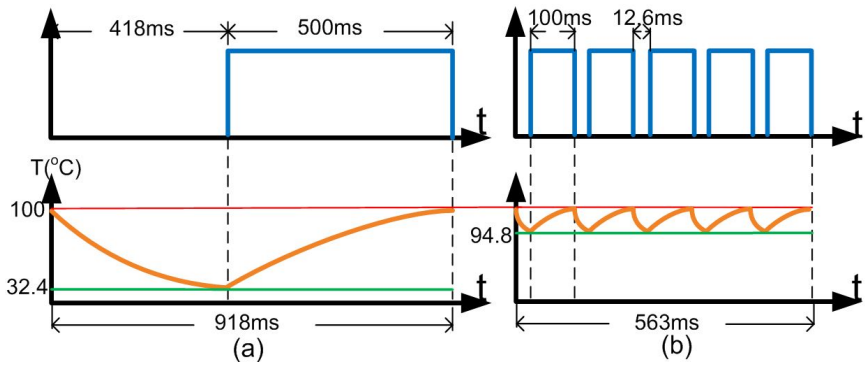


Figure 6.1: Improve the throughput by distribute sleep period between task executions

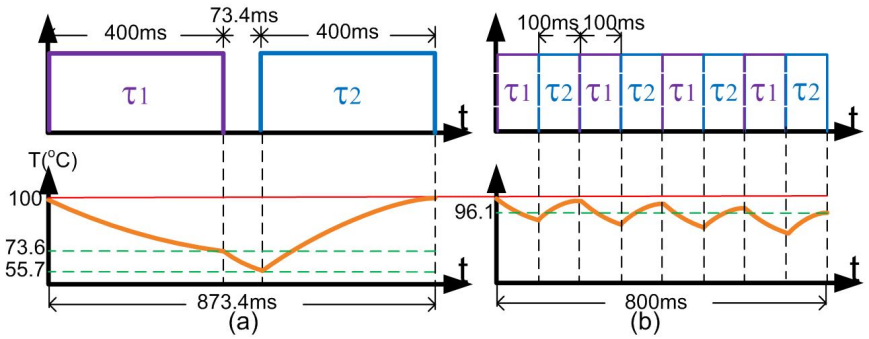


Figure 6.2: Improve the throughput by switching between hot and cool tasks

6.3 Motivational Examples

Consider a task τ with execution time t of $500ms$ and a steady state temperature T_{ss} of 115^oC. Let the peak temperature limit T_{max} to be 100^oC. Assume that the processor has already reached this temperature threshold before task τ starts to run. Then, the temperature constraint will definitely be violated if the execution of τ starts immediately.

To prevent the temperature from exceeding T_{max}, we can turn the processor into the *sleep* mode and let the processor cool down, as illustrated in Figure 6.1(a). Based on the calculation from our system model, the processor has to stay in the *sleep* mode for $418ms$ to make sure its temperature dropped to a *safe* temperature. With this *safe* temperature, if we continue to execute τ, the peak temperature constraint, i.e.

T_{max}, will not be violated.

Alternatively, as shown is Figure 6.1(b), we can divide the execution of τ equally into 5 sections and distribute the *sleep* periods before each of them. In this case, only $12.6ms$ is required for the processor to stay in the *sleep* mode to cool down to the *safe* temperature (i.e. 94.8^oC in this case) for each sub-interval. Consequently, the processor only needs to spend a total of $12.6 \times 5 = 63ms$ in the *sleep* mode to ensure the same maximal temperature constraint. Shorter idle time implies that the latency of the tasks can be reduced and therefore helps to improve the system throughput. This example indicates that it is more effective to insert multiple idle intervals *inside* the task than only at the task boundary to improve the throughput under the same peak temperature constraint.

Now consider another example as shown in Figure 6.2 with a *cool* task τ_1 ($T_{ss} = 68^oC$) and a *hot* task τ_2 ($T_{ss} = 115^oC$). Let us assume that both tasks have the same execution time ($400ms$). Both the initial temperature and the peak temperature constraint are assumed to be 100^oC. To reduce the execution latency without violating the peak temperature constraint, one intuitive approach is to run the *cool* task τ_1 first followed by the *hot* task. However, in this example, running the *cool* task alone cannot bring the temperature low enough such that τ_2 can immediately start to run without exceeding the peak temperature limit. Therefore, a *sleep* period of $73.4ms$ has to be inserted before τ_2 to further cool down the processor which results in a total latency of $873.4ms$ as shown in Figure 6.2(a).

In contrast, one can also divide the execution of τ_1 and τ_2 into 4 sub-intervals and run them alternatively. The schedule and the corresponding temperature curve are shown in Figure 6.2(b). Note that, both τ_1 and τ_2 are successfully executed under the peak temperature limit without inserting any idle interval at all. Moreover, the temperature at the end of the execution is reduced to 96.1^oC . This saved temperature

budget (i.e. 3.9^oC) could be used to further improve the throughput [119] of the ensuing tasks.

It is worth to mention that in the above examples as well as the following discussions, the mode switching overhead is only considered in timing analysis whereas omitted in thermal analysis for simplicity. The reason is that the t_{sw} is sufficiently small so that the temperature variation during t_{sw} is negligible. Moreover, during the mode switching, the processor clock is halted that no workload can be executed. Thus, in fact, the chip temperature during t_{sw} is slightly decreasing, if not ignored. Therefore, if one method can guarantee the temperature constraint without considering the temperature variation in t_{sw}, it is bound to be feasible if we incorporate t_{sw} into the thermal analysis.

The above example clearly shows that, by splitting tasks with different power and thermal characteristics into multiple sections and execute them alternatively, the throughput can be significantly improved. Several questions immediately rise. First, how effective this approach can be, especially when considering the switching overhead between task executions? Second, how can we appropriately choose the number of sections that a task needs to be split to achieve the best performance, i.e. throughput? We answer these questions in the next section.

6.4 Our Approach

In this section, we discuss our approach in detail and present our scheduling algorithms. We first consider a processor with only one *active* mode, i.e. $N = 1$, and assume all tasks are *hot* tasks with respect to a given peak temperature constraint. We then consider a task set consisting of both *hot* and *cool* tasks. Finally, we introduce our approach for a processor with multiple *active* modes, i.e. $N > 1$.

6.4.1 Sleep Mode Distribution for Hot Tasks

We begin our discussion by assuming that a processor has only one *active* mode and one *sleep* mode. We further assume that all tasks in Γ have $T_{ss} > T_{max}$ (i.e. *hot* tasks). When $T_{ss} \leq T_{max}$, the problem becomes trivial since no temperature limit violation can occur. Because Γ is a periodic task set and it is shown [27, 119] that the throughput of Γ is maximized when the temperature at the end of each period equals T_{max}, we can conveniently make the initial temperature of Γ to be the same as the ending temperature of each iteration, and set them to be T_{max}.

Since all tasks are *hot*, starting at T_{max}, we can only bring down the temperature by inserting idle intervals. The question is how long we should insert the interval. The shorter the total length of all idle intervals is, the smaller the overall latency is and thus the larger the throughput can be. To quantify the effectiveness of different choices, we use a metric called the *idle ratio* (Θ) which is defined as the ratio between the time that the processor stays in *sleep* mode and the *active* mode within one period. It is not difficult to see that the smaller the Θ, the larger the throughput.

From the first motivational example above, we can see that the length of overall idle interval can be reduced by splitting each task into multiple—i.e. $m(m > 1)$—sections, and inserting idle intervals in between. In fact, we find that, when the switching overhead is negligible, the larger the m is, the smaller the overall idle interval time is needed. The observation is formulated in the following theorem.

Theorem 6.4.1. *Given a task τ_i, a processor with only one active and one sleep mode, and a maximal temperature constraint T_{max}, assume that $T_{ss}(i) > T_{max}$. Let $\Theta(m)$ represents the idle ratio of a feasible schedule when τ_i is evenly split into m sections. Then,*

- *The idle ratio $\Theta(m)$ is a monotonically decreasing function of m.*

- *The lower bound of the idle ratio Θ_{min} exists as m approaches to infinity such that*

$$\lim_{m->\infty} \Theta(m) = \frac{B(i)}{B_s} \frac{T_{ss}(i) - T_{max}}{T_{max}}. \tag{6.6}$$

Proof: Assume that when $m = 1$, to cool down the processor, t_s seconds of sleep period has to be added before τ_i. To find t_s, we first need to find the *safe* temperature, i.e. $T_{safe}(i)$, of the *hot* task given the peak temperature limit T_{max}. For a given *hot* task, the *safe* temperature is defined such that if the starting temperature is T_{safe}, the ending temperature of the task execution reaches exactly at T_{max}. From the temperature dynamic described in equation (6.5), by letting the ending temperature T_e equal to T_{max}, we can solve for T_0 to obtain the corresponding *safe* temperature of a given task τ_i,

$$T_{safe}(i) = T_{ss}(i) - \frac{T_{ss}(i) - T_{max}}{e^{-B(i)t_i}}. \tag{6.7}$$

Once the *safe* temperature is available, we can calculate how long it takes to stay in the idle mode to cool down the temperature from T_{max} to $T_{safe}(i)$. Again, we can use equation (6.5). This time, the initial and ending temperature are given, the sleep period t_s is the length of the interval that we want to solve,

$$\begin{aligned} t_s &= -\frac{1}{B_s} \cdot \ln(\frac{T_{safe}(i)}{T_{max}}) \\ &= -\frac{1}{B_s} \cdot \ln(\frac{T_{ss}(i) - \frac{T_{ss}(i) - T_{max}}{e^{-B(i)t_i}}}{T_{max}}). \end{aligned} \tag{6.8}$$

Therefore, the *idle ratio* of the original task, i.e. when $m = 1$, can be expressed as

$$\Theta(1) = \frac{t_s}{t_i} = -\frac{1}{B_s t_i} \cdot \ln(\frac{T_{ss}(i) - \frac{T_{ss}(i) - T_{max}}{e^{-B(i)t_i}}}{T_{max}}). \tag{6.9}$$

Now, if the execution of τ_i is evenly split into m sections, we have t_i/m seconds of active period associated with a $safe$ temperature of $T_{safe}(i, m)$, which can be

calculated as

$$T_{safe}(i,m) = T_{ss}(i) - \frac{T_{ss}(i) - T_{max}}{e^{-B(i)\frac{t_i}{m}}}. \tag{6.10}$$

Based on this *safe* temperature, we only need to insert $t_s(m)$ seconds of sleep period before each active period. Specifically, we have

$$t_s(m) = -\frac{1}{B_s} \cdot \ln(\frac{T_{safe}(i,m)}{T_{max}}). \tag{6.11}$$

Now we can formulate the *idle ratio* Θ as a function of m that

$$\begin{aligned} \Theta(m) &= \frac{t_s(m)}{t_i/m} \\ &= -\frac{m}{B_s t_i} \cdot \ln(\frac{T_{safe}(i,m)}{T_{max}}) \\ &= \frac{1}{B_s t_i} \cdot \ln(\frac{T_{max}}{T_{safe}(i,m)})^m. \end{aligned} \tag{6.12}$$

