REAL-TIME TASK SCHEDULING UNDER THERMAL CONSTRAINTS

A Dissertation

by

YOUNGWOO AHN

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

August 2010

Major Subject: Computer Engineering

REAL-TIME TASK SCHEDULING UNDER THERMAL CONSTRAINTS

A Dissertation

by

YOUNGWOO AHN

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Approved by:

Co-Chairs of Committee,	Riccardo Bettati
	Narasimha Reddy
Committee Members,	Weiping Shi
	Deepa Kundur
Head of Department,	Costas Georghiades

August 2010

Major Subject: Computer Engineering

ABSTRACT

Real-Time Task Scheduling under Thermal Constraints. (August 2010)

Youngwoo Ahn, B.S.; M.S., Seoul National University

Co–Chairs of Advisory Committee: Dr. Riccardo Bettati
Dr. Narasimha Reddy

As the speed of integrated circuits increases, so does their power consumption. Most of this power is turned into heat, which must be dissipated effectively in order for the circuit to avoid thermal damage. Thermal control therefore has emerged as an important issue in design and management of circuits and systems. Dynamic speed scaling, where the input power is temporarily reduced by appropriately slowing down the circuit, is one of the major techniques to manage power so as to maintain safe temperature levels.

In this study, we focus on thermally-constrained hard real-time systems, where timing guarantees must be met without exceeding safe temperature levels within the microprocessor. Speed scaling mechanisms provided in many of today's processors provide opportunities to temporarily increase the processor speed beyond levels that would be safe over extended time periods. This dissertation addresses the problem of safely controlling the processor speed when scheduling mixed workloads with both hard-real-time periodic tasks and non-real-time, but latency-sensitive, aperiodic jobs.

We first introduce the Transient Overclocking Server, which safely reduces the response time of aperiodic jobs in the presence of hard real-time periodic tasks and thermal constraints. We then propose a design-time (off-line) execution-budget allocation scheme for the application of the Transient Overclocking Server. We show that there is an optimal budget allocation which depends on the temporal characteristics of the aperiodic workload. In order to provide a quantitative framework for the allocation of budget during system design, we present a queuing model and validate the model with results from a discrete-event simulator.

Next, we describe an on-line thermally-aware transient overclocking method to reduce the response time of aperiodic jobs efficiently at run-time. We describe a modified Slack-Stealing algorithm to consider the thermal constraints of systems together with the deadline constraints of periodic tasks. With the thermal model and temperature data provided by embedded thermal sensors, we compute slack for aperiodic workload at run-time that satisfies both thermal and temporal constraints. We show that the proposed Thermally-Aware Slack-Stealing algorithm minimizes the response times of aperiodic jobs while guaranteeing both the thermal safety of the system and the schedulability of the real-time tasks. The two proposed speed control algorithms are examples of so-called proactive schemes, since they rely on a prediction of the thermal trajectory to control the temperature before safe levels are exceeded.

In practice, the effectiveness of proactive speed control for the thermal management of a system relies on the accuracy of the thermal model that underlies the prediction of the effects of speed scaling and task execution on the temperature of the processor. Due to variances in the manufacturing of the circuit and of the environment it is to operate, an accurate thermal model can be gathered at deployment time only. The absence of power data makes a straightforward derivation of a model impossible.

We, therefore, study and describe a methodology to infer efficiently the thermal model based on the monitoring of system temperatures and number of instructions used for task executions.

To my family

ACKNOWLEDGMENTS

I would like to thank my research advisor, Dr. Riccardo Bettati, for his guidance and support during all these years. I am greatly indebted to him for his constant inspiration, encouragement, and patience throughout my doctoral study. His exceptional commitment to research and strong demand for excellence have guided me this far. And his valuable feedback contributed greatly to the improvement and progress of my research and dissertation.

I thank the rest of my dissertation committee members: Dr. Narasimha Reddy, Dr. Weiping Shi, and Dr. Deepa Kundur. I also thank Dr. Paul Gratz, who substitutes for Dr. Weiping Shi at my defense. Their insightful comments and constructive criticism helped me to improve the dissertation in many ways.

I would like to thank Dr. Lauren Cifuentes for her support with my graduate assistantship, and Mr. Willis Marti for his kind guidance in the VTECH project. Many thanks also go to my fellow workers for the project. They are: Mr. Omar Alvarez, Ms. Rene Mercer, Mr. Hutson Bets, Mr. Garg Kapil, Mr. Gaurav Yadav, and Ms. Jillian Michelle.

I would like to express my appreciation to Dr. Inchoon Yeo. His constant eagerness for research always stimulated me to work harder and the discussion with him improved my research as well. In addition, I thank my friends and fellow students at Texas A&M University for numerous discussions related to my research work. I sincerely thank all the current and previous members in Real-Time Systems Group: Dr. Shenquan Wang, Dr. Pubali Banerjee, Mr. Omar Alvarez, Mr. Ja-Ryeong Koo, Mr. Chien An Lai, Mr. Tuneesh Lella, Mr. Saswat Mohanty, and Mr. Animesh Pathak. Special thanks to Dr. Shenquan Wang for his comments and guidance in this research field.

Last but not least, I thank my parents and my family members for their continuous support and encouragement. I am especially grateful to my wife for her endless support and love. Without their tremendous sacrifices and great love, I would not have chance for successfully accomplishing my Ph.D.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

FIGURE Page

CHAPTER I

INTRODUCTION

A. Motivation

In integrated circuits, such as microprocessors and graphic chips, the power consumption is related to the speed of the circuit. In fact, the power density, that is the amount of power per area that needs to be dissipated in microprocessors has increased dramatically in recent years, since circuits at the same time increase in speed and decrease in size into sub-micron regions [2]. Because the power consumption by the microprocessors is converted into heat, resulting in significant increase of temperature and unreliability of systems, appropriate mechanisms for power management and power dissipation are becoming increasingly important.

1. Increasing Power Density of Microprocessors

The power density of modern microprocessors has been increasing due to both increase of total power dissipation and decrease of chip feature size.

For years, there has been significant improvement in many aspects of chip technologies (e.g. performance, size, input voltage, and etc.). In the effort to improve the performance of microprocessors with higher clock frequency or to reduce the chip feature size, we have hardly achieved an improvement in the power density until recently in spite of the reduced supply voltage to circuits.

Fig. 1 displays the increasing trend of Intel's microprocessor family [1]: The power density (i.e. power dissipation per chip area) of the latest generation of single core processor (Pentium 4 and Itanium) is very close to the level of a nuclear reactor.

The journal model is *IEEE Transactions on Automatic Control*.

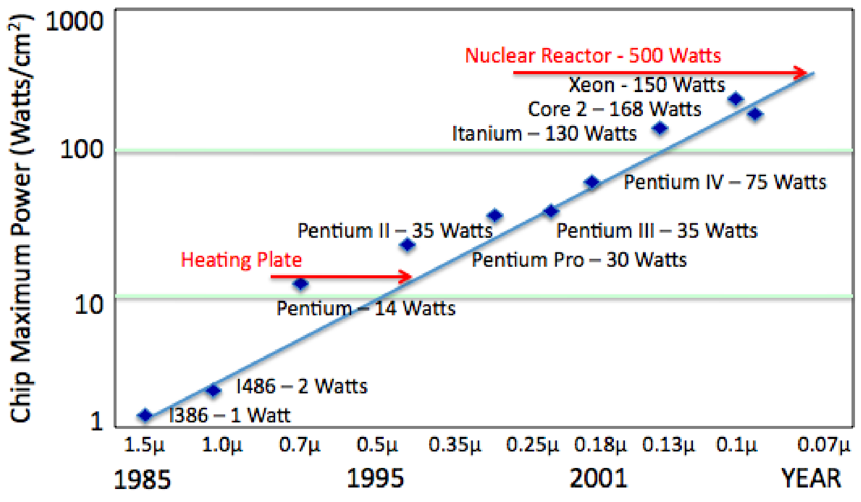


Fig. 1. Power density of Intel microprocessors [1]

Although Chip-Multi-Processor (CMP) CPUs and some mobile processors mitigate the increase of power density by running at lower clock frequencies, the trend towards increasing power density is expected to continue along with the request for higher performance and more aggressive computing technologies such as 3D chips [3]. With this rapidly increasing power density, the problem of thermal management in systems is becoming acute. Methods to manage heat to control its dissipation have been gaining much attention by researchers and practitioners and a number of thermal control mechanisms are routinely being used. For example, when the temperature of Pentium 4 CPUs exceed the configured threshold temperature, the *clock throttling* mechanism which is a popular thermal management technique first kicks in to lower the thermal level. If the temperature keeps increasing to a critical level due to any possible failure of cooling mechanism, the processor will automatically shut down as a safety feature. To maintain the Pentium 4 CPU temperature low while it continues executions, very powerful CPU fans are required, thereby increasing the cooling costs and the price of systems. Power and temperature now have become the primary constraints for modern processors.

2. Thermal Impact on Microprocessors

High temperatures have a significant impact on microprocessors in terms of performance, power consumption, and reliability. Since the mobility of electrons in semiconductor degrades under high thermal levels, transistors work poorer than at low temperature. And the metal resistivity of interconnections inside chips at high temperatures increases so that it results in longer interconnection delay. These two can be the main reasons for the performance degradation of microprocessors at high temperatures. Furthermore, the leakage power of microprocessors grows with increasing temperature, which in turn causes the power consumption to grow as well. It is also known that a higher operational temperature reduces reliability and decreases the life time of the microprocessors.

Thus, techniques are being investigated for thermal control both at design time, through appropriate packaging and active heat dissipation mechanisms, and at run time through various forms of Dynamic Thermal Management [4].

3. Thermal Management of Microprocessors

The design time packaging solutions (e.g. heat sinks, cooling fans, and etc.) can be very expensive. For example, Tiwari et al. in [5] show how the incremental packaging cost per additional Watt becomes very high for processors above 35-40W power dissipation. It is also predicted that the cooling solutions through packaging only, by which the packaging should prepare for the worst-case thermal condition, is increasingly challenging. This follows because there can be very high-level of peak power consumption in modern processors and the power density will increase extremely high in emerging systems-on-chips. In addition, for many high-performance embedded systems the packaging requirements and operating environments render expensive and bulky packaging solutions inapplicable.

Given the cost and difficulties associated with packaging-based approaches, thermal control techniques that rely on containing input power at run-time are being used. A number of Dynamic Power Management approaches to control the temperature at run-time have been proposed, which ranges from clock throttling to Dynamic Voltage Scaling (DVS):

- Clock Throttling [6]: The clock is stalled "on-the-fly", either to save power consumption or to reduce the heat generation.

- Clock Gating [7]: The clock gating is one of the power-saving techniques used on many integrated circuits. To save power, it disables unused functional units so that their power consumption goes to zero, which enables thermal management of architecture-level sub-components and the whole circuit as well.

- Dynamic Voltage Scaling (DVS) [4]: DVS is used in a variety of modern processors. Switching between different frequency and voltage operating points, microprocessors control the power consumption level and their temperature at run-time in response to the current thermal condition. In the Enhanced Intel SpeedStep mechanism in the Pentium M processor, for example, a low-power operating point is reached in response to a thermal trigger by first reducing the frequency and then reducing the voltage [6].

Most of the above Dynamic Power Management approaches to control the temperature can be considered as various forms of processor speed control mechanisms. In the next section, we discuss how real-time systems are affected by thermal management schemes especially by changing the speed of microprocessors, and we will lay out the directions of our study.

4. Thermal Issues in Real-Time Systems

In real-time systems, the correctness of an operation depends both on its logical correctness, and on the time by which it is performed. The classical conception is that in a hard real-time system, the completion of an operation after its deadline is considered at best useless and may at worst lead to a critical failure of the complete system. A soft real-time system on the other hand will tolerate such lateness, and may respond with decreased service quality.

Hard real-time systems are common in embedded systems interacting with physical hardwares. As an example of a hard real-time system, we can think of a car engine control system, where a delayed response to a signal may lead to poor performance or to failure of the engine. Some medical systems, like heart pacemakers, and industrial process controller are good examples of hard real-time systems as well.

Live multimedia systems are typically considered to be soft real-time systems; if the processing of a video frame gets delayed, this causes the degradation of audio or video quality because the system either is forced to render the frame too late, thus giving rise to display jitter, or is forced to drop the frame all together. While this clearly affects the quality of the rendered media, the system can continue to operate without experiencing a failure.

The various run-time thermal management mechanisms are mostly based on the control of the microprocessor's speed, which has clear implications for the system's ability to meet real-time requirements. Dynamic speed scaling allows for a trade-off in hard real-time systems between the following two performance metrics: To meet the deadline constraint, we run the processor at a higher speed; To maintain the safe temperature levels, we run the process at a lower speed.

B. Overview of Research Problems and Contributions

In this study, we propose dynamic speed control methods for mixed workload with both hard real-time periodic and soft real-time or non-real-time aperiodic tasks. The control of microprocessor speed in thermally constrained systems requires an accurate temperature model by which the thermal variations of the system can be predicted as a function of the speed input.

In traditional systems, aperiodic jobs are often scheduled in the background, that is, when the CPU has no hard real-time workload to execute. Additional mechanisms are often used to trade-off execution of periodic and aperiodic portions in order to reduce response times of aperiodic jobs without missing deadlines of periodic jobs; this is called *Slack Stealing* [8]. In conventional real-time systems, jobs executing in the background can be largely ignored when determining the schedulability[1] of real-time tasks, except for the occasional priority inversion due to non-preemptibility of critical sections. When temperature comes into play, new forms of blocking are introduced which are particularly problematic because the blocking occurs potentially long after the background job has finished executing: The low-priority background job may well be freely preemptible by incoming higher-priority jobs; the high-priority job, however, can be affected at a later time as a result of the additional dissipated power and ensuing higher temperatures caused by the low-priority job. The higher temperature in turn may force the CPU to slow down, therefore affecting the high-priority job.