To prove Theorem 6.4.1, we only need to show that the first order derivative of $\theta(m)$ is always less than zero, i.e. $\frac{d\Theta(m)}{dm} < 0$. Therefore, we have

$$\begin{aligned} \frac{d\Theta(m)}{dm} &= \frac{1}{B_s t_i} \cdot (\frac{T_{safe}(i,m)}{T_{max}})^m \\ &\cdot \quad m(\frac{T_{max}}{T_{safe}(i,m)})^{m-1} \cdot \frac{(-1) \cdot T_{max} \cdot \frac{dT_{safe}(i,m)}{dm}}{T_{safe}(i,m)^2}. \end{aligned} \tag{6.13}$$

On the right-hand-side of equation (6.13), the parameter B_s, t_i, m, T_{max}, T_{safe} are all greater than zero, therefore the sign of $\frac{d\Theta(m)}{dm}$ depends upon the term $\frac{dT_{safe}(i,m)}{dm}$. By observing equation (6.10), it is not difficult to see that the function $T_{safe}(i,m)$ is monotonically increasing with m. The physical meaning of this equation is that with a larger m (smaller active period), the required *safe* temperature becomes higher to

guarantee the same peak temperature limit. Therefore, we have $\frac{dT_{safe}(i,m)}{dm} > 0$, thus $\frac{d\Theta(m)}{dm} < 0$ and that $\Theta(m)$ monotonically decreases with m is proved.

We next find the lower bound of $\theta(m)$. As $m \to \infty$, we denote the active time and sleep time in each division as t'_i and t'_s, respectively. The corresponding *safe* temperature is represented as $T'_{safe}(i)$. Then, based on equation (6.5), we obtain the following relationship:

$$t'_s = -\frac{1}{B_s} \ln(\frac{T'_{safe}(i) - 0}{T_{max} - 0}), \tag{6.14}$$

$$t'_a = -\frac{1}{B(i)} \ln(\frac{T_{max} - T_{ss}(i)}{T'_{safe}(i) - T_{ss}(i)}). \tag{6.15}$$

Because a shorter active period requires a higher *safe* temperature, in the extreme case when $m \to \infty$, we have $T'_{safe}(i) \to T_{max}$. Then, the lower bound of Θ can be calculated as

$$\Theta_{min} = \lim_{T'_{safe} \to T_{max}} (\frac{1}{B_s} \ln(\frac{T'_{safe}(i)}{T_{max}}))/(\frac{1}{B(i)} \ln(\frac{T_{max} - T_{ss}(i)}{T'_{safe}(i) - T_{ss}(i)})). \tag{6.16}$$

Apparently, when $T'_{safe} \to T_{max}$, the above limit calculation involves an indeterminate term ($\frac{0}{0}$ type). Therefore, we apply the l'Hopital's rule [2] to find the first order derivative of the numerator and denominator of equation (6.16), respectively. Then, we have

$$\begin{aligned} \Theta_{min} &= \lim_{T'_{safe} \to T_{max}} (\frac{dt'_s}{dT'_{safe}})/(\frac{dt'_a}{dT'_{safe}}) \\ &= \frac{B(i)}{B_s} \frac{T_{ss}(i) - T_{max}}{T_{max}}. \end{aligned} \tag{6.17}$$

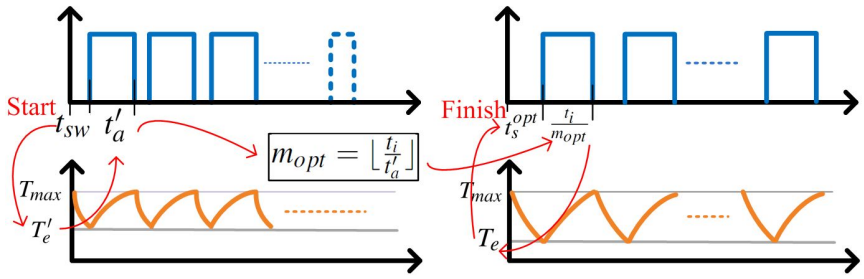


Figure 6.3: Illustrate the procedure to find the appropriate number of m for our sleep time distribution method by considering the non-negligible transition overhead

Hence, Theorem 6.4.1 is proved. □

Theorem 6.4.1 implies that the smaller we divide a task, the shorter the idle interval is needed. However, the impact of distributing idle intervals between task executions eventually can be saturated as we increase the number of m.

Based on the above discussion, in order to maximize the throughput of a processor, we should divide the execution of task τ_i into as many sub-intervals as possible. However, since there exists a lower bound of $\Theta(m)$, the benefit of dividing the task becomes saturated as m increases. Furthermore, as $m \to \infty$, the context switching overhead cannot be ignored anymore, no matter how small it can be. The question then becomes how to determine the optimal "m" for each *hot* task.

Assuming the timing overhead of each context switching is t_{sw}, we derive a method, i.e. Algorithm 4, to find the optimal sleep distribution pattern. Specifically, we want to find the optimal number of sub-intervals m_{opt} and the sleep time in each sub-interval t_s^{opt}. The procedure is illustrated in Figure 6.3.

Starting from T_{max}, the corresponding ending temperature of the sleep period can be obtained from equation (6.5),

$$T'_e = T_{amb} + (T_{max} - T_{amb})e^{-B_s t_{sw}}. \tag{6.18}$$

Based on T'_e, the duration of the subsequent *active* mode is

$$t'_i = -\frac{1}{B(i)} \ln(\frac{T_{max} - T_{ss}(i)}{T'_e - T_{ss}(i)}). \tag{6.19}$$

Accordingly, we have $m_{opt} = \lfloor \frac{t_i}{t'_i} \rfloor$ (using the floor function to make sure m_{opt} is an integer). Once m_{opt} is available, the duration of each active sub-interval can be determined, i.e. $\frac{t_i}{m_{opt}}$. The ending temperature of the corresponding sleep period can also be obtained as

$$T_e = T_{ss}(i) - \frac{T_{ss}(i) - T_{max}}{e^{-B(i)\frac{t_i}{m_{opt}}}}. \tag{6.20}$$

Finally, the minimized sleep time per sub-interval t_s^{opt} can be solved from equation (6.8) by replacing $T_{safe}(i)$ with T_e in equation (6.20):

$$t_s^{opt}(i) = -\frac{1}{B_s} \ln(\frac{T_{ss}(i)(e^{-B(i)\frac{t_i}{m_{opt}}} - 1)}{e^{-B(i)\frac{t_i}{m_{opt}}}} - 1). \tag{6.21}$$

The final schedule generated by our method will be t_s^{opt} seconds of sleep period followed by $\frac{t_i}{m_{opt}}$ seconds of task execution. By repeating this pattern by m_{opt} times, the latency of task τ_i can be minimized while the given peak temperature limit T_{max} is guaranteed.

Although Algorithm 4 is targeting at a single task, it is applicable to a task set consisting of multiple *hot* tasks, since the optimization procedure can be conducted on each individual task separately. For a special case that a task set consisting of hot tasks with homogeneous power, the task set can be considered as a single task with the execution time $t_{exe} = \sum t_i, (\tau_i \in \Gamma)$. The optimal sleep distribution pattern can be found conveniently by using Algorithm 4.

Algorithm 4 Sleep mode distribution for single hot task

1: ***SleepDistribution***(TASK τ_i)
2: //Set initial value of $t_s^{opt} = t_{sw}$
3: $t_s^{opt} = t_{sw}$;
4: //Find the ending temperature of sleep mode
5: $T'_e(t_{sw}) = T_{amb} + (T_{max} - T_{amb})e^{-B_s t_{sw}}$;
6: //Find the execution time of the ensuing active period
7: $t'_a = -\frac{1}{B(i)} \ln(\frac{T_{max} - T_{ss}(i)}{T'_e(t_{sw}) - T_{ss}(i)})$;
8: //Find optimum m
9: $m^{opt} = \lfloor \frac{t_i}{t'_a} \rfloor$;
10: //Find the minimal idle interval per division
11: $t_s^{opt} = -\frac{1}{B_s} \cdot \ln(\frac{T'_e}{T_{max}})$;
12: $latency = m \cdot t_s^{opt} + t_i$;
13: return ($latency$);

6.4.2 Improving Throughput by Task Switching

In this subsection, we extend our discussion to a task set consisting of both *hot* and *cool* tasks. First, let us consider only two tasks, one *hot* task and one *cool* task. Recall that as implied by the second motivational example, dividing both tasks into m ($m > 1$) sections, and alternating the execution of both tasks helps to improve the throughput. However, this method cannot always guarantee that the entire temperature curve can stay below the temperature threshold. Whether this task switching scheme work or not depends on the power consumption and the duration of the *cool* task and the *hot* task, respectively. Therefore, in this subsection, we develop a systematic way to determine whether the task switching scheme can be applied to a hot/cool task pair, and if yes, how to find the optimal number of task switching to achieve the maximum throughput.

Given a task pair, i.e. τ_i and τ_j ($T_{ss}(i) < T_{max} < T_{ss}(j)$), both τ_i and τ_j are equally divided into m divisions and alternatively executed (with τ_i first). In this scenario, if the initial temperature is T_{max}, the temperature is first cooled down to certain point by the *cool* task. Then it goes up by running the *hot* task. Note that even we apply

the task switching scheme, the peak temperature may still have chance to either stay below or go above the predefined temperature limit.

In order to determine whether the peak temperature constraint can be satisfied, we define a term called *critical temperature*, i.e. T_c, which is the temperature when completing the first division of τ_j, i.e. *hot* task. To determine whether the task pairing can guarantee the temperature constraint for an arbitrary m, we only need to check the *critical temperature*. If $T_c > T_{max}$, then apparently we have temperature limit violation. In contrast, if $T_c \leq T_{max}$, we can guarantee that the entire temperature curve can stay below T_{max} for the remaining part of the task execution. This is because if the initial temperature of the second cool task division, i.e. T_c, is less than the previous initial temperature T_{max}, it results in an even lower starting temperature for the second hot task division. By repeating this pattern, the entire temperature curve continues to decrease from T_{max}.

In fact, a similar theorem as Theorem 6.4.1 can be established for a task set consisting both *hot* and *cool* tasks.