This is particularly a problem in the design of techniques to handle aperiodic job arrivals. For example, applying Slack Stealing [9] is difficult because slack must

[1]We say that a system is *schedulable* if the given periodic application system can indeed meet all its hard deadlines when scheduled according to the chosen scheduling algorithm [8].

be managed both in the time and in the thermal domain: Even when there is slack available in the time domain, using it for aperiodic jobs may unduly heat up the processor and so trigger dynamic speed control, which in turn will delay the execution of subsequent jobs. We say that in addition to the slack in the traditional sense, the aperiodic job is using up thermal slack as well. Similarly, bandwidth preserving algorithms, such as the Deferable Server [10] and others, must be adapted to take into consideration the thermal effect of aperiodic jobs.

These complications are particularly unfortunate since transient increase of the processor speed which we call *transient overclocking* would be an effective mechanism to reduce response times of aperiodic jobs: While periodic jobs execute at safe execution speeds, for which all the periodic jobs satisfy their deadline constraints, the processor speed can be transiently increased to provide significantly shorter response times to aperiodic jobs. Due to the additional power dissipation during this period, however, the temperature of the processor increases, and dynamic thermal management is triggered, which reduces the processor speed to keep the temperature at safe level. In this study we describe how transient overclocking can be applied to safely reduce response times for aperiodic jobs in the presence of hard real-time periodic tasks.

We propose both off-line and on-line mechanisms for overclocking aperiodic job executions. For the off-line mechanism, we describe a method to allocate overclocking budget to aperiodic workloads in a way that is efficient at run-time. This budget can then be simply consumed similarly to that of a Deferrable Server without needing to predict what the thermal effect of aperiodic-job execution is on the periodics at run-time.

Although the off-line transient overclocking mechanism for aperiodic jobs effectively reduces the response times of aperiodic workload, off-line designed budget-based schemes are inherently conservative because they have to assume worst-case task executions. Furthermore, off-line mechanisms run in open-loop mode and do not monitor the temperature level of the processor. As a result they underestimate the currently available thermal slack, which limits the reduction of job response times. We propose an on-line speed control algorithm to further reduce aperiodic job response times by taking early task termination and current processor temperature into account. We combine on-line speed control with EDF task scheduling, and we describe how to assign speed levels for periodic workload and aperiodic workload efficiently at run-time without violating either delay or thermal constraint. Both off-line and on-line speed control schemes are implemented in an event-based real-time simulation environment, and the average response times of aperiodic jobs are are evaluated for the proposed dynamic speed control algorithms.

For any dynamic thermal control to be effective, the availability of an efficient and exact thermal model is essential. The thermal behavior of system is commonly modeled using linear circuit models.

Generic thermal models only loosely capture the thermal behavior, due to variability in fabrication, environmental effects, or the need for special configurations of either cooling devices or other aspects of packaging. When the thermal management must deal with such variabilities, effective configuration and efficient calibration methods are needed to accurately parameterize the thermal models that drive the predictive thermal control. Unfortunately, as thermal models describe the relation between input power and thermal behavior, they rely on the availability of a measurement of input power for parameterization purposes. Since such measurements are typically not available at processor level in most systems, we need to find alternative ways for thermal model calibration. We describe an indirect methodology for parameterization of thermal models, which relies on the off-line analysis of the thermal behavior for a reference application, for which we measure both the detailed utilization behavior of the processor and its thermal behavior (through thermal sensors on the chip). We determine and later exploit a linear relation between energy consumption and utilization level to calibrate the thermal model for new target applications by comparing relative utilization levels. This results in accurate thermal model parameters without the need for power measurements. The proposed calibration methodology allows the thermal model to be easily established, calibrated, and recalibrated at run-time to account for different thermal behavior due to either variations in fabrication or to varying environmental parameters. We validate the proposed methodology through a series of experiments on a Linux platform.

C. Dissertation Organization

The rest of the dissertation is organized as follows. In Chapter II, we describe the models on which the thermal management are based. The dynamic speed control mechanism for thermal management is reviewed with the related work of this thesis. Chapter III describes our implementation of design time budget-based transient overclocking for aperiodic task execution under thermally constrained hard real-time systems. In Chapter IV we discuss and describe the design of on-line thermally-aware transient overclocking approach. We show that the approach minimizes the response times of aperiodic workload accounting for the actual thermal conditions and the periodic task executions. Chapter V presents an efficient methodology of thermal model calibration emphasizing the importance of correct thermal modeling for the application of proactive thermally-aware speed control. Finally, we conclude this work and discuss future research directions in Chapter VI.

CHAPTER II

BACKGROUND AND RELATED WORK

Our research is based on the power model, the thermal model, the thermal management scheme, and the scheduling of mixed task set[1]. In this chapter, we discuss those system models and task model reviewing the prior work in the areas of aperiodic task scheduling, of the dynamic thermal management mechanisms, and of thermal circuit modelings of microprocessors.

A. System Model

The thermal effects on the real-time systems are defined by the thermal characteristics of the computational resources and by the dynamic thermal management system. In this section, we review how the thermal model is formulated and how the CPU speed affects the temperature.

1. Power Model

In microprocessors, the power consumption is composed of two major sources [11, 12, 13]; Dynamic power consumption and Static power consumption.

- Dynamic power consumption P_d: The dynamic power consumption is dependent on charging/discharging of gates of transistors in circuits. It is therefore modeled as the function of processor speed, s, as follows;

$$P_d(s) = C_{eff} \cdot V_{dd}^2 \cdot s \quad , \tag{2.1}$$

[1]In the following we will make use of the notations described in Table I for system and task model.

Table I. Notations for System and Task Model

Symbol	Meaning
P	Total power consumption of CPU
P_d	Dynamic power consumption of CPU
P_s	Static power consumption of CPU
$T(t)$	CPU temperature at time t
T_a	Ambient room temperature
T_c	Critical temperature level (which is the thermal constraint)
s	CPU speed
s_H	Maximum CPU speed
s_E	Equilibrium CPU speed (at which the final reaching temperature of CPU is T_c)
R_{th}	Thermal resistance
C_{th}	Thermal Capacitance
Γ_i	A periodic task
$\Gamma_{i,j}$	The j^{th} instance of the periodic task Γi
p_i	Constant period of the periodic task Γ_i
c_i	Worst-case workload of Γ_i expressed in CPU clock-cycles required per instance
e_i	The worst-case workload of Γ_i expressed in the execution time at the speed s_E
D_i	The relative deadline of Γ_i

where $s = \kappa_v \frac{(V_{dd}-V_t)^2}{V_{dd}}$. C_{eff}, V_t, V_{dd}, and κ_v denote the effective switch capacitance, the threshold voltage, the supply voltage, and a hardware-specific constant, respectively. All hardware-specific constants and the voltage levels have nonnegative values. We simply describe the dynamic power consumption as $P_d(s) \approxeq \kappa \cdot s^\alpha$, where $\alpha \leq 3$, because the CPU speed is known to be in an approximately linear relation with the supply voltage.

- Static power consumption P_s: The static power consumption is made by the leakage current of microprocessors. If the processor temperature is constant, this static power consumption is modeled as constant value. When the temperature is related and changing, however, the static power consumption is affected and can be modeled as a linear function of temperature [12];

$$P_s = \delta T + \rho \quad , \tag{2.2}$$

where δ and ρ are nonnegative constants, and T is the chip temperature.

We now represent the power consumption as $P = P_d + P_s = \kappa s^\alpha + \delta T + \rho$ considering both the dynamic and the static power consumptions.

2. Thermal Model

The relation between processor speed and chip-level thermal behavior can be approximated at first-order by the following Fourier Model [14, 15]:

$$\begin{aligned} T'(t) &= \hat{\alpha} P - \hat{\beta}(T(t) - T_a) \\ &= \hat{\alpha}\kappa s^\alpha - (\hat{\beta} - \hat{\alpha}\delta)T(t) + (\hat{\alpha}\rho + \hat{\beta}T_a) \ , \end{aligned} \tag{2.3}$$

where T_a is the environmental room temperature and assumed to have a constant value. $\hat{\alpha}$ is the heating coefficient and $\hat{\beta}$ is the cooling coefficient of microprocessors and they represent the specific thermal characteristics of chips. If we define the *adjusted temperature* as $\bar{T}(t) = T(t) - \frac{\hat{\alpha}\rho + \hat{\beta}T_a}{\hat{\beta} - \hat{\alpha}\delta}$, we can rewrite the equation in a simpler form as follows,

$$\bar{T}'(t) = \frac{\kappa s^\alpha(t)}{C_{th}} - \frac{\bar{T}(t)}{R_{th} \cdot C_{th}} \ , \tag{2.4}$$

where $C_{th} = \frac{1}{\hat{\alpha}}$ and $R_{th} = \frac{\hat{\alpha}}{\hat{\beta} - \hat{\alpha}\delta}$. $\bar{T}(t)$ denotes the adjusted temperature at time t (with respect to the ambient temperature), and $s(t)$ denotes the processor speed at time t. In the rest of the paper, we will omit the word "adjusted" and denote $\bar{T}(t)$ simply by $T(t)$ except where it is unclear whether by temperature, we mean an adjusted temperature or an actual temperature. The parameters R_{th} and C_{th} are the thermal resistance and capacitance, respectively, and describe the thermal characteristics of the chip. This includes packaging and heat dissipation mechanisms, such as fans. The parameter κ and α describe the relation between speed and power requirement and have some positive constant values. Typically α has a value of approximately 3.0 [14, 15].

A more compact description of the thermal behavior is as follows:

$$T(t) = (h \otimes s^{\alpha})(t) \quad , \tag{2.5}$$

where $h(t)$ is the impulse response function

$$h(t) = \frac{1}{RC} e^{-t/RC}, t \geq 0 \quad . \tag{2.6}$$

B. Task Model

Our research is centered on the thermally-aware speed control for mixed workload. We discuss briefly in this section about the definitions and the specifics of periodic and aperiodic tasks.

1. Periodic Tasks

The period is the amount of time between each iteration of a regularly repeated task. Such repeated tasks are called *periodic tasks.* Every job in periodic tasks has to be deadline-oriented, which means every job has to be accomplished within the set deadline. The deadline is a constraint on the latest time at which the operation has to come to the end. A job in a task that is released at t must complete D units of time after t; D is the (*relative*) *deadline* of the task. We usually assume that D is equal to the length of period p for all tasks in systems. This requirement is consistent with the throughput requirement that the system can keep up with all the work demanded of it all times.

One example of periodic tasks is the cruise control mechanism on an automobile. The purpose of cruise control is to keep the speed of a vehicle constant automatically.

When the cruise control is active and a desired speed is set, the embedded control system should monitor and adjust the vehicle speed regularly until the driver turns off the cruise control mechanism. The frequency in which the computer checks and adjusts the current speed is called as the control rate and it is fixed by the control system designer. We can regard the cruise control as a periodic task and the periodic speed monitoring/controls as jobs of the task.

For the periodic workload model, we consider the Liu and Layland periodic task model [16] that defines a task Γ as (p, e), where p is the period of Γ and e is the execution time requirement of Γ. If we need to consider the early deadlines of periodic tasks specifically, we define each task as Γ_i as (p_i, e_i, D_i).

2. Aperiodic Tasks

All real-time tasks need not to be periodic. Aperiodic tasks respond to randomly arriving events. The jobs in an aperiodic task, however, need to be similar in the sense that they have the same statistical behavior and the same timing requirement. That is, their interarrival times and the execution times of jobs are identically distributed with some probability distributions respectively. To distinguish aperiodic tasks from sporadic tasks, we say that a task is aperiodic if the jobs in it have either soft deadlines or no deadlines [16]. For example, the processing of user inputs from terminal application is considered as an aperiodic task since there is no deadline for the task execution but the shorter response time is desirable.

C. Scheduling of Mixed Task Set

In the context of traditional and energy-constrained real-time systems, a variety of approaches to scheduling a mixture of aperiodic tasks and periodic hard real-time tasks have been proposed. The simplest and least effective of these is to execute aperiodic tasks at a lower priority level than any of those with hard deadlines. This effectively relegates aperiodic tasks to background processing. Although this method satisfies the schedulability of periodic tasks, response times of aperiodic tasks are prolonged unnecessarily.

A variety of dedicated scheduling servers have been proposed to handle aperiodic tasks. As a general periodic task, the server is characterized by the pair (e_s, p_s), where e_s is the maximum budget and p_s is period of the server. The simplest server is the Polling Server (PS). PS is periodically activated and services the pending aperiodic tasks until the budget is exhausted. The budget can be replenished again at every activation of the server. If there is no pending aperiodic task in the task queue, PS immediately suspends itself exhausting the budget out until it is reactivated in the next period. Thus, if an aperiodic task arrives slightly after PS checks the empty aperiodic task queue, the beginning of service for it is delayed until the next activation of PS.

To overcome this unnecessary delay for aperiodic tasks, various bandwidth-preserving servers have been proposed. They preserve their budgets even when there is no pending aperiodic task in the queue instead of exhausting all remaining budget. Whenever any aperiodic task arrives and there is budget remaining, the server can activate and service the task. The Deferrable Server (DS)[10] is the simplest of bandwidth preserving servers. DS works in the same way as PS except the feature of budget preservation. This feature improves the performance for the response time of aperiodic tasks. This simple way of preserving budget, however, could make the lower-priority periodic task miss its deadline. The Sporadic Server (SS)[17] was proposed to resolve the problem. Using a rich combination of replenishment and consumption rules, it guarantees that no periodic task miss its deadlines in the presence of aperiodic tasks.

While the DS and SS are commonly used for Rate-Monotonic (RM) scheduling, the Total Bandwidth Server (TBS)[18] and the Constant Bandwidth Server (CBS)[19] are more suitable for Earliest-Deadline-First (EDF) scheduling. Assigning deadlines for the aperiodic server dynamically and using their special budget replenishment rules, they prevent aperiodic tasks from affecting the schedulability of hard real-time periodic tasks.

In general, all bandwidth preserving servers mentioned improves responsiveness over the polling approach. A number of problems are shared by these schemes, however: First, they tend to degrade to providing the same performance as the polling server at high loads. Next, they are unable to make use of slack time which may be present due to the favorable phasing of periodic tasks. Finally, they are also unable to reclaim spare budget gained when hard real-time tasks require less than their worst case execution time.

Lehoczky and Ramos-Thuel presented a different approach to service aperiodic requests, known as the Slack Stealer[9] which partly addresses some of the issues with bandwidth-preserving servers. The Slack Stealer addresses the problem of minimizing the response times of aperiodic tasks guaranteeing that the deadlines of hard periodic tasks are met. It steals all available slacks from periodic tasks and gives it to aperiodic tasks. Conceptually, it is known as the optimal aperiodic task server for the minimization of response times. Practically, however, the complexity in realization of the approach in a system limits its application. Thus, various approximate slack stealing algorithms were proposed [20].

The traditional schemes of handling aperiodic tasks in real-time systems have only the timing constraints for periodic tasks running with the aperiodic tasks. When thermal constraints are considered, however, the increase of temperature by aperiodic job execution affects the execution of periodic tasks. Since the increased temperature means the higher possibility of slowing down of periodic job execution to keep system temperature within the safe thermal level resulting in the delayed response time of periodic task. Without the appropriate thermal slack computation for aperiodic task, we should have the deadline violations of periodic tasks. In this study, we propose the design of thermally-aware speed control algorithms for aperiodic job executions and compare the performances.

D. Dynamic Speed Control for Thermal Management

For the safe operation of the system, the temperature must be prevented from reaching the critical junction temperature and so destroying the processor. In our study, we regard the system is *thermally safe* when the microprocessor's temperature is guaranteed to be equal or less than the critical temperature T_c at all times during the task executions. The thermal safety of systems is achieved through various forms of Dynamic Thermal Management, many of which are equivalent largely to forms of controlling the CPU speed. For example, constant speed scaling uses a sufficiently slow constant speed to keep the processor at a safe temperature. It has been shown [21] that better system utilization can be achieved with *dynamic speed scaling* schemes. Rao *et al.* [22] use variational calculus to find the optimal continuous-speed function to optimize utilization over a given interval. They also show that a simple, two-speed scheme approximates the optimal speed function well.

We first used a particular type of two-speed scheme, called *Reactive Speed Scaling* (RSS) in [21, 23] to improve the schedulable utilization and reduce worst-case delays in hard real-time systems: Whenever the CPU is busy, it is allowed to run at high speed s_H until it reaches a maximum temperature T_c at a safe margin from the junction temperature. Once T_c is reached, the CPU continues execution at a reduced *equilibrium speed* s_E, which keeps the temperature at or below T_c. We next extend to the continuous-speed scheme to apply to the online optimal speed control for the minimization of aperiodic response time. In the continuous-speed scheme, we predicts the job completion times and the instant at which the processor temperature reaches T_c with the thermal model for the runtime optimization. In this study we make use of RSS to formulate a scheme for *transient overclocking* of the processor to reduce the response time of aperiodic jobs.

There is an extensive literature on dynamic speed control on processors both in general-purpose and embedded applications. However, the majority of this literature focuses on power management for the purpose of saving energy, *not* for maintaining safe temperature levels [24, 25, 26, 27, 28, 29, 30]. While energy and temperature are closely related, power control mechanisms for energy and temperature are quite different. It has been shown that many energy-saving techniques do not work well in reducing peak temperature [14, 15, 31]. This is due to the fact that energy-aware techniques focus on dealing with the *average* power consumption while temperature-aware ones focus on handling *peak* power consumption [15].

The work on dynamic speed scaling techniques to control temperature in real-time systems was initiated in [14] and further investigated in [15]. Both [14] and [15] focused on online algorithms in real-time systems, where the scheduler learns about a task only at its release time. In contrast, in our work we first assume a predictive task model (e.g., periodic tasks) and so allows for design-time delay analysis and then propose an online algorithm for mixed workload in hard real-time systems.

In [22, 32], optimal speed profiles were derived to achieve high resource utilization. In [4, 33, 34], the use of the feedback control theory is proposed as a way to implement adaptive techniques in the processor architecture. In [34], a predictive frame-based DTM algorithm is presented. In [33]. a predictive DTM algorithm was designed to improve the performance of multimedia applications.

There are also many other run-time thermal management techniques studied for general-purpose applications. The authors in [4] perform extensive studies on empirical DPM techniques for thermal management. Their results show that DVFS can be very inefficient if the invocation time is not set appropriately. In [35], a thermal model was presented that is capable of modeling cooling faults such as CPU fan or case fan failures and load-balancing algorithms were designed based on this model. In [36, 37, 38], temperature-aware floorplanning is used to place circuit blocks, such that an even thermal profile is obtained.

System-level solutions have been defined to reduce the temperature in MPSoCs using different scheduling mechanisms [39]. Finally, in [40], a detailed review of thermal management techniques for multi-core architectures is presented.

Most of existing thermal management techniques are based on temperature monitoring and tuning of microprocessor speed that do not result in optimal solutions. Moreover, they do not provide a guarantee that the temperature constraints will be satisfied at all instances of operation, which is critical for achieving system reliability. Zhang and Chatha [41] addressed the knapsack problem for a given execution sequence of jobs by assigning discrete frequency/voltage states. They proved that the problem is NP-hard and proceeded to formulate a pseudo-polynomial optimal speed assignment algorithm and a polynomial time approximation algorithm.

In [42], it classifies existing dynamic speed control algorithms for real-time systems into two categories. One is *intra-task* dynamic speed control algorithms, which uses the slack time when a task is predicted to complete before its worst-case execution time. The other is *inter-task* dynamic speed control algorithms, which allocates the slack time between the current task and the following tasks. The difference between them is that *intra-task* speed controlling strategies adjust the microprocessor speed during an individual task boundary, while *inter-task* strategies adjust the speed task by task. In most systems, it is difficult to predict well the earlier completion of tasks before their worst-case execution time. We, therefore, consider *inter-task* dynamic speed controlling strategy and the *RSS* scheme for the thermal management.

In our study, we focus our work on the efficient speed control to minimize the response time of aperiodic workload in the presence of hard real-time periodic tasks satisfying the temperature constraints.

E. Thermal Parameter Calibration

The performance-power optimization of processors using the speed control policies has received considerable attention. Most of these research, however, does not consider thermal constraints of systems. In the thermally-aware design domain, two important subproblems can be considered: 1) Modeling the thermal behavior of systems and 2) Designing methods to control processor performance and power consumption to meet the temperature constraints. Several recent works have addressed the issue of on-chip thermal modeling [38, 43, 44, 45, 46, 47, 48]. At the physical level, various methods can be used to model the heat transfer in the substrate. Finite-difference time domain [43], finite element [38], model reduction [44], random walk [45] and Green-function [48] based algorithms have been applied for on-chip thermal analysis. At the architectural level, [46] presents a thermal/power model for super-scalar architectures. The work presented in [47] investigates the impact of temperature and voltage variations across the die of embedded cores. Based on these and other similar models, Dynamic Thermal Management (DTM) techniques have been suggested [4, 33, 34, 49]. A major impediment when putting proactive thermal management into practice is the need to develop a thermal model that is appropriate for the platform at hand. Generic thermal models only loosely capture the thermal behavior, due to variability in fabrication, environmental effects, or the need for special configurations of either cooling devices or other aspects of packaging. The models must therefore be appropriately calibrated before use. In this study we describe an efficient methodology for parameterization of thermal models. It relies on an off-line analysis of a reference application for the thermal behavior, which is based on the measurement of the utilization behavior and the thermal behavior of the system at hand running the application.

CHAPTER III

TRANSIENT OVERCLOCKING FOR APERIODIC TASK EXECUTION IN HARD REAL-TIME SYSTEMS

A. Introduction

In this chapter we describe how transient overclocking can be applied to safely reduce response times for aperiodic jobs in the presence of hard real-time periodic tasks. To obtain mechanisms that are efficient at run-time, design-time method to allocate overclocking budget to aperiodic workloads is proposed. This budget can then be simply consumed at run-time similarly to that of a Deferrable Server [10] without heeding to predict what the thermal effect of aperiodic-job execution is on the periodics. Furthermore, there is no need to monitor the temperature of microprocessors online with the management of overclocking budget.

This chapter will be organized as follows: In Section B, we will describe the system model used in the chapter. We will present the thermal model, which describes the interaction between processor speed and temperature. We will also describe a basic dynamic speed control mechanism that can be easily applied for transient overclocking. The task model will be described. In Section C we present the Transient Overclocking Server. We describe the principles of operation and the design-time computation of safe budget levels. In Section E we evaluate the performance of the Transient Overclocking Server. Finally, we conclude the chapter in Section F with summary and an outlook on unresolved issues.

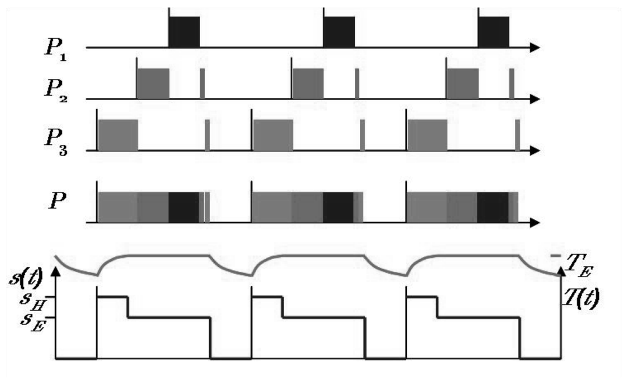


Fig. 2. The critical instance of periodic tasks

B. System Model

The effect of transient overclocking on periodic tasks in a thermally constrained system is defined by the thermal characteristics of the computational resources and of the dynamic thermal management system, the specifics of the periodic task model, and of the characteristics of the aperiodic job arrivals.

1. Thermal Model

The relation between processor speed and chip-level thermal behavior can be approximated at first-order by the following Fourier Model as shown in Chapter II:

$$T'(t) = \frac{\kappa s^{\alpha}(t)}{C} - \frac{T(t)}{R \cdot C} , \tag{3.1}$$

where $T(t)$ denotes the temperature at time t (with respect to the ambient temperature), and the processor speed $s(t)$ at time t. The parameters R and C are the thermal resistance and capacitance, respectively, and describe the thermal characteristics of the chip.

2. Periodic Hard Real-Time Tasks

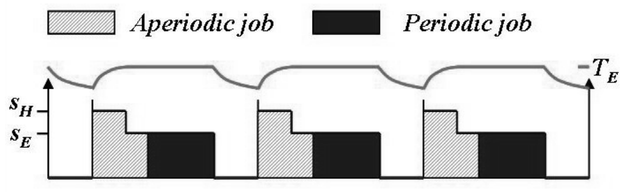
We consider a set of identical-period hard real-time tasks $\{\Gamma_i : i = 1, 2, \ldots, n\}$, where each task $\Gamma_i = (p, c_i)$ has a minimum time period p between jobs and each job requires c_i processor cycles to complete in the worst case.[1] We adopt a fixed-priority scheduling scheme, and Γ_i is assigned priority i (the smaller the index, the higher the priority).

To guarantee the schedulability of periodic tasks, we first determine the thermal steady-state for the execution of periodic tasks. Figure 2 shows the critical instance under the thermal constraint for identical-period tasks [21]: When the periods of tasks are identical and no aperiodic jobs are present, the system can reach a worst-case steady state, in which the temperature - call it T_0 - at the period boundary is maximized, and therefore the amount of time the system runs at high speed s_H is minimized. As a result, the schedulable utilization is minimized. Wang *et al.* proved in [21] that, in a static-priority system, the critical instant for a medium-priority task, say Γ_2 in Figure 2, is when it is released in the thermal steady state together with all higher-priority tasks, just after all lower-priority tasks just completed (and therefore heated up the processor).

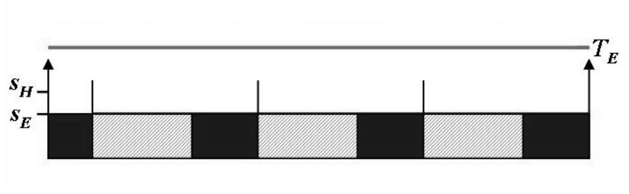
3. Aperiodic Tasks

The system shown in Figure 2 leads to two observations. *Observation 1:* The steady-state is robust for periodic arrivals; whenever the execution time of a task invocation is shorter than specified, or whenever the inter-arrival time of tasks exceeds the period, the CPU may temporarily cool off. As a result, the speed control will kick in later

[1]We limit ourselves to identical-period tasks. The critical instance for arbitrary-period task set is an open problem. For general arrival curves we developed a bound on the critical instant in [23].



(a)



(b)

Fig. 3. Worst-case execution with aperiodic tasks

during the next busy period, and the steady state will eventually be reached again. *Observation 2:* Any additional execution of (aperiodic) jobs disrupts the steady state: the initial temperature at the beginning of the period increases, and less idle time for processors remains. Figure 3 illustrates the effects of allowing increasing amounts of aperiodic jobs to execute per period. Figure 3(a) shows how aperiodic and periodic workloads can be executed together, with the aperiodic making use of transient overclocking in an RSS fashion[2]. Increasing the amount of aperiodic execution pushes the completion time of periodic tasks back towards the end of the period. This in turn prevents the CPU from cooling off after the busy period ends, which in turn triggers the speed control to kick in earlier in the next period. As a result, the completion of the periodic tasks is pushed back even further. As this happens, the opportunities for transient overclocking disappear. Figure 3(b) shows the case where uncontrolled exe-

[2]In order to simplify the presentation, the illustrations in this paper display systems with a small thermal capacitance with respect to periods and execution time. The results can be applied to thermally more inert systems by looking at multiple periods.

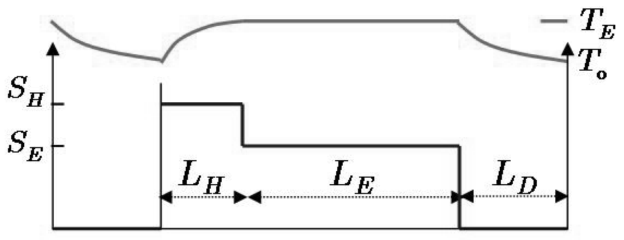


Fig. 4. Thermal steady state of system

cution of aperiodic leads to the RSS system degenerating to a constant-speed system, which executes at low speed throughout.