Theorem 6.4.2. *Given*

- *a task pair, i.e.* τ_i *and* τ_j*,*
- *a processor with only one active and one sleep mode,*
- *a maximal temperature constraint* T_{max}*,*

if $T_{ss}(i) < T_{max} < T_{ss}(j)$ *and both* τ_i *and* τ_j *are equally divided into* m *sections and alternatively executed (with* τ_i *first). Then the* critical temperature *w.r.t different* m*, i.e.* $T_c(i, j, m)$*, is a monotonically decreasing function of* m *if*

$$m < \frac{B(i)t_i}{\ln(\frac{K_1}{K_2} \cdot \frac{T_{max} - T_{ss}(i)}{T_{ss}(j) - T_{ss}(i)})}, \tag{6.22}$$

where $K_1 = B(i)t_i + B(j)t_j$ *and* $K_2 = B(j)t_j$.

Proof: To prove Theorem 6.4.2, we need to find the range of m that makes the first order derivative of $T_j(i, j, m)$ less than zero. Given the initial temperature T_{max}, based on equation (6.5), the ending temperature of task τ_i after $\frac{t_i}{m}$ seconds of execution can be expressed as

$$T_i(i, j, m) = T_{ss}(i) + (T_{max} - T_{ss}(i))e^{-B(i)\frac{t_i}{m}}. \tag{6.23}$$

Similarly, the ending temperature of task τ_j after $\frac{t_j}{m}$ seconds can be formulated as

$$T_c(i, j, m) = T_{ss}(j) + (T_i(i, j, m) - T_{ss}(j))e^{-B(j)\frac{t_j}{m}}. \tag{6.24}$$

After substituting $T_i(i, j, m)$ with equation (6.23), we have

$$\begin{aligned} T_c(i, j, m) &= T_{ss}(j) + (T_{ss}(i) - T_{ss}(j))e^{\frac{-K_2}{m}} \\ &+ (T_{max} - T_{ss}(i))e^{\frac{-K_1}{m}}. \end{aligned} \tag{6.25}$$

By solving the first order derivative of $T_c(i, j, m)$, we have

$$\begin{aligned} \frac{dT_c(i, j, m)}{dm} &= \frac{K_2}{m^2} \cdot (T_{ss}(i) - T_{ss}(j))e^{\frac{-K_2}{m}} \\ &+ \frac{K_1}{m^2} \cdot (T_{max} - T_{ss}(i))e^{\frac{-K_1}{m}}. \end{aligned} \tag{6.26}$$

If we let $\frac{dT_c(i,j,m)}{dm} < 0$ and solve the inequality, then

$$\begin{aligned}
\frac{dT_c(i,j,m)}{dm} &< 0, \\
K_1(T_{max} - T_{ss}(i))e^{\frac{-K_1}{m}} &< K_2(T_{ss}(j) - T_{ss}(i))e^{\frac{-K_2}{m}}, \\
\frac{K_1}{K_2} \cdot \frac{T_{max} - T_{ss}(i)}{T_{ss}(j) - T_{ss}(i)} &< e^{\frac{B(i)t_i}{m}}, \\
m &< \frac{B(i)t_i}{\ln(\frac{K_1}{K_2} \cdot \frac{T_{max} - T_{ss}(i)}{T_{ss}(j) - T_{ss}(i)})}. \qquad (6.27)
\end{aligned}$$

Hence, Theorem 6.4.2 is proved. □

Theorem 6.4.2 implies that given a *cool/hot* task pair, one can always try to find a feasible schedule by increasing m within the bound specified by equation (6.27), or

$$m^1_{bound} = \lfloor \frac{B(i)t_i}{\ln(\frac{K_1}{K_2} \cdot \frac{T_{max} - T_{ss}(i)}{T_{ss}(j) - T_{ss}(i)})} \rfloor. \qquad (6.28)$$

Now the problem becomes how to judiciously choose the appropriate m to maximize the throughput, when the context switching overhead is considered. Note that, the non-negligible switching overhead also imposes a bound to the choice of m. The total amount of switching time associated with m task switchings is $m \cdot t_{sw}$. If the original idle interval required to cool down a *hot* task from T_{max} to T_{safe} is t_s, we must have $m \cdot t_{sw} < t_s$ in order to further reduce the latency. Therefore, we set another bound for m that

$$m^2_{bound} = \lfloor \frac{t_s}{t_{sw}} \rfloor. \qquad (6.29)$$

Then, the upper bound of m is defined as

$$m_{max} = MIN\ (m^1_{bound},\ m^2_{bound}). \qquad (6.30)$$

The hot/cool task pairing algorithm is depicted in Algorithm 5. The optimal m

can be found by sequentially search from 1 to m_{max} (among the positive integers). The searching is stopped as soon as we find an m so that $T_c(i, j, m) \leq T_{max}$. Note that, if the ending temperature at $m = m_{max}$ still cannot satisfy the temperature constraint, the task switching scheme fails. Thus, we first test if $T_c(i, j, m_{max}) < T_{max}$, and the searching process starts only if the result is true.

Algorithm 5 Pairing hot/cool task

```
1:  TaskParing(TASK τ_i, TASK τ_j)
2:      //Find the upper bound of m
3:      m_max = MIN (m^1_bound, m^2_bound);
4:      //Find the T_c at m = m_max, check constraint
5:        if (T_c(i, j, m_max)) ≤ T_max
6:      //Find m_opt by sequential search from m = 1
7:          for k = 1; k < m_max; k++
8:            if (T_c(i, j, k)) ≤ T_max
9:      //If constraint satisfied, get m
10:                 m_opt = k;
11:                 Latency = m_opt · t_sw + t_i + t_j;
12:                 return (Latency);
13:           endif
14:         endfor
15:       else return 0
```

We present the proposed throughput maximization algorithm in Algorithm 6, for processors with one active and one sleep mode. Our algorithm works as follows: First, tasks are classified into *cool* tasks and *hot* tasks based on their power/thermal characteristics and the given peak temperature constraint. Then, we put *cool* tasks in a queue Q_c and sort them in a decreasing order based on the ending temperature (assuming the execution starts at temperature T_{max}). The *hot* tasks are put in Q_h and sorted in an increasing order based on their *safe* temperatures. Starting from the initial temperature T_{max}, the task at the beginning of Q_h attempts to pair with the head task in Q_c. If this task pairing is feasible, the m_{opt} can be obtained by a sequential search. The latency of the hot/cool task pair is calculated by $TaskParing()$ defined in Algorithm 5. Then, both tasks are marked as scheduled and their ending

Algorithm 6 Throughput maximization without DVS

Input: $\Gamma = \{\tau_1, \tau_2, ...\tau_n\}$
Initialization: classify all tasks into cool/hot tasks based on T_{ss};
Find T_{end} for all cool tasks;
Find T_{safe} for all hot tasks;
Sort cool tasks into Q_c in decreasing order of T_{end};
Sort hot tasks into Q_h in increasing order of T_{safe};
$T_{ini} = T_{max}$;
for i=1:length(Q_h)
  for j=1:length(Q_c)
    if (tasks NOT 'scheduled') && (pairing is feasible)
      $Latency = Latency +$ ***TaskParing***($Q_c[j]$, $Q_h[i]$);
      Mark $Q_c[j]$ and $Q_h[i]$ as 'scheduled';
      Update initial temperature: $T_{ini} = T_{end}(i, j, m_{opt})$;
    endif
  endfor
endfor
for i=1:length(Q_h)
  if ($Q_h[i]$ is not 'scheduled')
    $Latency = Latency +$ ***SleepDistribution***($Q_h[i]$);
    Mark $Q_h[i]$ as 'scheduled';
endfor
for i=1:length(Q_c)
  if ($Q_c[i]$ is not 'scheduled')
    $Latency = Latency + t_{Q_c[i]}$;
    Mark $Q_c[i]$ as 'scheduled';
endfor
Return: ($Latency$);

temperature after m_{opt} times of switching needs to be updated as the new initial temperature of the next attempted pairing.

To find the ending temperature, we derive a closed-form formula based on [88]. If there exists m_{opt} times of task switching between τ_i and τ_j, the ending temperature of the schedule can be formulated as

$$T_{end}(i,j,m) = T_{ini} + \frac{(T_c(i,j,m) - T_{ini}) \cdot (1 - K_3^m)}{1 - K_3}, \tag{6.31}$$

where $K_3 = e^{-B(i)\frac{t_i}{m} - B(j)\frac{t_j}{m}}$ and T_{ini} is the initial temperature of the task pairing between task i and j. Based on the above equation, the ending temperature after m_{opt} times of task switching can be obtained by replacing m with m_{opt}.

For a given *hot* task $Q_h[i]$, if the attempted task pairing fails with the pth cool task, i.e. $Q_c[p]$, it is still possible to make a feasible combination with the qth ($p \neq q$) *cool* task, i.e. $Q_c[q]$. Therefore, the *hot* task is left in the Q_h until the end of the iteration to get chances to be matched with all *cool* tasks. Finally, after the attempted task pairing procedure, if there are still tasks left in Q_c or Q_h, the *hot* tasks are executed with *SleepDistribution*() introduced in Section 6.4.1 and the cool tasks are simply attached at the end of the final schedule.

6.4.3 Improving Throughput by DVS

In previous discussion, we assume that a processor has only one *active* mode. In this subsection, we consider a more complex processor model, i.e. the processor with $N(N > 1)$ different *active* modes, and the processor can change its working mode dynamically. Employing DVS is a double-edged sword in terms of throughput maximization. On one hand, reducing the supply voltage slows down the task execution and reduces the throughput. On the other hand, reducing the supply voltage helps to reduce the power consumption and thus the thermal pressure. How to make an

appropriate trade-off needs careful analysis.

Given a *hot* task τ_i, if the execution time at the highest speed level k is $t_i(k)$, then the corresponding execution time at $(k-l)$th mode is calculated by

$$t_i(k-l) = \frac{f_k}{f_{k-l}} t_i(k), \tag{6.32}$$

where f_k and f_{k-l} are the clock frequencies associated with V_{dd} level k and $k-l$, respectively. From equation (6.8), the required sleep time of task τ_i under the supply voltage level $k-l$ can be obtained and expressed as

$$t_s(i,k-l) = -\frac{1}{B_s} \ln\left(\frac{T_{ss}(i,k-l)(e^{-B(i,k-l)\cdot t_i(k-l)} - 1)}{e^{-B(i,k-l)\cdot t_i(k-l)}} - 1\right). \tag{6.33}$$

Note that if the steady state temperature of task τ_i at the $(k-l)$th speed level is equal to or less than T_{max}, then it becomes a *cool* task and thus $t_s(i,k-l) = 0$.

By combining the actual execution time and the required cooling period, the overall latency of τ_i when executed at the speed level $k-l$ is formulated as

$$t(i,k-l) = t_s(i,k-l) + t_i(k-l). \tag{6.34}$$

One straightforward way to determine the optimal speed level of a given task is to calculate the overall latency of the task under all supply voltage levels. Then, the speed level that leads to the least latency will be selected as the optimal speed k_{opt}. In fact, we should also take advantage of the proposed sleep distribution option to see if we can further improve the throughput. Given a task, we apply Algorithm 7 to find the optimal speed level in terms of the latency minimization.