4. Thermal Steady State of System

In this section, we describe how to compute the durations of maximum and of equilibrium processor speed to accommodate periodic and non-periodic workloads with Reactive Speed Scheduling. We say that the temperatures at the beginning and end of a period are equal in the *thermal steady state.* Based on the thermal model in Section 1, we can derive the relations among durations, temperature, and the processor speed.

Figure 4 shows one of the periods of a system that is in the thermal steady state by an identical periodic task set and some amount of periodic budget consumption for aperiodic jobs. According to the Fourier thermal model, the relationship among the duration L_H of the processor's maximum speed execution, the temperature T_0 at the beginning of the period, and the processor speeds s_E and s_H can be expressed using the following equation:

$$T_E = T_H + (T_0 - T_H)e^{-L_H/\tau}, \tag{3.2}$$

where $T_H = R\kappa s_H^\alpha$, $T_E = R\kappa s_E^\alpha$, and $\tau = RC$.

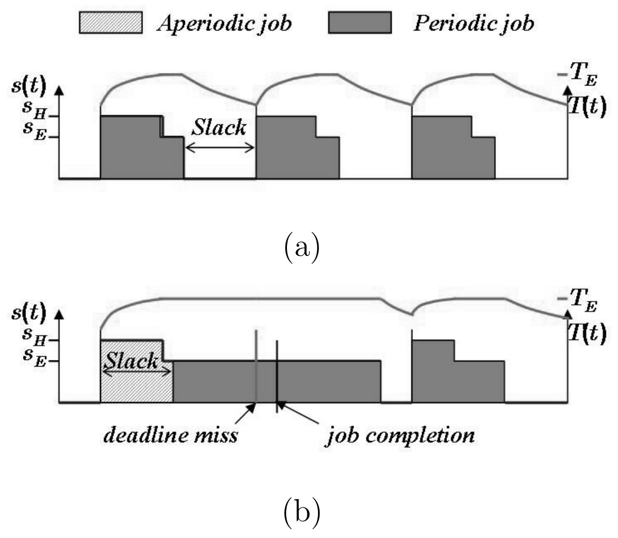


Fig. 5. Erroneous slack computation leads to deadline miss

From (3.2), T_0 can be expressed as

$$T_0 = T_H - (T_H - T_E)e^{L_H/\tau} \tag{3.3}$$

depending on the duration L_H of the processor's maximum speed execution in a period. Similarly, the length of the idle time L_D is also obtained from the thermal model as follows:

$$L_D = \tau \ln\left(\frac{T_E}{T_0}\right) \quad . \tag{3.4}$$

Finally, the length L_E of equilibrium-speed execution is expressed simply as $L_E = p - (L_H + L_D)$, where p is the period of tasks which is identical for all tasks.

Thus, for any given single-period workload, the temperature T_0 at the beginning of a period and the durations of L_H, L_E, and L_D in the thermal steady state of system are obtained by the above relations based on the thermal model.

C. Aperiodic Task Servers and Transient Overclocking

For traditional real-time systems, Slack Stealer [9] is known to be an optimal aperiodic-task server. When the system is thermally constrained, however, conventional ways of slack computation without special consideration for temperature do not allocate the proper amount of slack, which can lead to deadline misses. Figure 5 shows a situation where a deadline is missed in a system with RSS for periodic and aperiodic jobs. This is due to a slack computation that fails to consider thermal effects. In this example, the slack allocated to aperiodic jobs should be used to compensate for the transient overclocking portion at the beginning of the period. If an aperiodic job used this slack, the periodics cannot take advantage of overclocking, and misses the deadline (Figure 5(b). Since the example in Figure 5 uses an RSS scheme that does not differentiate between periodic and aperiodic workload, the periodic job in Figure 5(b) is penalized as it gets "crowded out" of the overclocking period, and thus further delayed.

Accurate slack management in systems that use transient overclocking comes at a high run-time overhead: Upon arrival of the aperiodic job, the amount of slack must be computed based on:

- current temperature of the processor,
- current speed of the processor,
- remaining execution time of periodic jobs.

Since the relationship between current temperature and available slack is not linear, due to the transient overclocking, this computation is too costly to be performed at run-time: the slack stealer would have to predict when to transition from overclocking back to equilibrium speed before computing the available slack. We therefore do not expect on-line naïve slack stealers to be used in practice, despite their optimality. We will propose an on-line thermally-aware slack stealer with the effort of minimizing the overhead in next chapter.

1. Bandwidth Preserving Server under Thermal Constraint

Since slack stealing cannot be applied directly, we resort to a budget-based, bandwidth-preserving scheme: In this chapter, we present the *Transient Overclocking Server*, which allows for a very efficient management at run time of allocated budget to aperiodic jobs. Budget management needs only $O(1)$ online computation using simple consumption and replenishment rules: Whenever aperiodic jobs are ready and budget remains, the budget is consumed at a pre-determined rate, and it is periodically replenished to a pre-defined amount. In order to enable this efficient run-time operation, the budget has to be computed offline. We partition the budget into two portions, an *overclocked* portion B_H of length L_H (we call this loosely the "overclocking budget") and a *non-overclocked* portion B_E of length L_E. The processor is overclocked at speed s_H during the overclocked portion and runs at equilibrium speed s_E during the rest of the time, in addition to the idle time necessary to compensate for the overclocked portion. In next section, we describe how to compute L_H and L_E and how to manage the budget amount.

2. Design of Transient Overclocking Server

The offline computation of the overclocking budget B_H presupposes a worst-case steady state for the processor temperature. Figure 3(a) and Figure 3(b) indicate how this steady state is linked to the overclocking budget.

3. Budget Computations

The budget is computed at design time by determining the lengths L_H and L_E of the overclocked and non-overclocked portions, respectively. At run-time, the budget is consumed in RSS fashion: first, any available overclocking budget B_H is consumed, followed by the non-overclocking budget B_E.

Theorem 1. *When we apply two different speed levels in the execution of a given job with fixed durations of high-speed and low-speed of the processor, the amount of temperature increase is the smallest when we have the order of 'high-speed to low-speed.'*

Proof. Since the temperature variation by the change of processor speed and the power thereof is interpreted into the linear RC-electric circuit model [35], the temperature as an output to the power input can be superposed. Thus, the comparison between temperature increases by two different orderings of speed assignment is same as the description of *Lemma* 1 in [21] with a level of offset.

Lemma 1. *[21] Given a time instance t, we consider a successive execution part of job with a starting time t_0 and the completion time t_1, where $t_0 < t$ and $t_1 < t$. We assume the system is idle during $[t_1, t]$. Define T_t as the temperature at t. If we shift this part of job into a starting time t_0^* and a completion time t_1^* such that $t_0 < t_0^* < t$ and $t_1 < t_1^* < t$. Define T_t^* as the new temperature at t. Then we have $T_t \leq T_t^*$.*

Based on the *Lemma*, running the processor at high-speed earlier than at the other lower speed level increases the temperature less than the other way of speed assignments. □

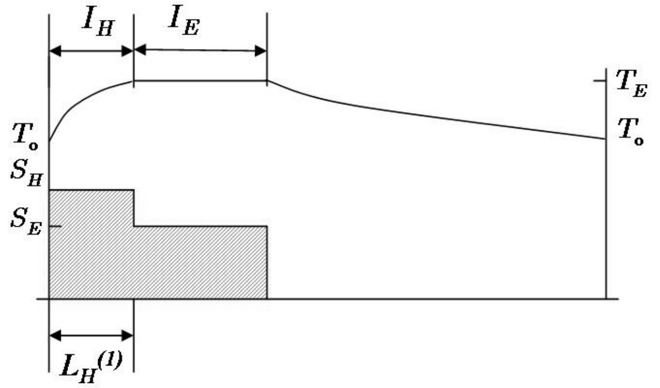
Theorem 2. *For a given thermal constraints, the RSS-style consumption of the budget of which the budget consumption order is B_H followed by B_E, maximizes the* *amount of processing clock-cycles that can be handled by the transient overclocking server.*

Proof. Suppose a fixed length of continuous CPU execution, l, composed of l_H and l_E in written order, where l_H is the duration of CPU runs with high speed s_H and l_E is the duration of equilibrium speed s_E. By the Theorem 1, if we reverse the order of the execution durations to (l_E, l_H), the temperature at the end of the length becomes higher. In the case, under thermal constraints, l_H should decrease to satisfy the constraints. By the decrease of high-speed duration to $l'_H (l'_H < l_H)$, B'_H becomes smaller than B_H so that the total available budget B diminishes by the reverse ordering of budget consumption. Thus the RSS-style consumption of the budget maximizes the amount of processing cycles for the transient overclocking server under a given thermal constraints. $\square$

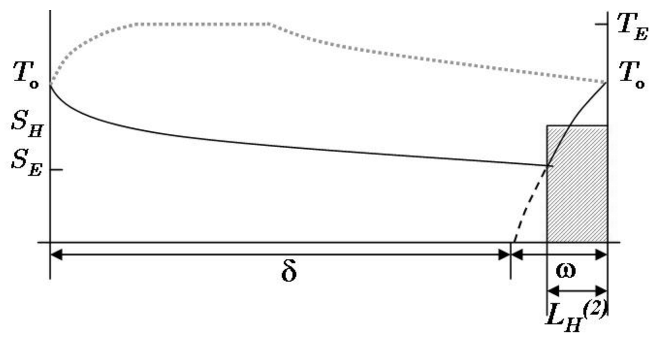
In the following section we describe how to compute the amounts B_H and B_E of overclocking and non-overclocking budget, respectively. We first describe the budget computation in the absence of periodic workload. We then explain the integration with periodic workloads.

a. Overclocking without Periodic Tasks

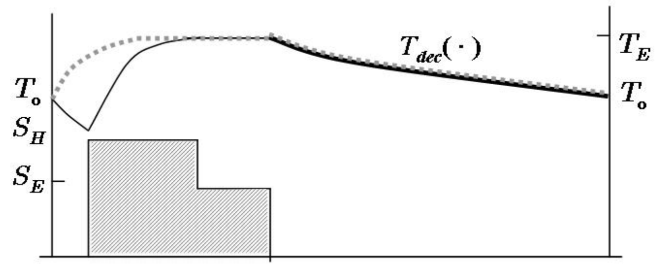
Overclocking Budget B_H. Figure 6 shows three scenarios of arrivals of aperiodic jobs that are sufficiently long to make use of all the available budget. The figure also shows the resulting temperature curves and available clock cycles within a period under a steady-state thermal envelope for the case of no periodic tasks. In the figure, I_H and I_E are the maximum durations of speed s_H and equilibrium speed s_E so that a steady-state temperature of T_0 can be kept at the beginning of each period. Although the duration of the overclocking portion, L_H, changes depending on current



(a) Case 1



(b) Case 2



(c) Other Cases

Fig. 6. Temperature curves and available clock cycles

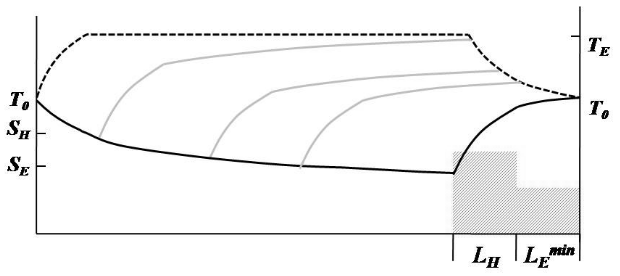


Fig. 7. Finding non-overclocking executing duration

temperature, the minimum safe length can be determined at system design time to guarantee the thermal constraints.

Lemma 2. *In the overclocking without periodic tasks, the maximum safe amount of overclocking budget B_H is $min(L_H^{(1)}, L_H^{(2)})$, where $L_H^{(1)}$ is the duration for the aperiodic job is ready and executed from the beginning of a period until the temperature increases to the critical level. And $L_H^{(2)}$ is the duration for which the processor can run at s_H so that the temperature increases to the level T_0 at the end of a period after some processor idle duration.*

Proof. We look at two cases, first.

Case 1: The aperiodic job is ready at the beginning of a period, as shown in Figure 6(a). $L_H^{(1)}$ is easily computed, given the initial temperature, T_0. L_H then is equal to I_H in Figure 6(a). We obtain the solution from Equation (3.2):

$$L_H^{(1)} = \tau \cdot \ln \frac{(T_H - T_0)}{(T_H - T_E)}. \tag{3.5}$$

Case 2: The processor is idle, and the aperiodic job arrives before the end of the period. Since the processor is now overclocked until at least the end of the period, the temperature rises quickly. As shown in Figure 6(b), the point of intersection between the decreasing thermal curve, $T_{dec}(t)$, of the idling processor and the increasing thermal curve, $T_{inc}(t)$, of the overclocked processor has to be formulated to obtain the length $L_H^{(2)}$. $T_{dec}(t)$ and $T_{inc}(t)$ are described as following equations;

$$T_{dec}(t) = T_0 \cdot e^{-t/\tau}. \tag{3.6}$$

$$T_{inc}(t) = T_H - T_H \cdot e^{-(t-\delta)/\tau}, \tag{3.7}$$

where $\delta = p - \tau \cdot \ln(\frac{T_H}{T_H - T_0})$.

Thus, the intersection happens at $t_x = \tau \cdot \ln\left(\frac{T_0 + T_H \cdot e^{\delta/\tau}}{T_H}\right)$, and $L_H^{(2)}$ for Case 2 is computed as the length $L_H^{(2)} = p - t_x$.

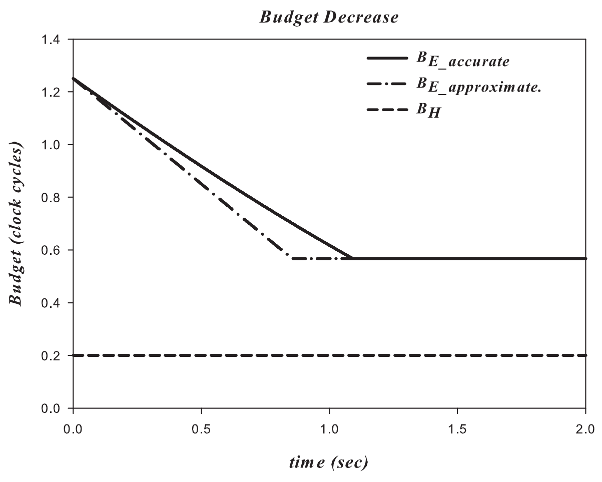
Now, for other cases when the aperiodic job is ready to execute at a moment different from that of the above two cases, the value for L_H exceeds $L_H^{(1)}$ and $L_H^{(2)}$. That is because the difference of current temperature at the moment and the temperature it has to catch up to maintain thermal steady state is larger than that of two cases above. No computation for other cases are therefore needed for L_H. Thus, the length L_H is determined by comparing the two values for Case 1 and Case 2, $L_H = min(L_H^{(1)}, L_H^{(2)})$. □

The overclocking budget B_H is obtained by simply multiplying L_H by s_H.

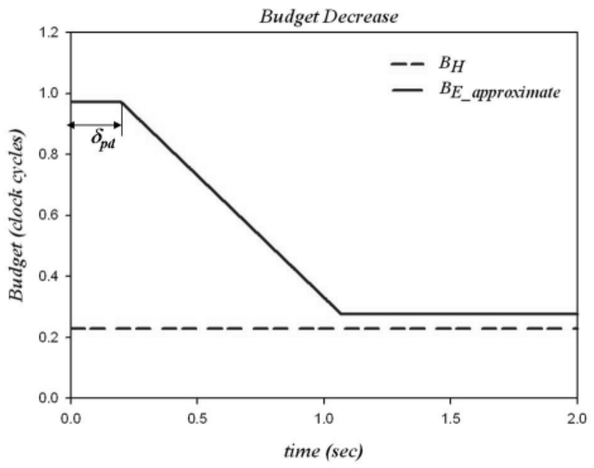
Non-overclocking Budget B_E. The duration L_E is the length for which aperiodic jobs execute at speed s_E after the overclocking budget B_H is exhausted. At that point, the processor may continue to execute at speed s_E for a while without violating the thermal steady state. The amount of this non-overclocking budget B_E varies over the length of the period, as can be seen from Figures 7 and 8.

Lemma 3. *In the overclocking without periodic tasks, non-overclocking budget B_E is bound approximately by the following function in a period*

$$B_E(t) = max(B_E(0) - t \cdot s_E, B_E^{min}), \tag{3.8}$$



(a) No periodic tasks



(b) With periodic tasks

Fig. 8. Decrease of budget for aperiodic task server in period

where $B_E(0) = I_E \cdot s_E$.

Proof. Given the overclocking budget $B_H(L_H)$, non-overclocking budget $B_E(L_E)$ can be computed to satisfy the thermal envelope built based on the pre-supposed thermal steady state. At the beginning of the period, B_E is the largest, as the processor has the most time to cool off before the end of the period. As time progresses and the processor idles, B_E decreases, as the deferred execution of the entire budget would cause T_0 to be exceeded at the end of the period. Toward the end of the period, B_E is zero, as the execution of deferred overclocking budget B_H would cause T_0 to be exceeded otherwise. Although the function decreases in convex shape, we approximate it in linear fashion for the fast online computation. We denote the approximate function as "$B_{E\_approximate}$" in Figure 8. That is, B_E decreases at the rate of s_E (In other words, L_E decreases at rate 1) while the system is in idle state and the amount is larger than B_E^{min}. It is consumed at low speed rate, s_E, whenever B_E is used for aperiodic jobs.

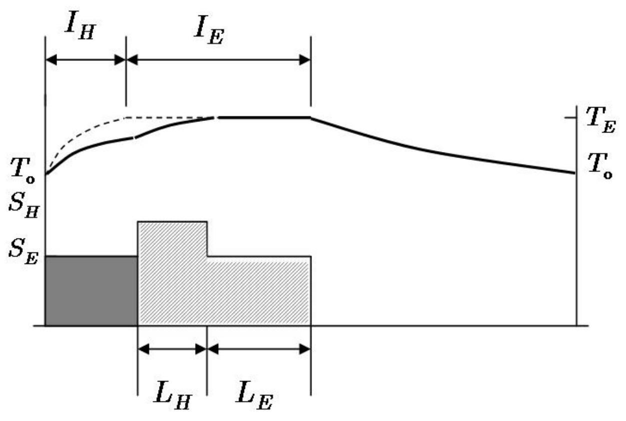
The minimum amount of B_E is computed at the moment from which the successive execution of B_H and B_E makes the temperature at the end of a period be equal to T_0. Figure 7 shows the case for the computation of B_E^{min}.

In practice, towards the end of the period, both B_H and B_E are constrained by *time* rather than *temperature*: Even for large value of B, there is not sufficient time remaining in the period to consume the budget. For this reason, the value for B_E in Figure 8 is displayed as $B_E(t) = max(B_E(0) - t \times s_E, B_E^{min})$. □

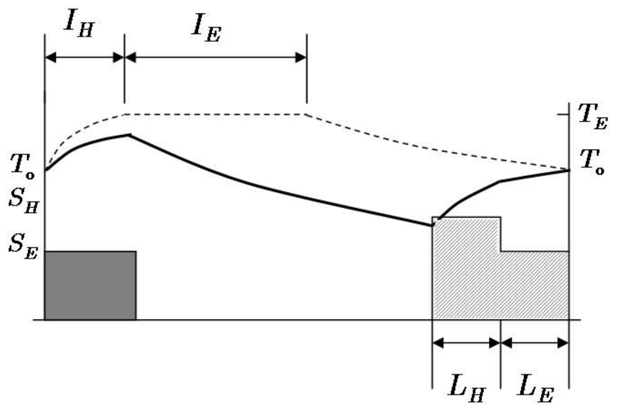
b. Combining Periodic and Aperiodic Jobs

For the integration of the Transient Overclocking Server with periodic tasks, we make three assumptions: First, all periodic tasks are schedulable in the system with constant-speed (s_E) scaling. That is, without aperiodic workload there is no need for overclocking. Second, the periods of periodic tasks are identical. Third, for the simplicity in handling the budgets, the budget replenishment period is set to be identical to that of the periodic tasks.

We define two strategies for dealing with periodic tasks depending on whether we allow to overclock the execution of periodic tasks as well, in addition to that of



(a) Case 1



(b) Case 2

Fig. 9. Temperature curves and available clock cycles: integration with periodic task

aperiodic jobs.

In the *budget-sharing* approach, both the periodic and aperiodic jobs can be overclocked. If a periodic task becomes ready when no aperiodic job is in the queue, it can make use of the overclocking budget, in the way described in Section a. While this approach is very simple, it penalizes aperiodic jobs (in particular short ones) by sharing the overclocking budget to the periodic tasks, which carry no benefit from it.

In the *non-budget-sharing* approach, the overclocking budget is used only for aperiodic jobs, and the periodic jobs execute only at low speed, s_E. Since we reserve the overclocking for aperiodic jobs only, we achieve better average response times of aperiodic jobs.

Lemma 4. *In the case of budget sharing, the maximum available budget computed in Section a is used both for periodic task and aperiodic task without ruining the schedulability of periodic tasks.*

Proof. Since the periodic task set is schedulable at constant low speed s_E, it must be schedulable after addition of the transient overclocking server as well. This is because the overclocking server executes the periodic task at high speed as well in the way of budget sharing so that the response time gets shorter. In this way, periodic tasks are always schedulable in budget sharing strategy. □

Lemma 5. *For the non-sharing approach, the amount of maximum safe available budget under thermal constraints is obtained by comparing two cases: either the aperiodic execution directly follows the periodic task set, or it directly preceeds the periodic task set before the beginning of next period.*

Proof. On the contrary to the case without periodic tasks, the available safe durations of s_H and s_E execution become shorter because the processor is heated by the execution of periodic task before the start of budget consumption. With the higher beginning temperature, the duration until the temperature hits the critical level earlier than the case with no periodic task. Therefore, we need to look into cases that aperiodic jobs are executed after the completion of periodic jobs in each period for the computation of the available budget in the strategy. These cases are illustrated in Figure 9. (We do not consider the thermal effect of back-to-back executions of the aperiodic server since we are making sure that the temperature at the beginning of the period does not exceed the the steady-state temperature T_0). □

The resulting budget allocation is illustrated in Figure 8(b). The length δ_{pd} in the figure describes the execution duration of the periodic tasks. The computation of budget follows the same way as that for the budget sharing approach except for the thermal effects of δ_{pd} by periodic tasks.

D. Queueing Model

In this section, a queueing model for budget sharing approach is developed to analyze the performance of the server under the thermal constraints. For a simple approximation, average processor speed, s_{avg} is introduced in the analysis.

Although there must be a little amount of error for the accurate response time computation with the use of average processor speed, the error can be canceled out by the probabilistic characteristic of aperiodic tasks and the effect of durations of s_H, and s_E is well reflected on s_{avg}.

With the average processor speed, s_{avg}, the queueing model analysis for the transient overclocking server can be done in the same way as that for the conventional slack stealing algorithm.

1. Arrivals and Computation Times of Aperiodic Task

• Arrivals of aperiodic tasks form a Poisson process with rate λ, i.e., the CDF of inter-arrival time S of aperiodic tasks is [50]

$$F_S(s) = 1 - e^{-\lambda s} \quad s \geq 0. \tag{3.9}$$

• The computation times of aperiodic tasks are exponentially distributed with mean $1/\mu$ at s_H, i.e., the computation time X of an aperiodic task is random with the CDF

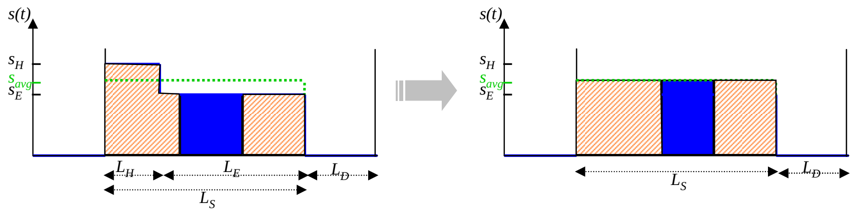


Fig. 10. Approximation using average processor speed s_{avg}

[50]

$$F_X(x) = 1 - e^{-\mu x} \quad x > 0. \tag{3.10}$$

2. Modeling Transient Overclocking Server in Thermally Constrained Environment

Figure 10 illustrates how we simplify the effect of reactive speed scheduling by replacing the multiple speed levels by an *average processor speed* to compute and approximate the average response time analysis for the aperiodic tasks.

The average processor speed is obtained using following equation:

$$s_{avg} = (s_H L_H + s_E L_E)/L_S \quad , \tag{3.11}$$

where $L_S = L_H + L_E$.

From this point on with average processor speed, we are able to apply the conventional way analysis of slack stealer except the processor-idle duration. In the design of transient overclocking server under thermal constraints, the effect of the duration is reflected on the average speed s_{avg}. That is, if L_H is designed to be long, s_{avg} increases, otherwise s_{avg} decreases [3].

Let Z be the first service time duration that an aperiodic task receives from

[3]We note that, due to the thermal constraints, the total amount of cycles available during interval L_S is not constant with varying L_H.

the transformed(averaged) transient overclocking server [4]. Obviously, Z is a random variable taking value between zero and S_A (maximum available service duration in one period by the transient overclocking server). The amount of Z is different depending on the beginning moment of execution of the task in a period. The probability that a server is busy can be approximated by $\lambda/\left(\frac{S_A}{p}\cdot\eta\cdot\mu\right)$ using an `M/M/1` queueing model, where p is the budget replenishment period, S_A is the maximum available service duration in a period, and η is the ratio between s_{avg} and s_H. Then, the probability that Z is S_A is $1-\lambda/\left(\frac{S_A}{p}\cdot\eta\cdot\mu\right)$. The distribution of Z other than the case that Z is S_A is then assumed to be uniform. Thus, the `PDF` of Z is expressed as

$$f_Z(z) = \begin{cases} \dfrac{1-M_2}{S_A} = M_1, & 0 \le z < S_A \\ 1-\lambda/\left(\dfrac{S_A}{p}\cdot\eta\cdot\mu\right) = M_2, & z = S_A \end{cases}. \tag{3.12}$$

If a full-time server (no periodic tasks and no thermal constraints) handles aperiodic tasks, the mean response time can be easily obtained through an `M/M/1` queueing model. Because of periodic task execution and some idle duration, however, the actual service time is no longer an exponentially distributed variable. The actual service time is expressed differently according to the beginning moment of execution of an aperiodic job, as follows:

- $0 \le Z < S_A$,

$$X' = W + \left(\left\lfloor \frac{W-Z}{S_A} \right\rfloor_0 + 1\right)\cdot(p - S_A), \tag{3.13}$$

- $Z = S_A$,

$$X' = \begin{cases} W, & W \le Z \\ W - b + \left\{\left\lfloor \dfrac{W-Z}{S_A} \right\rfloor_0 + 1\right\}\cdot(p-S_A), & W > Z \end{cases}, \tag{3.14}$$

where X' is defined as the actual service time which means the delay between the

[4]In the development of this model we loosely follow an approach described in [51] for the simpler case of the Immediate Server.

beginning and the completion of the execution of the task. b is the beginning moment of job execution. When we define X as the execution time of an aperiodic job at the speed level of s_H without any interference from periodic tasks, we set $W = X/\eta$ as the delay at the average speed, s_{avg} through an `M/M/1` queueing model.

Equations (3.13) and (3.14) describe how to obtain the actual service time for the given aperiodic job depending on the beginning moment of the job's execution. This enables us to determine the average response time for aperiodic tasks using a `M/G/1` queueing model as explained in following section. In Equation (3.14), the variable for beginning moment of job execution b is also assumed to be uniformly distributed within its possible ranges. That is, $f_B(b) = 1/(p - S_A)$ when b is within $[0, p - S_A]$.

3. M/G/1 Queueing Model

As shown in the above section, since the actual service time is no longer exponentially distributed, an `M/G/1` queueing model is required to compute the mean response time R_m for aperiodic tasks. R_m is expressed as

$$R_m = E[X'] + \frac{\lambda E[X'^2]}{2(1 - \lambda E[X'])} \tag{3.15}$$

from [50].