In Algorithm 7, given a *hot* task, we first calculate the overall latency by applying our sleep distribution method proposed in Section 6.4.1 and the optimal speed level

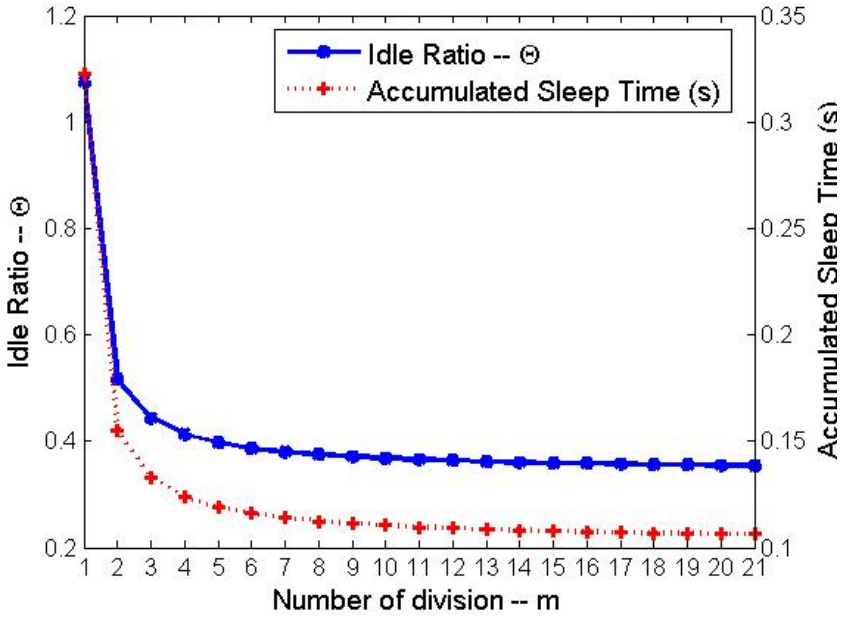
Algorithm 7 Find the optimal speed level for task

1: ***Calculate-Speed-Opt***(TASK τ)
2: //Find total latency by sleep distribution at the Max.Spd
3: $Latency[SpdMax-1]$=***SleepDistribution***(τ);
4: //Optimum speed initially set as the Max.Spd
5: $SpdOpt = SpdMax-1$;
6: ***for*** $L = SpdMax-2$; $L >= 0$; $L--$
7: ***if*** $T_{ss} > T_{max}$
8: $Latency[L] =$ ***SleepDistribution***(τ);
9: ***else***
10: $Latency[L] = t(\tau, L)$; Eq. 6.34
11: ***endif***
12: ***if*** $Latency[L] < Latency[L+1]$
13: $SpdOpt = L$;
14: ***endif***
15: ***endfor***
16: return ($SpOpt$);

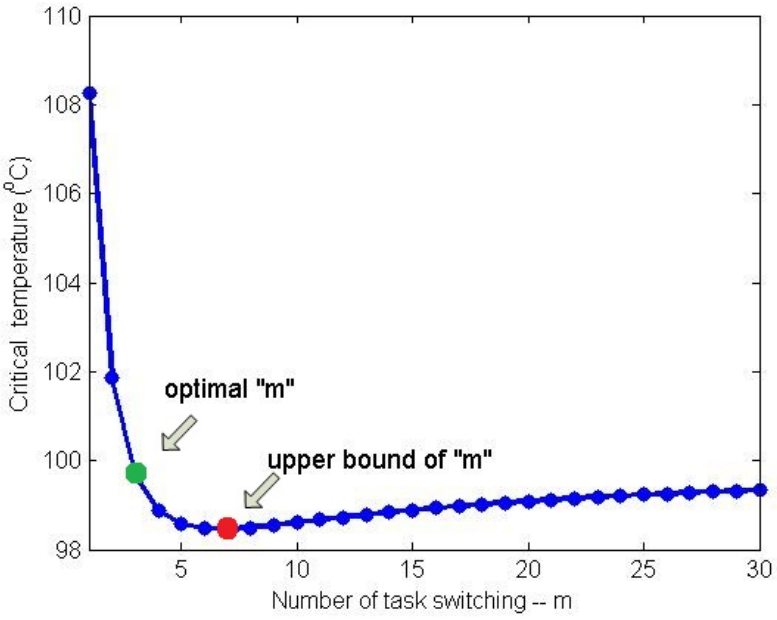
is initially set as the maximum speed. Then we reduce the speed level one by one and check if there are improvement in terms of overall latency. Finally, the optimal speed level is returned after we iterate through all available speed levels. It is worth to mention that, each time we reduce the speed level, the corresponding steady state temperature of a given task is evaluated because a *hot* task might transform into a *cool* task. If that is the case, no sleep time is needed and instead, we only need to set the latency as the actually execution time of the task at that speed level.

6.5 Experimental Results

In this section, we validate the theorems and show the performance of the proposed approaches through a set of simulations. We again use the processor model introduced in Chapter 3. The mode switching overhead is assumed to be 5ms [110]. We let the ambient temperature and the maximal temperature limit to be 25^oC and 100^oC [119], respectively unless otherwise specified. Also, for simplicity reason, we assume that for a given task, the *activity factor* is equal to its *leakage factor*.

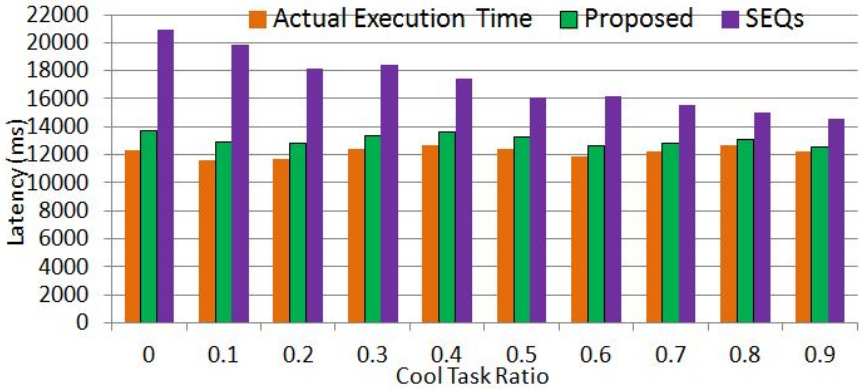


(a) Theorem validation: The execution latency of a task is monotonically decreased as we increase the number that we divide the task, i.e. m.

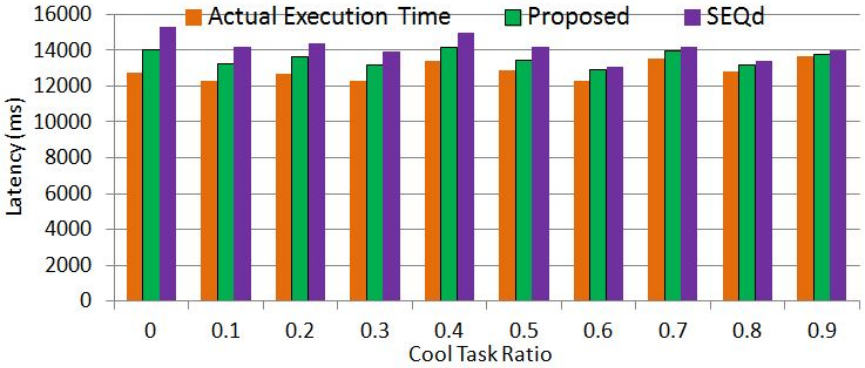


(b) Theorem validation: The critical temperature is a monotonically decreasing function of m when $m < m_{bound}$.

Figure 6.4: Theorem validation



(a) Processor without DVS



(b) Processor with DVS

Figure 6.5: Latency comparisons for synthetic task sets of different scheduling approaches

6.5.1 Theorem Validation

To validate the conclusion drawn in Theorem 6.4.1, we run a *hot* task τ_1 ($t_i = 300ms$, $T_{ss} = 138^oC$ @$V_{dd} = 1.2v$). The idle ratio Θ as well as the total sleep time are plotted in Figure 6.4(a), as we increase m from 1 to m_{max}. The result conforms to the conclusion in Theorem 6.4.1 that Θ monotonically decreases with m. When $m = 1$ the schedule is identical to the one proposed in [119] that $322ms$ sleep time is required to cool down the processor before τ_1 starts to run (thus $\Theta = 1.07$). As m increases, the total sleep time and thus the *idle ratio* is decreasing and reaches the minimal at $m = m_{max} = 21$. The final latency is $310ms$ compared with $622ms$ by the approach in [119], a 50% reduction is achieved in this example.

We next validate Theorem 6.4.2 by running a *cool* task τ_2 ($T_{ss} = 81.7^oC$) followed by a *hot* task τ_3 ($T_{ss} = 115.9^oC$) without introducing any sleep interval in between. The execution time of the *cool* task and *hot* task are $700ms$ and $400ms$, respectively. By using the proposed task switching method, the *critical temperature*, i.e. T_c, is plotted in Figure 6.4(b). We can see that as m increases from 1 to 30 (an arbitrarily chosen large number), the critical temperature first drops and then increases steadily. The upper bound of T_c's decreasing region is calculated by using equation (6.22), i.e. 7, in this case. Again, the result conforms to Theorem 6.4.2 that T_c is monotonically decreasing with m when m is less then the *bound*. From Figure 6.4(b), we can also see that $T_c(2, 3, m)$ drops drastically at the first a few steps and becomes relatively stable when m is further increased. It implies that, in this particular case, m_{opt} can be found within 3 rounds of evaluation (since $T_{max} = 100^oC$ and $T_c(2, 3, 3) < 100^oC$).

6.5.2 Latency Minimization for Synthetic Task Sets

In this subsection, we evaluate the performance of the proposed method in terms of latency minimization. We created 10 representative task sets each consisting of 20 randomly generated tasks with execution time and utility factor μ uniformly distributed within $[100, 1000ms]$ and $[0.4, 1]$, respectively. Based on our thermal model, the steady state temperature of these tasks are ranging from 62^oC to 145^oC. For each task set, we specify the ratio of the number of cool tasks versus the total number of tasks (i.e. 20), and vary this ratio from 0 to 0.9 with 0.1 increment.

For a processor without DVS, we compare our approach with a heuristic similar to the one proposed in [119], i.e. the SEQ_s method. In this case, we set the processor to the highest available speed level. We recorded the average total latency achieved by both methods in each test case and plotted in Figure 6.5(a). We also provide the

actual time that the processor spends in task execution as references. As shown in Figure 6.5(a), the proposed method consistently outperforms the SEQ_s method in all test cases. Compared with the SEQ_s method, on average our method reduces the latency by 23.3% (up to 35.7%). From Figure 6.5(a), we can see that, in terms of overall latency reduction, the performance of the proposed approach is relatively stable compared with the SEQ_s method, which is sensitive to the amount of *cool* tasks available in a given task set. Since in the SEQ_s method, if the number of *cool* tasks are small, one has to insert a large sleep period before each *hot* task to reduce the temperature. Under the same circumstance, in contrast, the proposed approach can rely on sleep distribution to reduce the latency even without *cool* tasks.

Next, we repeat the entire procedure for a DVS enabled processor and compare our DVS scheduling approach with the one similar to the SEQ_d method [119] (a DVS approach as well). As shown in Figure 6.5(a), the proposed method still achieves much better performance. Specifically, compared with the SEQ_d method, on average, our method reduces the latency and idle time by 5.3% and 46.9%, respectively. We notice that, given a processor with the DVS capability, the margins of the performance differences between our approach and the SEQ_d method are significantly reduced compared with non-DVS cases. The reason is that, with the support of speed scaling, the original *hot* tasks (at the maximal speed) have a great chance to become cool tasks after speed reduction. As a result, for the SEQ_d method, instead of inserting large chunk of sleep period, only a linear increase of overall latency is incurred.

6.5.3 Latency Minimization for Real Benchmarks

In this subsection, we evaluate the performance of the proposed scheduling algorithms in terms of latency minimization by using real benchmark programs. Specifically, we choose 12 different benchmark programs (shown in Table 6.1) from *MediaBench* [3]

Table 6.1: Selected benchmark programs and their parameters

Benchmarks	Avg.Power(w)	Exe.Time(s)	Stead State Temp. (^oC)	Hot or Cool
galgel	52.2	0.046	80.2	Cool
bzip2	52.7	0.037	80.2	Cool
crafty	52.1	0.028	79.1	Cool
gap	55.5	0.04	84.6	Cool
gcc	55.1	0.043	83.5	Cool
mcf	53.5	0.028	81.3	Cool
crc	84.3	0.214	127.9	Hot
dijksstra	76.5	0.136	115.4	Hot
ffti	74.9	0.41	102.7	Hot
gsm	77.8	0.373	116.7	Hot
qsort	76.7	0.475	115.4	Hot

and *SPEC2000* benchmark suites [7]. The total execution time of the benchmark are individually obtained by *Simplescalar*[1] [6] (at 1.2GHz). We use *Wattch* [23] to get the averaged power consumption of each program and then normalize to the highest one to get the *activity factor* μ of each task. Then, based on these tasks' parameters as well as our system model, we can calculate the steady state temperature of each individual task and divide them into *hot* and *cool* tasks (the temperature limit is 100^oC in this experiment).

In our experiment, 6 representative task sets, i.e. $T1 - T6$, were tested. Specifically, Task set $T1$ includes all 12 tasks we selected. Task set $T6$ contains only the hot tasks. Task set $T2 - T5$ each consists of 8 tasks with different hot/cool task ratios.

For each task set, the overall latency achieved by using the proposed methods and the approaches in [119] were collected for a processor with and without DVS capability. We normalized the overall latency obtained by different methods to the actual execution time of each individual task set and plotted the results in Figure. 6.5.3.

From Figure 6.5.3, we can see that, the results obtained from real benchmark

[1]Default configuration was used to setup the processor in our experiment.

Table 6.2: Representative task sets consisting of real benchmark programs

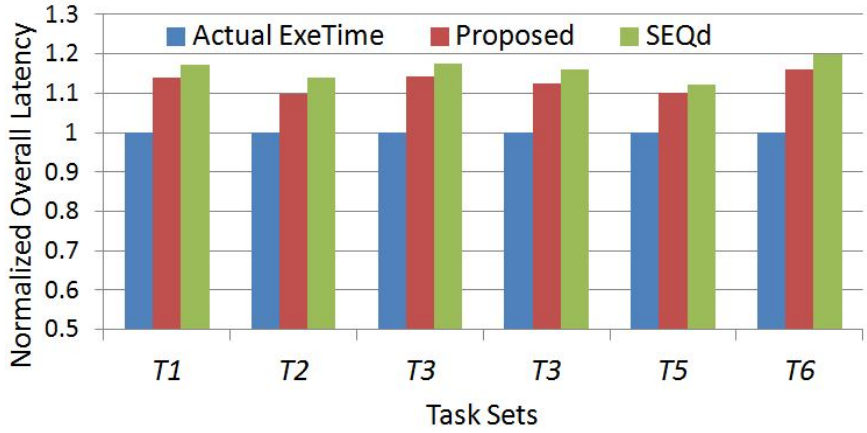
Task Set	Benchmarks
T1	galgel,bzip2,crafty,gap,gcc,mcf,twolf,crc,dijkstra,ffti,gsm,qsort
T2	bzip2,crafty,gap,crc,dijkstra,ffti,gsm,qsort
T3	galgel,bzip2,crafty,gap,dijkstra,ffti,gsm,qsort
T4	qsort,ffti,dijkstra,bzip2,galgel,gcc,twolf,mcf
T5	ffti,qsort,bzip2,crafty,gap,gcc,mcf,twolf
T6	crc,dijkstra,ffti,gsm,qsort

programs are similar to those of synthetic benchmarks with and without considering DVS. The proposed algorithms again consistently outperform the base line approaches, i.e. the SEQ_s and SEQ_d method. For non-DVS case, compared with THE SEQ_S method, our method reduces the overall latency by 14.4% on average (up to 21% for $T6$ consisting of all hot tasks). On the other hand, for processor with DVS capability, our method achieves 3% latency reduction on average.

6.5.4 Feasibility Improvement

We next investigate the performance of the proposed method in terms of feasibility improvement for a processor without DVS capability. Recall that a task is defined as infeasible if the required *safe* temperature is below the ambient temperature. We use the same parameters to randomly generate task sets without specifying the number of cool tasks in each task set. Instead, we set the ambient temperature as a variable and change it from 25^oC to 65^oC with a step size of 10^oC. Under each ambient temperature condition, we generated 10 task sets each including 100 tasks. The average feasibility ratio (the number of feasible tasks divided by 100) achieved by the proposed method and the SEQ_s method were recorded and plotted in Figure 6.7. The proposed method completes all tasks without infeasible task and thus improves the feasibility ratio by 21% on average compared with the SEQ_s method.

(a) Processor without DVS



(b) Processor with DVS

Figure 6.6: Latency comparisons for real benchmarks of different scheduling approaches

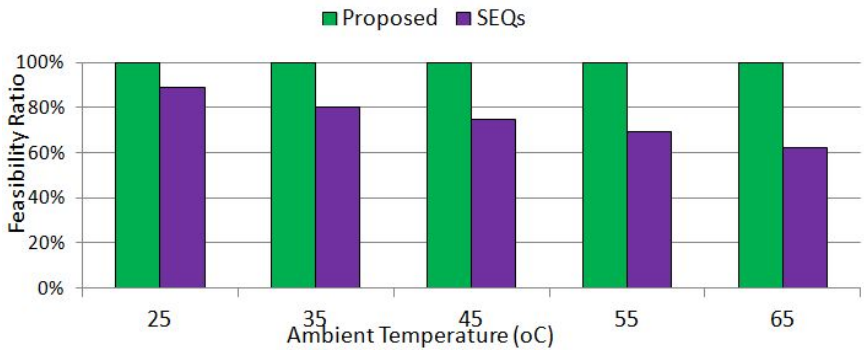


Figure 6.7: Feasibility comparisons of different scheduling approaches

6.6 Summary

In this chapter, we study the problem on how to maximize the throughput of a periodic real-time system under a given peak temperature constraint. We incorporate the interdependency between the leakage, temperature and supply voltage into analysis and assume that different tasks in our system may have different power and thermal characteristics. Two algorithms are presented in this chapter. The first one is built upon processors that can be either in active or sleep mode. By judiciously selecting tasks with different thermal characteristics as well as alternating the processor active/sleep mode, the sleep period required to cool down a processor is kept at a near optimum level and as the result, the throughput is maximized. We further extend this approach for processors with dynamic voltage/frequency scaling (DVS) capability. Our experiments on large amount of synthetic as well as real benchmark programs show that the proposed methods not only consistently outperform the existing approaches in terms of throughput maximization, but also significantly improve the feasibility of a task set when more stringent temperature constraints are imposed.

CHAPTER 7

CONCLUSIONS

In this chapter, we summarize our research presented in this dissertation and discuss possible future work of this research.

7.1 Concluding Remarks

The continuous shrinking of the semiconductor transistor feature size together with the increasingly complicated circuit architecture, have resulted in the exponential increase of power density, which has imposed enormous challenges and threatens to slow down this progress. The soaring power consumption has posed immediate challenges for system designers on how to extend the battery life of portable devices and reduce the energy bills of power-rich systems, respectively.

Moreover, the elevated power consumption directly translates into high system temperature, which not only increases the packaging and cooling cost, adversely affects the performance but also reduces the lifespan of a system. More importantly, it increases the temperature-dependent leakage power consumption, which is becoming one of the major components of the overall power in DSM domain digital integrated circuits. A vicious circle exists that high power converts to high temperature and high temperature in turn increases the leakage power. Apparently, power/thermal-aware design techniques are urgently needed at each design abstraction level. And the crucial leakage/temperature dependency has to be incorporated into the study.

In this dissertation, we present our research efforts on how to apply the real-time scheduling techniques to solve various system-level, resource-constrained, design-optimization problems. We first introduce a set of simple yet effective system-level models to capture the processor's thermal dynamic as well as the leakage/temperature dependency. The linear-approximation leakage power model developed in this re-

search helps to greatly simplify the complexity of temperature calculation while achieves reasonably good accuracy, e.g. 1.3% average relative error for leakage calculation and less than 4.8% relative error when calculating the temperature. Then, based on these models, we focus on three closely related, but distinctly different problems: (*i*) temperature minimization under timing constraint, (*ii*) energy minimization under peak temperature constraint, and (*iii*) throughput maximization under peak temperature constraint.

Specifically, we first present a novel real-time scheduling scheme, based on an existing algorithm, the *M-Oscillating*, that oscillates between the high and low processor speeds to minimize the peak temperature of a processor when executing a periodic task set. Different from the original algorithm proposed in [28], in this research, we incorporate the non-negligible processor mode switching overhead into analysis and present a fast searching algorithm that can be used to efficiently find the appropriate number of mode switchings that can lead to the minimized peak temperature.