From the PDFs of variables and Equations (3.13) and (3.14),

$$\begin{aligned} E[X'] &= \int_0^{S_A} \int_0^{\infty} X' f_X(x) f_Z(z) dx dz \\ &= \int_0^{S_A} \int_0^{\infty} X' \eta f_W(w) f_Z(z) dw dz \end{aligned} \tag{3.16}$$

and

$$E[X'^2] = \int_0^{S_A} \int_0^{\infty} X'^2 \eta f_W(w) f_Z(z) dw dz. \tag{3.17}$$

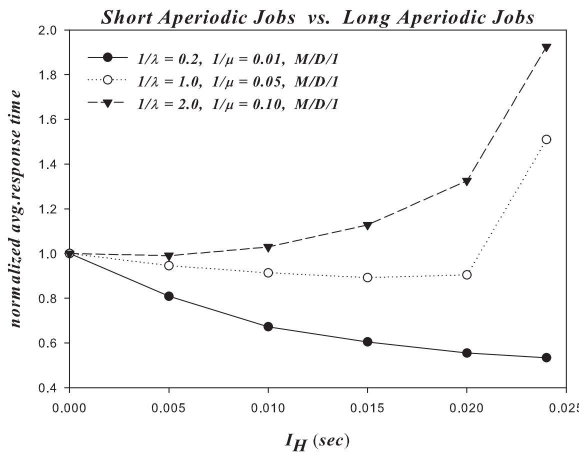


Fig. 11. Comparison of average response times of aperiodic jobs without periodic task

In equations (3.16) and (3.17), we use the following expansion for an integral,

$$\int_0^\infty \left\lfloor \frac{W-Z}{S_A} \right\rfloor_0 \cdot \eta f_W dw = \sum_{n=0}^{\infty} \int_{nS_A+z}^{(n+1)S_A+z} n \cdot \eta f_W dw, \tag{3.18}$$

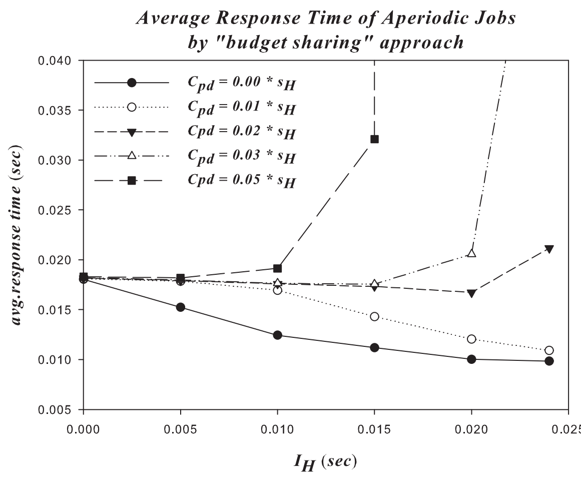
which is a similar approach described in [51].

E. Performance Evaluation

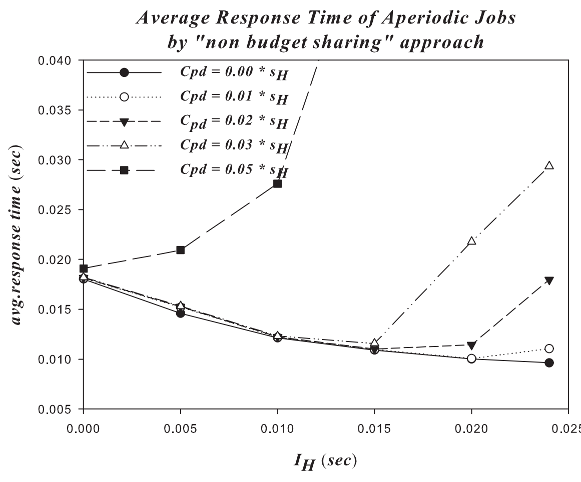
1. Simulations

In this section we evaluate the performance of the proposed Transient Overclocking Server for various aperiodic arrivals alone or in combination with various periodic workloads. We do this using a discrete-event simulation of a thermally-constrained CPU that runs a single periodic task (which represents the worst-case periodic workload) and a Poisson stream of incoming aperiodic jobs.

To evaluate the Transient Overclocking Server, we run it with different overclocking budgets and measure the response times experienced by the aperiodic jobs. In all experiments, we vary the overclocking budget B_H by varying the steady state temper-



(a) Budget-sharing approach



(b) Non-budget-sharing approach

Fig. 12. Average response time of aperiodic jobs with execution of periodic task: $p =$ 200 msec

ature T_0. This is represented by the value for I_H on the x-axis, which represents the time to reach the maximum temperature T_C from the steady-state temperature T_0. In all figures except for Figure 12(b), I_H is equal to L_H, the length of the overclocking budget.

The thermal parameters (thermal resistance R and thermal capacitance C) for this evaluation have been experimentally derived through thermal monitoring of a

Lenovo T43 notebook with an Intel Pentium M CPU running at various speeds up to 2.0GHz. Based on these parameters, we extrapolated the behavior of the same processor at a hypothetical overclocked speed of 3GHz. We use the performance at 1.6GHz as the equilibrium speed s_E and that at 3GHz as the overclocking speed s_H. We define the critical temperature to be 25^oC above ambient temperature.

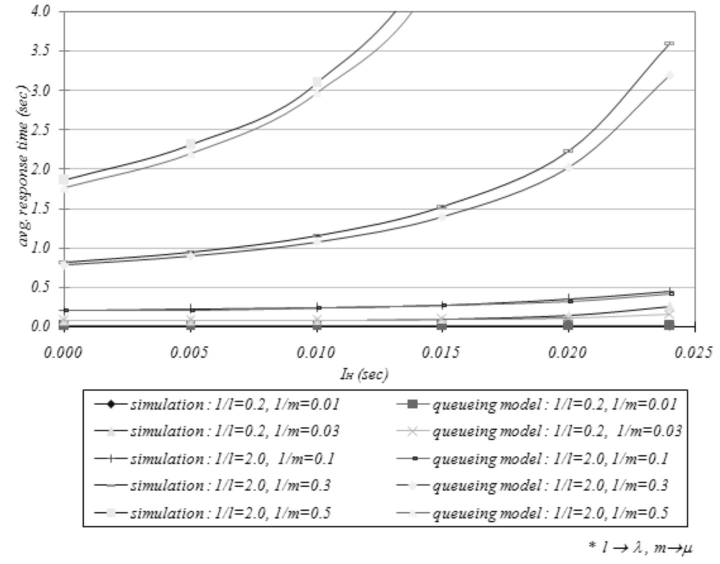
Figure 11 shows the effect of overclocking on aperiodic arrivals with different job lengths. It clearly shows that systems with short aperiodic jobs (10msec in our experiments) benefit from overclocking. In fact, the more aggressively we overclock (I_H large,) the better are the response times. This is easy to explain: for large overclocking budgets (L_H of 10msec and above) the aperiodic jobs can be executed entirely within the overclocking budget, and therefore fully benefits from the increased CPU speed. Long aperiodic jobs (50msec and 100msec in our experiments) show only negligible reduction in response times, or have worse response times. Since these jobs do not fit in the overclocking budget, and fit increasingly less into the non-overclocked portion of the budget the more aggressively we overclock, they get penalized by the reduced total execution time they have available during each period, and therefore take more than one period to complete.

Figure 12 shows the effect of varying amounts of periodic workload on the response times of the aperiodic jobs. In both graphs we use the "short" aperiodic jobs from Figure 11 and add increasing amounts of periodic workload, starting from zero, to 50msec execution time at overclocking speed s_H. In both graphs we see how large amounts of periodic workloads eventually "crowd out" the aperiodic workload from the overclocking budget, and cause an increase in response time. The graphs also show that the non-budget-sharing approach better protects the aperiodic jobs by forcing the periodic jobs to run at equilibrium speed. The result is lower response times in the non-sharing case.

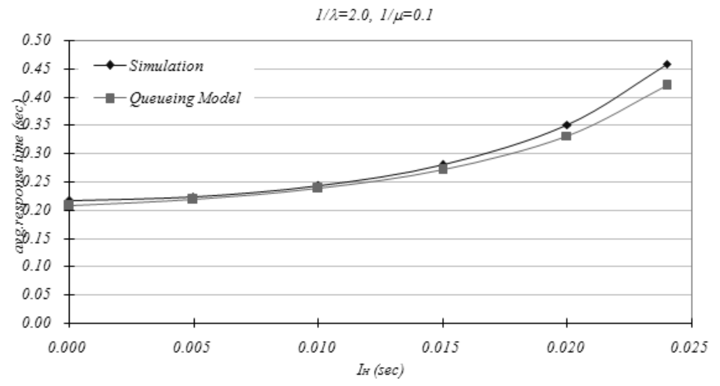
In the case of very aggressive overclocking (I_H of 20msec and higher) with high levels of periodic workload (20msec execution time at s_H and higher) a weakness of the non-sharing approach becomes apparent: Since the executing time of the periodics must be treated separately from the aperiodics, there is not sufficient slack left to allocate a sufficiently large overclocking budget. In such cases, increasing I_H actually reduces the overclocking budget (in such cases $L_H^{(1)} < L_H^{(2)}$), and therefore increases the response times.

2. Queueing Model Validation

We validated the queueing model against the discrete event simulations. Figure 13 and Figure 14 compare average response times obtained by the queueing model and by simulations. As a simple way to validate our queueing model, we compare the values at $I_H = 0$. Having zero length of I_H and no workload from periodic tasks means that a CPU runs always at *equilibrium speed* s_E and the transient overclocking server is able to run whenever there are aperiodic tasks to execute without any interruption of other tasks. We can take this way of working as a general `M/M/1` at service rate of $\mu \cdot \frac{s_E}{s_H}$. We can see that analytical and simulation results match the response time by simple `M/M/1` model at $I_H = 0$. In Figure 13, we see that the results match each other well with very small errors. With larger amounts of periodic task workload, the effect of increasing the length of I_H becomes severe, leading to large increases in response time. From the comparison in Figure 14, we can also notice that the effect by increase of I_H is smaller than the simulations. Since we averaged two levels of processor speed into s_{avg}, the response time appears to be less sensitive to the change of I_H relatively than simulations.



(a)



(b)

Fig. 13. Comparison of average response times between simulation and queueing model without periodic task

F. Conclusion

By their very nature, embedded real-time systems are exposed to *environmental effects*, some of which can influence the ability of the system to provide the promised performance guarantee levels. For example, atmospheric conditions may affect the ability of free-space optical links to carry traffic (e.g., [52]), or electromagnetic interference may prevent 801.11-style networks to provide QoS guarantees [53]. Similarly, service can be disrupted due to attacks by third-parties, software errors, changes in security posture and other maintenance requirements, flash crowds, thermal influences, and many other reasons. Many of these environmental effects are outside of the control of the system designer and operator[5].

In contrast, other environmental effects that are triggered by system- or program behavior, and for which therefore *predictive models* exist. Examples of this type of environmental effects are: Thermal overload due to power dissipation, memory availability in garbage-collected memory systems [54, 55], or energy availability in battery-powered systems [56].

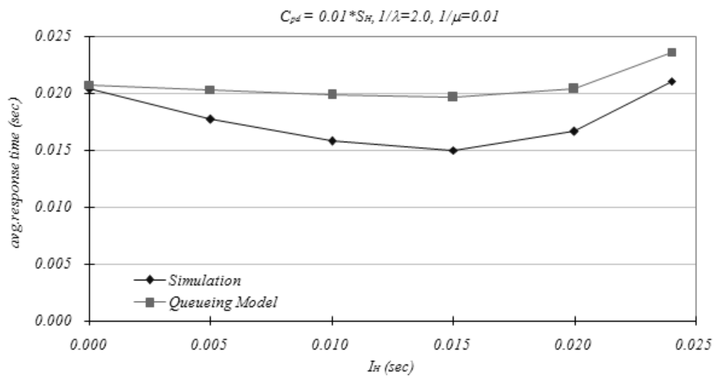
Such effects give rise to interesting *priority inversions* between tasks, where access to the computational resource by one fully preemptible tasks may cause a higher-priority task to miss its deadline in the future because of speed control. This interaction is particularly interesting when it comes to handling of (supposedly low-priority) aperiodic jobs. In this chapter we illustrate that traditional algorithms (such as Slack Stealing and Deferrable Server) cannot be naïvely applied in thermally constrained environments, as this can lead to missed deadlines. We then proceed to describe a design-time scheme to allocate budget to accommodate aperiodic job arrivals. Finally, we show how transient overclocking with the budget computed offline at design time reduces response times for aperiodic jobs with a series of discrete-event simulations.

Many issues remain open after this work. For this paper we set out to define a design-time approach to budget allocation for the Transient Overclocking Server under thermal constraints. The benefit of using transient overclocking have been illustrated

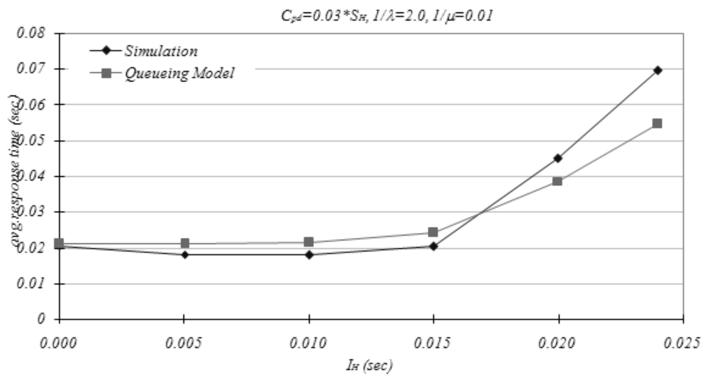
[5]More precisely, we mean to say that the occurrence of events cannot be controlled by the operator or designer; the frequency of occurrences may be in some cases controlled: for example, one can reduce the rate of software faults by better testing.

through experiments. Currently we have no tools available that support the engineer's decision on whether and how to apply transient overclocking. Analytical tools and models must be developed that allow to optimize the budget allocation for given settings.