We next solve the energy minimization scheduling problem based on the *M-Oscillating* concept. Specifically, We derived a novel, closed-form energy estimation method that can be used to accurately and efficiently calculate the energy consumption of a candidate schedule at both the transient and the thermal steady state. Based on the proposed energy calculation method, we then developed two scheduling techniques, i.e. an off-line method for a periodic task set and an on-line method targeting at an aperiodic task set, to minimize the overall energy consumption of real-time systems. Our experimental results show that the proposed energy estimation method can achieve up to 177X speedup compared with an existing approach [73] while still maintaining high accuracy (with relative error no more than 4.1%). With a large number of different test cases, the proposed off-line energy minimization scheduling method consistently outperforms two existing approaches, e.g. the Naive approach

and the *Pattern-based* approach [110], by 26.5% and 15.3%, respectively. Our on-line approach, on the other hand, also shows superior performance in terms of over energy reduction. Compared with the *Pattern-based* approach and the *On-line DVS* approach [64], on average, our method gains 14% and 10% additional energy savings, respectively. To the best of our knowledge, this is the first thermal-aware scheduling technique that can guarantee the hard real-time deadlines for an aperiodic task set.

Finally, we study the problem on how to maximize the throughput for a periodic real-time system under a given peak temperature constraint. We observe that the overall latency when running a task can be reduced if we split the task execution and insert the cooling periods in between. Moreover, we notice that by frequently switching between the execution of tasks with different thermal characteristics, e.g. *hot* and *cool* tasks, the entire temperature curve can be maintained under a predefined threshold without introducing any cooling period. Based on these observations, we formally establish and successfully prove two important theorems, which are used to guide the development of our scheduling algorithms, i.e. one for processors without DVS capability and the other one for processors with DVS feature. Our experimental results, based on parameters drawn from the 65nm technology, show that on average our methods outperform the existing approaches [119] by over 23.3% and 5.3%, for a processor without and with DVS capability, respectively. Moreover, under different ambient temperature conditions, the proposed method can guarantee that all task can be completed without a single occurrence of peak temperature constraint violation. It is a 21% feasibility improvement over the existing approach [119].

To summarize, in this research, the proposed system-level leakage model and the energy estimation method are reasonably accurate while achieving significant computational overhead reduction. Moreover, the performance of the proposed scheduling techniques consistently outperform a number of recently published existing works.

Furthermore, the theoretical study in this dissertation lays a solid foundation for the guidance of the power/thermal-aware scheduling algorithms development in practical single and multi-processor platforms.

7.2 Future Work

Our research in this dissertation focuses on a single-core-processor platform. However, considering the fact that multi-processor architectures are becoming more popular, especially when the three-dimensional integrated circuit (3D-IC) emerged as a novel technology [40, 62, 43, 63], it is worth to extend our existing work into the 3D multi-processor arena.

Advantages of a 3D multi-processor Recently, 3D-IC [71, 121, 75, 122, 104, 35, 105] is emerging as a novel technology that vertically integrates multiple dies with through-silicon vias (TSVs). The benefits of 3D-IC come from several aspects. 1) Due to the significantly reduced interconnect length, higher performance can be achieved with less delay and lower power consumption. 2) The device density is increased with a smaller footprint/form factor which may also help to improve the manufacturing yield and devise more cost effective packaging methods. 3) 3D-IC enables heterogeneous integration which allows us to integrate components processed by different technologies into a single system, i.e. integrate a RF die manufactured by low cost 130nm technology with a fast but expensive 45nm logic chip. Considering these advantages, how to effectively implement 3D-IC is becoming a research hot spot for both industry and academia.

Challenges in designing a 3D multi-processor As an immediate consequence of vertically stacking multiple silicon dies, the power density increases by a factor of

the number of layers, e.g. the heat fluxes up to $250W/cm^2$[103]. As we mentioned before, the high power consumption directly translates into high temperature, which has many negative effects on a system, e.g. high packaging and cooling cost, reduced performance, system life span and reliability. In 3D-IC, the temperature issue can only become worse. It is reported in [74] that for a commercial 65nm dual-core processor, the peak temperature of a two-die 3D design can be 18 ^oC higher than its 2D counterpart. Besides the high power density, the heterogeneity of heat-removing efficiency at different layers and locations on a 3D-IC also makes the problem complicated. In fact, the layer that is close to the heat sink can remove heat off the chip more efficiently. Moreover, the unbalanced heat-transfer efficiency also causes the temperature variation across the chip, which in turn results in thermal stress, hot spot and reliability degradation.

Evidently, power and thermal issue is one of the major hurdles of implementing 3D-ICs. Therefore, power and thermal-aware design techniques are needed. The OS level scheduling method, given the fact that it knows the characteristics of both hardware and software, is one of the most effective methods to alleviate the power and thermal-stress and is worth to be further explored. Despite that there are various techniques proposed for 2D processors to address the power and thermal issue, e.g. [52, 27, 119, 38], the heterogeneous cooling efficiency and the heat transfer in the additional dimension have simply made the existing thermal management techniques less effective.

Extending our research into 3D To extend our existing work on single-core platform, the first and most important step is to develop appropriate system models, especially the thermal and power models, to capture the three dimensional heat flow and the crucial leakage/temperature dependency. More importantly, the model is

required to be simple enough for system-level analysis.

One straightforward way to derive a 3D thermal model is to use our existing results on single-core platforms. Specifically, we use a pair of RC to model the temperature dynamic of each individual core and then add the lateral and vertical thermal resistance between processor cores to capture the horizontal and vertical heat transfer. By doing so, a single linear differential equation (required to describe the thermal dynamic of a single-core platform) becomes a group of linear differential equations. The computational overhead of using this model can be prohibitively expensive when the number of cores we are trying to model is large, not to mention analytically formulating the temperature change or establishing some theorems to guide the scheduling algorithm development. Apparently, additional effort must be and worth to be spent to find efficient ways to model the power consumption and the thermal dynamic of a 3D multi-processor.

This dissertation has built a theoretical foundation to study power/thermal-aware scheduling problem for single-processor platform. Although 3D multi-processor scheduling is a much more complicated task than its single-core counterpart, we still feel there are ways to extend our existing work into the 3D arena. For example, the *M-Oscillating* in time horizon may be extended to both temporal and spatial oscillation.

REFERENCES

[1] Engadget. page http://www.engadget.com.

[2] L'hopital's rule, http://en.wikipedia.org/wiki/l'h

[3] Mediabench, http://euler.slu.edu/ fritts/mediabench/.

[4] Moore's law. pages http://en.wikipedia.org/wiki/Moore's–law.

[5] Round-robin, http://en.wikipedia.org/wiki/round-robin.

[6] Simplescalar, http://www.simplescalar.com.

[7] Spec2000 benchmarks, http://www.spec.org.

[8] Hotspot 4.2 temperature modeling tool. *University of Virgina*, page http://lava.cs.virginia.edu/HotSpot, 2009.

[9] Deloitte TMT predictions. page http://www.deloitte.com, 2011.

[10] A. Abdollahi, F. Fallah, and Massoud. Runtime mechanisms for leakage current reduction in cmos vlsi circuits. In *Low Power Electronics and Design, 2002. ISLPED '02. Proceedings of the 2002 International Symposium on*, pages 213 – 218, 2002.

[11] E. P. Agency. Report to congress on server and data center energy efficiency. page http://www.energystart.gov, 2006.

[12] AMD. Power and cooling in the data center. page http://enterprise.amd.com, 2007.

[13] F. Assaderaghi, D. Sinitsky, S. A. Parke, J. Bokor, P. Ko, and C. Hu. Dynamic threshold-voltage mosfet (dtmos) for ultra-low voltage vlsi. *IEEE Trans. on Elec. Dev.*, 44(3):414–422, Mar 1997.

[14] N. Bansal, T. Kimbrel, and K. Pruhs. Speed scaling to manage energy and temperature. *Journal of the ACM*, 54(1):1–39, 2007.

[15] M. Bao, A. Andrei, P. Eles, and Z. Peng. On-line thermal aware dynamic voltage scaling for energy optimization with frequency/temperature dependency consideration. In *Design Automation Conference*, pages 490–495, 2009.

[16] M. Bao, A. Andrei, P. Eles, and Z. Peng. Temperature-aware idle time distribution for energy optimization with dynamic voltage scaling. In *DATE*, pages 21 – 27, 2010.

[17] C. Belady. In the data center, power and cooling costs more than the it equipment it supports. *Electronics Cooling*, 23(1), 2007.

[18] U. Berkley. Bsim3/bsim4. *Device Research Group, UC Berkeley*, 2008.

[19] S. Borkar. Design challenges of technology scaling. *Micro, IEEE*, 19(4):23 –29, jul-aug 1999.

[20] S. Borkar. Thousand core chips: a technology perspective. In *DAC*, pages 746–749, 2007.

[21] K. G. Brill. The invisible crisis in the data center: The economic meltdown of moores law. *Uptime Institute*, 2007.

[22] D. Brooks and M. Martonosi. Dynamic thermal management for high-performance microprocessors. In *HPCA '01: Proceedings of the 7th International Symposium on High-Performance Computer Architecture*, page 171, 2001.

[23] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: a framework for architectural-level power analysis and optimizations. In *ISCA*, pages 83–94, 2000.

[24] J. Cadenas, R. Sherratt, P. Huerta, and W.-C. Kao. Parallel pipelined array architectures for real-time histogram computation in consumer devices. *Consumer Electronics, IEEE Transactions on*, 57(4):1460 –1464, november 2011.

[25] B. H. Calhoun, F. A. Honore, and A. Chandrakasan. Design methodology for fine-grained leakage control in mtcmos. *ISLPED*, pages 104–109, 2003.

[26] T. Chantem, R. P. Dick, and X. S. Hu. Temperature-aware scheduling and assignment for hard real-time applications on mpsocs. In *DATE*, pages 288–293, 2008.

[27] T. Chantem, X. S. Hu, and R. Dick. Online work maximization under a peak temperature constraint. In *ISLPED*, pages 105–110, 2009.

[28] V. Chaturvedi, H. Huang, and G. Quan. Leakage aware scheduling on maximal temperature minimization for periodic hard real-time systems. In *ICESS*, pages 1802–1809.

[29] J. Chen, C. Hung, and T. Kuo. On the minimization fo the instantaneous temperature for periodic real-time tasks. *RTAS*, pages 236–248, 2007.

[30] J.-J. Chen, H.-R. Hsu, and T.-W. Kuo. Leakage-aware energy-efficient scheduling of real-time tasks in multiprocessor systems. In *RTAS*, pages 408–417, 2006.

[31] J.-J. Chen, S. Wang, and L. Thiele. Proactive speed scheduling for real-time tasks under thermal constraints. *RTAS*, 0:141–150, 2009.

[32] M.-S. Chen, K. Shin, and D. Kandlur. Addressing, routing, and broadcasting in hexagonal mesh multiprocessors. *Computers, IEEE Transactions on*, 39(1):10 –18, jan 1990.

[33] J.-Y. Chung, J. Liu, and K.-J. Lin. Scheduling periodic jobs that allow imprecise results. *Computers, IEEE Transactions on*, 39(9):1156 –1174, sep 1990.

[34] A. Cohen, F. Finkelstein, A. Mendelson, R. Ronen, and D. Rudoy. On estimating optimal performance of cpu dynamic thermal management. *IEEE Computer Architecture Letter*, 2(1):6–9, 2003.

[35] A. Coskun, J. Ayala, D. Atienza, T. Rosing, and Y. Leblebici. Dynamic thermal management in 3d multicore architectures. In *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.*, pages 1410 –1415, april 2009.

[36] T. Cucinotta, A. Mancina, G. Anastasi, G. Lipari, L. Mangeruca, R. Checcozzo, and F. Rusina. A real-time service-oriented architecture for industrial automation. *Industrial Informatics, IEEE Transactions on*, 5(3):267 –277, aug. 2009.

[37] S. Dash and T. Srikanthan. Modeling rtos components for instruction cache hit rate estimation. In *Circuits and Systems, 2009. ISCAS 2009. IEEE International Symposium on*, pages 2978 –2981, may 2009.

[38] J. Donald and M. Martonosi. Techniques for multicore thermal management: Classification and new exploration. *SIGARCH Comput. Archit. News*, 34(2):78–88, 2006.

[39] S. Duarte, Y. Tsai, N. Vijaykrishnan, and M. Irwin. Evaluating run-time techniques for leakage power reduction. *VLSID'02*, 2002.

[40] P. Falkenstern, Y. Xie, Y.-W. Chang, and Y. Wang. Three-dimensional integrated circuits (3d ic) floorplan and power/ground network co-synthesis. In *Design Automation Conference (ASP-DAC), 2010 15th Asia and South Pacific*, pages 169 –174, jan. 2010.

[41] S. Fischmeister, O. Sokolsky, and I. Lee. A verifiable language for programming real-time communication schedules. *Computers, IEEE Transactions on*, 56(11):1505 –1519, nov. 2007.

[42] N. Fisher, J.-J. Chen, S. Wang, and L. Thiele. Thermal-aware global real-time scheduling on multicore systems. *RTAS*, 0:131–140, 2009.

[43] S. Fujita, K. Abe, K. Nomura, S. Yasuda, and T. Tanamoto. Perspectives and issues in 3d-ic from designers' point of view. In *Circuits and Systems, 2009. ISCAS 2009. IEEE International Symposium on*, pages 73 –76, may 2009.

[44] F. Gao and J. P. Hayes. Exact and heuristic approaches to input vector control for leakage power reduction. In *ICCAD '04: Proceedings of the 2004 IEEE/ACM International conference on Computer-aided design*, pages 527–532, 2004.

[45] A. Gerstlauer, H. Yu, and D. Gajski. Rtos modeling for system level design. In *Design, Automation and Test in Europe Conference and Exhibition, 2003*, pages 130 – 135, 2003.

[46] S. Gunther, F. Binns, D. M. Carmean, and J. C. Hall. Managing the impact of increasing microprocessor power consumption. *Intel Technology Journal*, 5(1), 2001.

[47] V. Hanumaiah, R. Rao, S. Vrudhula, and K. S. Chatha. Throughput optimal task allocation under thermal constraints for multi-core processors. In *Proceedings of the 46th Annual Design Automation Conference*, DAC '09, pages 776–781, New York, NY, USA, 2009. ACM.

[48] V. Hanumaiah, S. Vrudhula, and K. Chatha. Maximizing performance of thermally constrained multi-core processors by dynamic voltage and frequency control. In *Computer-Aided Design - Digest of Technical Papers, 2009. ICCAD 2009. IEEE/ACM International Conference on*, pages 310 –313, nov. 2009.

[49] H. Huang and J. Fan. Multicore processor cluster based sleep transistor sizing considering delay profile. In *IEEE 8th International Conference on ASIC (ASICON)*, 2009.

[50] H. Huang and G. Quan. Leakage aware energy minimization for real-time systems under the maximum temperature constraint. In *DATE*, 2011.

[51] H. Huang, G. Quan, and J. Fan. Leakage temperature dependency modeling in system level analysis. In *ISQED*, pages 447–452, 2010.

[52] H. Huang, G. Quan, J. Fan, and M. Qiu. Throughput maximization for periodic real-time systems under the maximal temperature constraint. In *DAC*, pages 363–368, 2011.

[53] W. Huang, M. R. Stan, K. Skadron, K. Sankaranarayanan, S. Ghosh, and S. Velusam. Compact thermal modeling for temperature-aware design. In *DAC*, pages 878–883, 2004.

[54] Y. Hwang, G. Schirner, S. Abdi, and D. Gajski. Accurate timed rtos model for transaction level modeling. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, pages 1333 –1336, march 2010.

[55] ITRS. *International Technology Roadmap for Semiconductors*. International SEMATECH, Austin, TX., http://public.itrs.net/.

[56] R. Jayaseelan and T. Mitra. Temperature aware task sequencing and voltage scaling. In *ICCAD*, pages 618 – 623, 2008.

[57] R. Jejurikar, C. Pereira, and R. Gupta. Dynamic slack reclamation with procrastination scheduling in real-time embedded systems. *DAC*, pages 111 – 116, 2005.

[58] S.-O. Jung, K.-W. Kim, and S.-M. Kang. Noise constrained transistor sizing and power optimization for dual v/sub t/ domino logic. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 10(5):532 –541, oct. 2002.

[59] C. Kim and K. Roy. Dynamic vth scaling scheme for active leakage power reduction. *DATE*, pages 163–167, 2002.

[60] P. Kumar and L. Thiele. Cool shapers: Shaping real-time tasks for improved thermal guarantees. In *DAC*, pages 468–473, 2011.

[61] P. Kumar and L. Thiele. Thermally optimal stop-go scheduling of task graphs with real-time constraints. In *ASP-DAC*, pages 123–128, 2011.

[62] J. Lau. Evolution, challenge, and outlook of tsv, 3d ic integration and 3d silicon integration. In *Advanced Packaging Materials (APM), 2011 International Symposium on*, pages 462 –488, oct. 2011.

[63] J. H. Lau. Tsv manufacturing yield and hidden costs for 3d ic integration. In *Electronic Components and Technology Conference (ECTC), 2010 Proceedings 60th*, pages 1031 –1042, june 2010.

[64] C.-H. Lee and K. Shin. On-line dynamic voltage scaling for hard real-time systems using the edf algorithm. In *Real-Time Systems Symposium, 2004. Proceedings. 25th IEEE International*, pages 319 – 335, 2004.

[65] W. Liao, L. He, and K. Lepak. Temperature and supply voltage aware performance and power modeling at microarchitecture level. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(7):1042 – 1053, 2005.

[66] W. Liao, L. He, and K. Lepak. Temperature and supply voltage aware performance and power modeling at microarchitecture level. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(7):1042 – 1053, 2005.

[67] C. H. Lim, W. Daasch, and G. Cai. A thermal-aware superscalar microprocessor. In *Quality Electronic Design, 2002. Proceedings. International Symposium on*, pages 517 – 522, 2002.

[68] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20:46–61, January 1973.

[69] S. Liu and M. Qiu. Thermal aware scheduling for peak temperature reduction with stochastic workloads. *RTAS(WiP)*, 0:53–56, 2010.

[70] S. Liu, M. Qiu, W. Gao, X.-J. Tang, and B. Guo. Hybrid of job sequencing and DVFS for peak temperature reduction with nondeterministic applications. In *ICESS*, pages 1780–1787, 2010.

[71] S. Liu, J. Zhang, Q. Wu, and Q. Qiu. Thermal-aware job allocation and scheduling for three dimensional chip multiprocessor. In *Quality Electronic Design (ISQED), 2010 11th International Symposium on*, pages 390 –398, march 2010.

[72] Y. Liu, R. P. Dick, L. Shang, and H. Yang. Accurate temperature-dependent integrated circuit leakage power estimation is easy. In *DATE*, pages 1526–1531, 2007.

[73] Y. Liu, H. Yang, R. P. Dick, H. Wang, and L. Shang. Thermal vs energy optimization for dvfs-enabled processors in embedded systems. In *ISQED*, pages 204–209, 2007.

[74] G. H. Loh, Y. Xie, and B. Black. Processor design in 3d die-stacking technologies. *Micro, IEEE*, 27(3):31 –48, may-june 2007.

[75] C.-L. Lung, Y.-L. Ho, D.-M. Kwai, and S.-C. Chang. Thermal-aware on-line task allocation for 3d multi-core processor throughput optimization. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2011*, pages 1 –6, march 2011.

[76] A. Mahajan and D. Teneketzis. Optimal design of sequential real-time communication systems. *Information Theory, IEEE Transactions on*, 55(11):5317 –5338, nov. 2009.

[77] H. Mahmoodi, V. Tirumalashetty, M. Cooke, and K. Roy. Ultra low-power clocking scheme using energy recovery and clock gating. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 17(1):33 –44, jan. 2009.

[78] J. Markoff. Intel's big shift after hitting technical wall. *New York Times*, 2004.

[79] H. Mitra and P. Ramanathan. A genetic approach for scheduling non-preemptive tasks with precedence and deadline constraints. In *System Sciences, 1993, Proceeding of the Twenty-Sixth Hawaii International Conference on*, volume ii, pages 556 –564 vol.2, jan 1993.

[80] K. Mohanram and N. Touba. Lowering power consumption in concurrent checkers via input ordering. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 12(11):1234 –1243, nov. 2004.

[81] J. Moore, J. Chase, P. Ranganathan, and R. Sharma. Making scheduling "cool": Temperature-aware resource assignment in data centers. *Usenix Annual Technical Conference*, 2005.

[82] R. Moraes, F. Vasques, P. Portugal, and J. Fonseca. Vtp-csma: A virtual token passing approach for real-time communication in ieee 802.11 wireless networks. *Industrial Informatics, IEEE Transactions on*, 3(3):215 –224, aug. 2007.

[83] S. Murali, A. Mutapcic, D. Atienza, R. Gupta, S. Boyd, and G. D. Micheli. Temperature-aware processor frequency assignment for mpsocs using convex optimization. In *CODES+ISSS*, pages 111–116, 2007.

[84] M. Pedram and J. M. Rabaey. *Power Aware Design Methodolodies*. Kluwer Academic Publishers, 2002.

[85] D.-T. Peng and K. Shin. Static allocation of periodic tasks with precedence constraints in distributed real-time systems. In *Distributed Computing Systems, 1989., 9th International Conference on*, pages 190 –198, jun 1989.

[86] R. W. Pike. *Optimization for Engineering Systems*. Van Nostrand Reinhold Company, 2001.