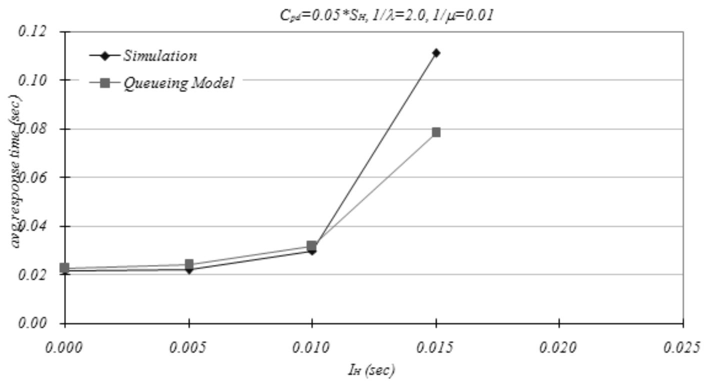
In general, the type of priority inversions caused by the use of resources that later need to be reclaimed (either through speed control, or garbage collection, or energy-aware DVS, or others) needs to be better understood, and appropriate resource access protocols need to be devised.



(a)



(b)



(c)

Fig. 14. Comparison of average response times between simulation and queueing model with periodic task execution

CHAPTER IV

ON-LINE THERMALLY-AWARE TRANSIENT OVERCLOCKING FOR MIXED-WORKLOADS

A. Introduction

In the previous chapter we described the design-time budget based transient overclocking method for mixed-workloads. The budget based approach guarantees thermally safe execution of aperiodic jobs, and the approach is very efficient during run-time with low overhead in the budget management and no need for on-line temperature monitoring. Since this speed control scheme effectively runs in open-loop mode it can, however, be too conservative in minimizing the response times of aperiodic workload. This follows because the scheme computes budgets based on the assumption that the system will be fully utilized at all times with worst-case periodic workload. In addition it must assume that aperiodic jobs are always backlogged in the task queue. In practical systems, however, the workload is often significantly lower, and the temperature level is usually lower than the worst-case thermal levels as well.

In this chapter we propose an *on-line thermally-aware transient overclocking* method to reduce the response times of aperiodic jobs efficiently at run-time. We describe a modified Slack-Stealing algorithm to consider the thermal constraints of systems together with the deadline constraints of periodic tasks. With the thermal model and temperature data provided by embedded thermal sensors, we compute slack for aperiodic workload (*slack estimation*) at run-time that satisfies both thermal and temporal constraints. The slack estimation is based on the system thermal model, thermal management strategy, and the microprocessor's temperature monitored by thermal sensors embedded. For the speed control policy, we consider inter-task speed control scheme which adjusts the speed task by task. We also integrate the simple *Reactive Speed Scaling* (RSS) as the thermal management strategy in which the speed is switched to the equilibrium speed s_E when temperature increases and reaches the critical temperature T_c.

We will see in this chapter that the proposed *Thermally-Aware Slack-Stealing* (TASS) algorithm minimizes the response times of aperiodic jobs accounting for the lower temperature and the early completion of periodic tasks and it satisfies both the thermal and the deadline constraints. Furthermore, the algorithm has low computational complexity with $O(1)$ on-line computation for slack estimation per aperiodic task arrival.

B. System Model, Assumptions and Notation

1. Assumptions and Notations

Throughout this chapter, we adopt preemptive Earliest-Deadline-First (EDF) [8] scheduling policy. We assume that the computational workloads has two components: A periodic task set with hard deadlines, and a set of soft deadline aperiodic jobs. We follow standard practice and assume that the worst-case execution times and the arrivals of periodic tasks are known a priori, and the execution times of soft aperiodic jobs become known upon job arrival. We first propose a static solution for aperiodic job executions, assuming that the execution times of the instances of all periodic tasks are equal to their worst-case execution times at all times, thus maximizing the periodic tasks' utilization. In practice, task instances often complete well before their worst-case execution time. In such cases, service to aperiodic workload can be increased or power consumption decreased by accounting for the unused scheduled processor cycles. This is generally called dynamic slack reclamation [25, 57]. We will

Table II. Notations for Task Model

Symbol	Meaning
U_t	The total utilization of periodic task set at s_E, that is, $U_t = \sum_{i=1}^{n} \frac{e_i}{p_i}$
$U_e(s)$	The effective utilization of periodic task set at speed s, $U_e(s) = U_t \cdot \frac{s_E}{s}$
s_N	Nominal CPU speed (which makes the *effective utilization* of the system be equal to 1)
J_i	The i^{th} aperiodic job
t_i^r	The arrival time of J_i
t_i^f	The finish time of J_i
NTA	The earliest next arrival time of a periodic task instance in the system after time t, which is equal to the next earliest deadline if $p_i = D_i, \forall \Gamma_i$
$c_{[t,NTA]}$	The maximum amount of clock-cycles available from the current moment t until NTA by RSS
c^{pd}	The amount of periodic processing clock-cycles to execute until NTA
c^{ap}	The amount of aperiodic processing clock-cycles available until NTA

discuss the role of dynamic slack reclamation in the context of thermally constraint systems and present a thermally-aware dynamic slack reclamation technique. As in the previous chapter, all periodic tasks are assumed to be independent and in-phase. The deadline of each periodic task instance is assumed to be at the end of the task period.

Recall that the combination of RSS speed control and inter-task speed assignment allows the system to adjust the processor speed at the following points: (a) at task boundaries, (b) at each task arrival/finish instant, and (c) at the moment when the critical temperature is reached.

In addition to the terms and notation defined in Chapter II, in the following we will make use of the notation described in Table II.

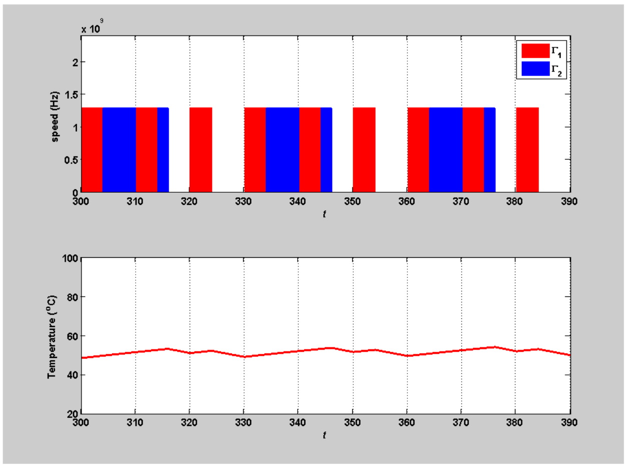
2. Periodic Hard Real-Time Task Execution

We consider a set of hard real-time periodic tasks $\Gamma = \{\Gamma_1, \Gamma_2, \ldots, \Gamma_n\}$, where each task $\Gamma_i = (p_i, e_i)$ has a minimum time p_i between job instances. Each job requires

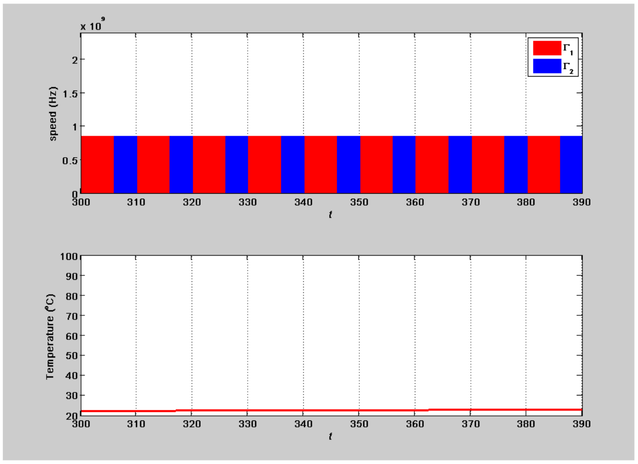
e_i execution time at equilibrium speed s_E to complete in the worst-case. We adopt the Earliest-Deadline-First (EDF) scheduling scheme to schedule the execution of the periodic tasks. Given a set Γ of periodic tasks, we define the total utilization of Γ at *equilibrium-speed* as $U_t = \sum_{i=1}^{n} \frac{e_i}{p_i}$. It is well known that Γ can be feasibly scheduled by EDF if and only if $U_t \leq 1.0$. We assume that the total utilization of the given periodic task set is smaller or equal to 1.0, and so the task set is schedulable and thermally safe at speed s_E. Figure 15(a) shows an example of an EDF schedule and the resulting temperature variation for a given task set $\Gamma = \{(10, 4), (30, 8)\}$ at the constant equilibrium speed s_E. We also define the *effective utilization* (U_e) of the periodic task set at speed s as follows:

$$U_e(s) = \sum_{i=1}^{n} \frac{c_i}{s \cdot p_i} = \sum_{i=1}^{n} \frac{s_E \cdot e_i}{s \cdot p_i} = U_t \cdot \frac{s_E}{s} \ . \tag{4.1}$$

When there is no aperiodic job to consider for scheduling, we decide to use the speed which makes $U_e(\cdot) = 1$. We call this speed level the nominal speed and denote it by s_N. We know that $s_N = U_t \cdot s_E$, and that when $U_e(s_N) = 1$, the task set Γ is schedulable at speed s_N by EDF. Figure 15(b) shows the EDF scheduling of the given periodic task set Γ at the constant nominal speed s_N. The advantages of the use of nominal speed s_N for periodic task set can be described in three respects: 1) As research in energy-aware DVS schemes shows, the energy consumption is minimized when a constant lowest-possible speed is selected for task executions [58, 59]. 2) Lower speeds reduce the peak temperature during job executions, thus delaying the triggering of the RSS thermal management mechanism. Finally, 3) When the speed s_N at which $U_e = 1$ is selected for periodic task execution, it facilitates the run-time slack estimation since the amount of slack in the periodic workload is easily computed without any reference to precomputed slack time tables.



(a) Scheduling with s_E



(b) Scheduling with s_N

Fig. 15. EDF scheduling with constant speed: $\Gamma = \{(10, 2), (30, 8)\}$

3. Aperiodic Task Execution

In systems with both periodic and aperiodic workloads, the latter must be scheduled so as to minimize their response times while at the same time not causing deadline violations for the periodic tasks.

A popular such scheduling algorithm is the *Slack Stealing* scheduler [9]. Slack stealing executes aperiodic tasks by using the available slack times of periodic tasks.

If there is available slack in the periodic tasks, aperiodic jobs can be serviced first without violating deadlines of periodic tasks as long as slack is available. The slack times are computed on-line by the slack stealer with all the information about the periodic task executions (e.g., periods, deadlines, remaining execution times, and etc.). We call this computation *slack estimation*. It can be shown that slack stealing minimizes the response times of aperiodic jobs [9].

In the following we base our thermally-aware scheduling on the Slack-Stealing algorithm described in [9]. The slack stealer is particularly appropriate for aperiodic job executions in our case because the algorithm can fully utilize the temporal resources made available by the on-line slack estimation, which in turn minimizes the response times of aperiodic jobs. In systems that provide multiple speeds, transient overclocking can be applied to further reduce the response times of aperiodic jobs. By temporarily increasing the microprocessor speed at times, more slack can be retrieved and more clock-cycles during the slack can be allocated to aperiodic jobs.

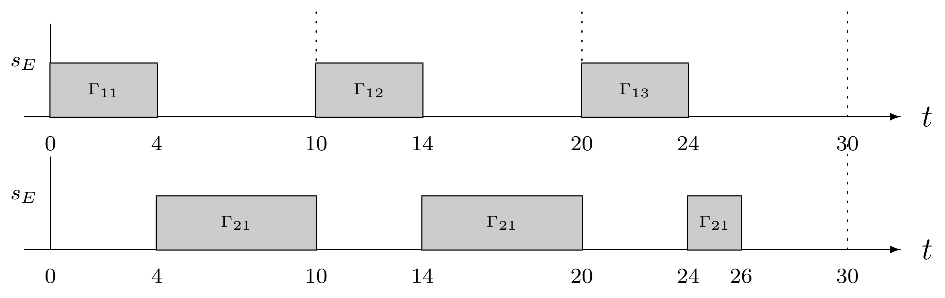
While this form of transient overclocking is very efficient in the minimization of the response times of aperiodic jobs, we need to take into account the thermal safety of systems. That is, at any time during overclocking we need to keep track of the current temperature in order to predict how much longer we can keep overclocking. We call this the thermal slack in the system. As a result, in a thermally aware slack scheduling we need to keep track temporal slack and thermal slack, and - if needed - trade them off against each other.

Thus, when the system is thermally constrained, three issues arise in the application of transient overclocking scheme:

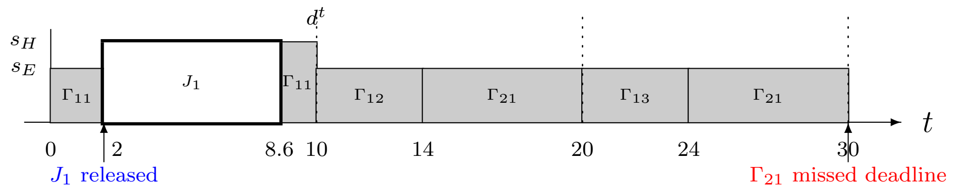
- How fast and how long can the processor execute aperiodic jobs before thermal constraints are exceeded ?

- How to compute the *thermal slack* ?

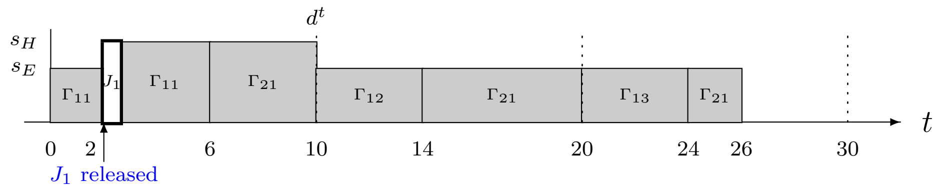
We propose the following *thermally-aware slack-stealing* algorithm. It is based on the traditional slack-stealing algorithm and transient overclocking in a thermally aware fashion.