[87] R. Prasher, J. Chang, I. Sauciuc, S. Narasimhan, D. chau, G. Chrysler, A. Myers, S. Prstic, and C. Hu. Nano and micro technology-based next-generation package-level cooling solutions. *Intel Technology Journal*, 9(4), 2005.

[88] G. Quan and V. Chaturvedi. Feasibility analysis for temperature-constraint hard real-time periodic tasks. *IEEE Transaction on Industrial Informatics*, 6(3):329–339, 2010.

[89] G. Quan, L. Niu, B. Mochocki, and X. S. Hu. Fixed-priority scheduling for reducing both the dynamic and leakage energy on variable voltage processors. *International Journal of Embedded Systems*, 2008.

[90] G. Quan and Y. Zhang. Leakage aware feasibility analysis for temperature-constrained hard real-time periodic tasks. *ECRTS*, pages 207–216, 2009.

[91] G. Quan, Y. Zhang, W. Wiles, and P. Pei. Guaranteed scheduling for repetitive hard real-time tasks under the maximal temperature constraint. *ISSS+CODES*, 2008.

[92] J. Rabaey, A. Chandrakasan, and B. Nikolic. *Digital Integrated Circuits: A Design Perspective*. Prentice Hall, 2003.

[93] K. Ramamritham, J. Stankovic, and W. Zhao. Distributed scheduling of tasks with deadlines and resource requirements. *Computers, IEEE Transactions on*, 38(8):1110 –1123, aug 1989.

[94] R. Rao, S. Vrudhula, C. Chakrabarti, and N. Chang. An optimal analytical solution for processor speed control with thermal constraints. In *ISLPED*, pages 292–297, 2006.

[95] K. Roy, S. Mukhopadhyay, and H. Mahmoodi-meimand. Leakage current mechanisms and leakage reduction techniques in deep-submicometer cmos circuits. *Proceedings of the IEEE*, 19(2):305–327, February 2003.

[96] M. Santarini. Thermal integrity: A must for low-power ic digital design. *EDN*, pages 37–42, 2005.

[97] S. Schliecker, J. Rox, M. Negrean, K. Richter, M. Jersak, and R. Ernst. System level performance analysis for real-time automotive multicore and network architectures. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 28(7):979 –992, july 2009.

[98] K. Shin and P. Ramanathan. Real-time computing: a new discipline of computer science and engineering. *Proceedings of the IEEE*, 82(1):6 –24, jan 1994.

[99] S. Sirichotiyakul, T. Edwards, C. Oh, R. Panda, and D. Blaauw. Duet: An accurate leakage estimation and optimization tool for dual-vt circuits. *IEEE Trans. on VLSI*, 10(2):79–90, April 2002.

[100] K. Skadron, M. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan. Temperature-aware computer systems: opportunities and challenges. *IEEE Micro*, 23(6):52–61, 2003.

[101] K. Skadron, M. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan. Temperature-aware microarchitecture. *ICSA*, pages 2–13, 2003.

[102] K. Skadron, M. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan. Temperature-aware microarchitecture. *ICSA*, pages 2–13, 2003.

[103] A. Sridhar, A. Vincenzi, M. Ruggiero, T. Brunschwiler, and D. Atienza. 3d-ice: Fast compact transient thermal modeling for 3d ics with inter-tier liquid cooling. In *Computer-Aided Design (ICCAD), 2010 IEEE/ACM International Conference on*, pages 463 –470, nov. 2010.

[104] C. Sun, L. Shang, and R. Dick. Three-dimensional multiprocessor system-on-chip thermal optimization. In *Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2007 5th IEEE/ACM/IFIP International Conference on*, pages 117 –122, 30 2007-oct. 3 2007.

[105] K. Torki. Design platform and tools for 3d ic integration. In *3D Systems Integration Conference (3DIC), 2010 IEEE International*, pages 1 –25, nov. 2010.

[106] T.-H. Tsai and Y.-N. Pan. High efficiency architecture design of real-time qfhd for h.264/avc fast block motion estimation. *Circuits and Systems for Video Technology, IEEE Transactions on*, 21(11):1646 –1658, nov. 2011.

[107] V. Venkatachalam and M. Franz. Power reduction techniques for microprocessor systems. *ACM Computing Surveys*, 37(3):195–237, Sep 2005.

[108] S. Wang and R. Bettati. Reactive speed control in temperature-constrained real-time systems. *ECRTS*, pages 161–170, 2006.

[109] P. Weiler, R. Kopp, and R. Dorman. A real-time operating system for manned spaceflight. *Computers, IEEE Transactions on*, C-19(5):388 – 398, may 1970.

[110] C.-Y. Yang, J.-J. Chen, L. Thiele, and T.-W. Kuo. Energy-efficient real-time task scheduling with temperature-dependent leakage. In *DATE*, pages 9–14, 2010.

[111] J. Yang, X. Zhou, M. Chrobak, Y. Zhang, and L. Jin. Dynamic thermal management through task scheduling. In *International Symposium on Performance Analysis of Systems and Software*, pages 191–201, 2008.

[112] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced CPU energy. In *FOCS*, pages 374–382, 1995.

[113] L.-T. Yeh and R. C. Chu. *Thermal Management of Microelectronic Equipment: Heat Transfer Theory, Analysis Methods, and Design Practices*. ASME Press, New York, NY, 2002.

[114] I. Yeo, C. C. Liu, and E. J. Kim. Predictive dynamic thermal management for multicore systems. In *DAC*, pages 734–739, 2008.

[115] H. Yu, B. Veeravalli, and Y. Ha. Leakage-aware dynamic scheduling for real-time adaptive applications on multiprocessor systems. In *DAC*, pages 493–498, 2010.

[116] L. Yuan, S. Leventhal, and G. Qu. Temperature-aware leakage minimization technique for real-time systems. In *ICCAD*, pages 761–764, 2006.

[117] L. Yuan and G. Qu. Alt-dvs: Dynamic voltage scaling with awareness of leakage and temperature for real-time systems. *Adaptive Hardware and Systems, NASA/ESA Conference on*, 0:660–670, 2007.

[118] S. Zhang and K. S. Chatha. Approximation algorithm for the temperature-aware scheduling problem. In *ICCAD*, pages 281–288, 2007.

[119] S. Zhang and K. S. Chatha. Thermal aware task sequencing on embedded processors. In *DAC*, pages 585 – 590, 2010.

[120] Y. Zhang, D. Parikh, K. Sankaranarayanan, K. Skadron, and M. Stan. Hotleakage: a temperature-aware model of subthreshold and gate leakage for architects. *University of Virginia Dept. of Computer Science Technical Report*, 2003.

[121] X. Zhou, Y. Xu, Y. Du, Y. Zhang, and J. Yang. Thermal management for 3d processors via task scheduling. In *Parallel Processing, 2008. ICPP '08. 37th International Conference on*, pages 115 –122, sept. 2008.

[122] C. Zhu, Z. Gu, L. Shang, R. Dick, and R. Joseph. Three-dimensional chip-multiprocessor run-time thermal management. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 27(8):1479 –1492, aug. 2008.

[123] X. Zhu, S. Han, P.-C. Huang, A. Mok, and D. Chen. Mbstar: A real-time communication protocol for wireless body area networks. In *Real-Time Systems (ECRTS), 2011 23rd Euromicro Conference on*, pages 57 –66, july 2011.

VITA

HUANG HUANG

EDUCATION

Ph.D. Candidate in Electrical Engineering — 08/2007-Present
Florida International University, Miami, FL
Advisor: Dr. Gang Quan

Bachelor of Science in Electrical Engineering — 08/2003-07/2007
Northwest University, Xi'an, China

TEACHING EXPERIENCES

Teaching Assistant — Spring 2011
EEE3396 Solid State Devices

Instructor — Fall 2010
EEL3712L Logic Design Lab

Lecturer — Summer 2009, Fall2009
EEL3003 Electrical Engineering I

Instructor — Fall 2007, Spring 2008, Spring 2009
EEL3111L Circuit I Lab

PUBLICATIONS

Journals

J5. H. Huang M. Fan, G. Quan, "Leakage and Temperature Aware Energy Minimization Scheduling for Hard Real-Time Systems", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. (under review)

J4. H. Huang, V. Chatervedi, G. Quan, J. Fan, "Throughput Maximization for Periodic Real-Time Systems under the Maximal Temperature Constraint", IEEE Transaction on Very Large Scaled Integrated Circuit. (under review)

J3. H. Huang, V. Chaturvedi, G. Quan, "Leakage Aware Scheduling On Maximum Temperature Minimization For Periodic Hard Real-Time Systems", Journal of Low Power Electronics. (under review)

J2. V. Chaturvedi, H. Huang G. Qua, "On the Fundamentals of Leakage Aware Real-Time DVS Scheduling for Peak Temperature Minimization " , IEEE Transaction on Industrial Informatics. (under review)

J1. A. Mullaguru, H. Huang, X. Yuan, J. Fan, "Model Order Reduction via Eigen Decomposition Analysis", International Journal of Modeling, Identification and Control (IJMIC), 2009.

Refereed Conferences

C7. H. Huang, M. Fan, G. Quan, "On-Line Leakage-Aware Energy Minimization Scheduling for Hard Real-Time Systems", 17th IEEE Asia and South Pacific Design Automation Conference (ASP-DAC), pp.677-682, 30 January - 2 February, 2012

C6. H. Huang, G. Quan, J. Fan, M. Qiu, "Throughput Maximization for Periodic Real-Time Systems under the Maximal Temperature Constraint", Design Automation Conference (DAC), 2011 48th ACM/EDAC/IEEE, pp.363-368, 5-9 June, 2011

C5. H. Huang and G. Quan, "Leakage Aware Energy Minimization for Real-Time System under the Maximal Temperature Constraints", Design, Automation & Test in Europe Conference & Exhibition (DATE), pp.1-6, 14-18 March, 2011

C4. V. Chaturvedi, H. Huang, G. Quan, "Leakage Aware Scheduling On Maximal Temperature Minimization For Periodic Hard Real-Time Systems", Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on, pp.1802-1809, June 29 2010-July 1, 2010

C3. H. Huang, G. Quan, J. Fan, "Leakage Temperature Dependency Modeling in System Level Analysis", 2010 11th International Symposium on Quality Electronic Design (ISQED), pp.447-452, 22-24 March, 2010

C2. H. Huang, J. Fan, "Multi-core Processor Cluster Based Sleep Transistor Sizing Considering Delay Profile", IEEE 8th International Conference on ASIC (ASICON), Changsha, China, October 20-23, 2009.

C1. C. Liu, R. Chen, H. Huang, J. Fan, "Parameterized Interconnect Modeling and Simulation in VLSI Design Considering Variations", IEEE 40th Southeastern Symposium on System Theory (SSST), pp. 225-229, March 17-18, 2008.