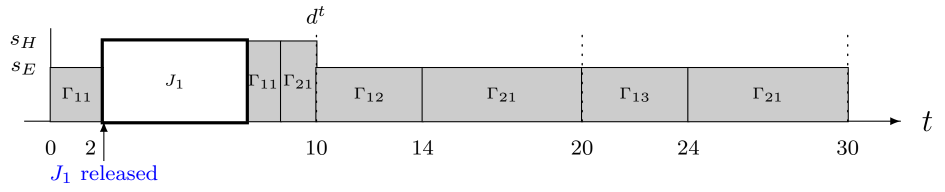
(a) EDF scheduling of periodic tasks



(b) Deadline miss by erroneous slack computation



(c) Conservative slack estimation: $c^{pd} = s_E \times (d^t - t_1^r) = s_E \times 8.0$



(d) Correct slack estimation

Fig. 16. Illustrations of slack computation: $\Gamma = \{(10, 4), (30, 14)\}$

Before providing the details of our approach, we show the necessity of using s_N

for the slack estimation. The slack stealing algorithm estimates slack times using a procrastination scheduling approach, where periodic task execution is delayed until any slack is exhausted to maximize the duration of idle intervals. It therefore requires the information of periodic task deadlines and the execution times of the tasks. The conventional slack stealing algorithm creates a precomputed slack table and dynamically updates the table to find appropriate slack time at run-time. In systems with dynamic speed control, however, it would be very inefficient for the on-line slack estimation due to the computational overhead if we keep a precomputed slack table or maintain the task execution history on the system as in the traditional way. In this study, we propose an efficient on-line method to estimate slack in systems with dynamic speed control. For the online slack estimation, we first need to set the instantaneous deadline d^t by which the completion of specific amount of periodic workload can be delayed without violating the deadline constraints of periodic tasks. In the proposed method, we use the next task arrival *NTA* as the instantaneous deadline for the slack estimation. With the instantaneous deadline d^t set to NTA, we then estimate the periodic workload amount of which we can delay the completion until d^t. When we correctly compute the periodic workload amount to complete by d^t, we are able to estimate the slack for aperiodic workload. However, the correct computation of periodic workload to procrastinate requires the information of all the instances of given periodic tasks and slack computation for each task instance, which will not be appropriate for the on-line slack stealing under the system with dynamic speed control. Instead, we decide to use the single time information of the instantaneous deadline d^t which is set to NTA. In addition, we introduce the use of the speed s_N at which $U_e(\cdot) = 1$ for the on-line efficiency in slack estimation. By the use of the speed s_N, it will be shown that we can simplify the computation of how long to delay the execution of periodic workload to benefit aperiodic job execution without having to explicitly manage the slack table and the history of periodic task executions.

In the following after we describe the difficulty of correct slack estimation only with the information of the next task arrival (NTA) by some scheduling examples, we explain how the speed s_N can be used to simplify the slack computation. We will then illustrate why we need the thermal-awareness in the application of transient overclocking scheme for the mixed-workload execution. We initially simplify the discussion by not considering thermal constraints. Rather, we assume the aperiodic workload to be executed at the highest speed s_H (*transient overclocking*). To maximize the available slack times, we also assume that CPU runs at s_H for the periodic workload execution between the release moment of aperiodic job t^r and the instantaneous deadline d^t.

In the proposed slack estimation method, when we do not take into thermal-safety, the slack (in clock-cycles) can be simply computed at t^r as follows:

$$slack = c_{[t^r,d^t]} - c^{pd}, \tag{4.2}$$

where $c_{[t^r,d^t]}$ is the total number of clock cycles available at the speed s_H during the time interval $[t^r, d^t]$, that is, $c_{[t^r,d^t]} = s_H \times (d^t - t^r)$. Recall that c^{pd} denotes the estimated periodic workload that needs to complete within the interval $(t^r, d^t]$. As mentioned above, however, the correct computation of c^{pd} is not trivial. First, it is difficult to find on-line the proper set of periodic task instances to estimate slacks. Second, the overhead of on-line slack computation for all task instances must be very high.

In the following we illustrate the scheduling of newly arriving aperiodic tasks until NTA to point out the issue of periodic workload estimation for the slack computation. We use a task set with two tasks Γ_1 and Γ_2 :

$$p_1 = 10, e_1 = 4, p_2 = 30, e_2 = 14.$$

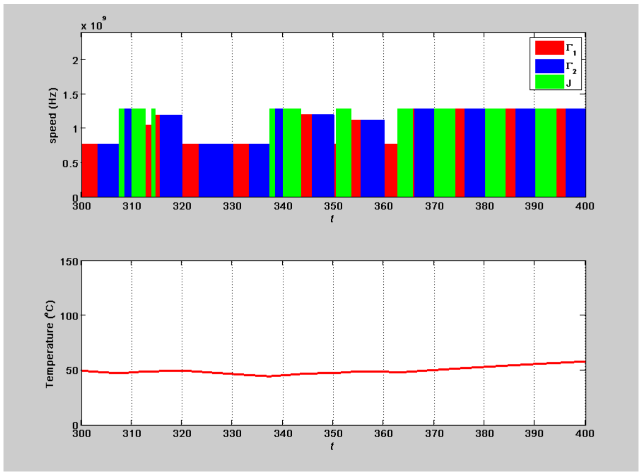
Figure 16(a) shows an EDF schedule of the periodic tasks. Suppose an aperiodic job becomes ready at $t = 2$ at which point d^t is set to NTA ($t = 10$). For the estimation of the periodic workload that can be delayed until d^t, we can think of using the information about the periodic task instances currently in the EDF queue. The naive use of task information in the take queue, however, results in deadline errors of periodic tasks. Figure 16(b) shows the erroneous task schedule. In Figure 16(b), when we take only the backlogged workload of $\Gamma_{1,1}$ for the periodic workload c^{pd} estimation and do the slack estimation at $t = 2$, $\Gamma_{2,1}$ would not have enough processing time to complete and miss the deadline at $t = 30$. If, on the other hand, we consider both $\Gamma_{1,1}$ and $\Gamma_{2,1}$, which is backlogged in the EDF queue and regard them as the c^{pd} for slack estimation, we find no slack available until d^t. In fact, the workload sum of $\Gamma_{1,1}$ and $\Gamma_{2,1}$ is not able to be finished in the interval $(2, 10]$ even at the highest CPU speed s_H if s_H is assumed as $s_H = 1.5 \cdot s_E$ in this example. One possible solution for the c^{pd} estimation is, therefore, to simply multiply s_E with the time interval between the aperiodic job release moment t^r and NTA. Although this way of periodic workload estimation and the slack computation guarantees all periodic task deadlines, it does poorly at reducing the response time of aperiodic workload. In fact, it significantly overestimates the periodic workload to execute during the interval. In Figure 16(c), we notice that very small amount of slack is retrieved at time $t = 2$ for aperiodic task by estimating $c^{pd} = s_E \cdot (NTA - t^r) = s_E \times 8.0$, which leaves CPU idle during the time interval $(26, 30]$. By the overestimation of periodic workload during the interval $(t^r, NTA]$, the slack stealing algorithm becomes very conservative. On the contrary, Figure 16(d) shows the correct slack estimation for the aperiodic job execution, which leaves no idle time until the end of hyperperiod of Γ_1 and Γ_2 at $t = 30$. By estimating the accurate minimum workload amount of $\Gamma_{2,1}$ to complete until NTAs which satisfies the deadline constraint of Γ_2 in the example, we can retrieve the maximum amount of slack for aperiodic workload at the moment $t = 2$.

However, the accurate estimation of the minimum periodic workload amount to complete until NTA is not trivial in the case that the periodic task set is composed of large number of periodic tasks. It should require a bookkeeping of periodic task executions with a large memory space and high run-time computational overhead which is not appropriate for on-line slack estimation. Furthermore, when the thermal-awareness and the speed control scheme for thermal management is concerned, the slack estimation becomes more complicated.

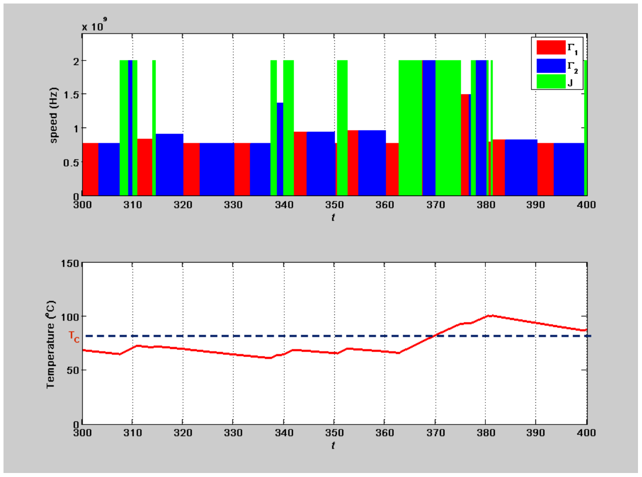
Therefore we apply the nominal speed, s_N, to make the efficient and accurate run-time estimation of periodic workload to execute for a specific time interval. By the application of s_N at which $U_e(\cdot) = 1$, the periodic task set is always schedulable under the EDF scheduling algorithm and the periodic workload to complete during a specific interval $(t_1, t_2]$ is computed simply as $c^{pd} = s_N \times (t_2 - t_1)$. In following sections, we will describe the slack estimation based on the nominal speed s_N in detail.

Lastly, we show the necessity of thermal-awareness for the transient overclocking scheme through another example of task scheduling. Figure 17 shows two different EDF scheduling with aperiodic jobs with the use of s_E or s_H for aperiodic jobs and s_N for periodic workload. The speed assignment to periodic workload is assumed to follow the conventional energy-aware constant speed algorithm by which the lowest-possible constant speed is applied for given workload.

In the examples, the slack is computed simply by looking at every interval length from the moment when an aperiodic job is released to an otherwise idle aperiodic job server to NTA and supposing the CPU runs either at s_E or s_H during the interval. In Figure 17(a) in which aperiodic workload runs at speed s_E, the temperature never reaches to the critical level T_c, it however delays the response time of aperiodic jobs. When aperiodic jobs run at speed s_H (Figure 17(b)), the response time is significantly



(a) Scheduling with s_E



(b) Scheduling with s_H

Fig. 17. EDF scheduling with aperiodic tasks: $\Gamma = \{(10, 2), (20, 8)\}$, $\lambda = 0.1, \mu = 0.5$

reduced, but the thermal constraint cannot be satisfied. (In the example, we set $T_c = 80^oC$.)

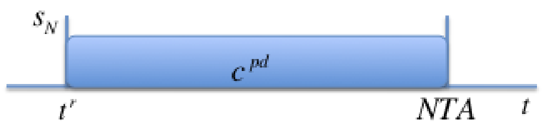
In summary theses examples illustrate that the speed assignment for the execution of mixed workload must consider current thermal condition. Otherwise, it either violates the microprocessor's thermal constraint or unduly delays the response times of aperiodic jobs. In the next section, we propose how to compute the slack for aperiodic jobs with the application of *RSS* thermal management mechanism and how to control the processor speed to minimize the response times of aperiodic jobs satisfying both the deadline constraints of periodic tasks and the thermal constraints of microprocessor.

C. Aperiodic Task Server with Transient Overclocking

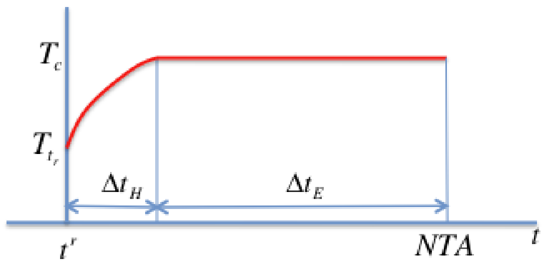
As we showed in the previous section, any safe and effective slack computation method must consider both the mechanism for thermal management and the temperature variation. Although the use of the thermally safe constant speed s_E guarantees both the deadlines of periodic tasks and the thermal constraints, it is too conservative to sufficiently reduce the response times of aperiodic jobs. We thus propose a thermally-aware slack-stealing (*TASS*) algorithm to minimize the response times of aperiodic jobs by fully utilizing the available slack without threatening the thermal safety of systems.

1. Slack Stealing Server under Thermal Constraints

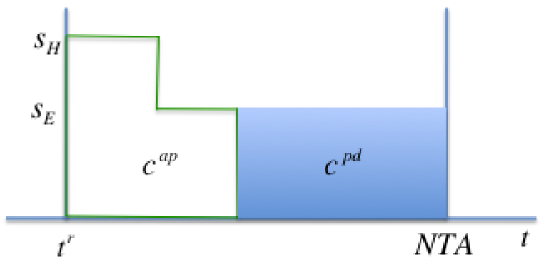
While the traditional slack stealing algorithm takes into account the slack in the temporal dimension, any thermally aware slack-stealing algorithm must take into consideration the thermal domain as well. [e.g. "One could informally say that such an algorithm should account and trade-off timing slack and thermal slack."] A slack computation algorithm therefore requires a specific thermal model and the monitoring of current temperature in addition to the available time interval to compute the available slack. To better represent the temporal and the thermal dimensions, in the following we will use the *clock-cycles* as the unit for the thermal slack. This is in contrast to traditional slack computation algorithms, which account for execution



(a) Periodic workload estimation



(b) Temperature prediction



(c) Thermal slack estimation

Fig. 18. Basic idea of thermally-aware slack-stealing algorithm

time only. For the thermal model, we select the simple model represented in Equation 3.1. The thermal slack estimation is based on the projection of temperature variations. First, the TASS algorithm predicts the temperature increase between the current moment and NTA which is set as the instantaneous deadline. For the thermal prediction, it considers the transient overclocking and the RSS speed control scheme as a thermal management. It also assumes that the system is fully utilized during the interval, that is, the CPU is continuously running in the interval. The TASS algorithm then computes the total available clock-cycles $c_{[t^r,NTA]}$ for the interval with the reference to the projected thermal profile by which the high speed duration Δt_H and the equilibrium speed duration Δt_E are obtained. Since the periodic workload to complete within the interval, c^{pd}, is estimated by the speed s_N and the length of the interval, we know the available slack amount by a simple arithmetic.

Figure 18 describes the basic idea how to find the thermal slack for aperiodic job execution. At the release instant of aperiodic job, the system monitors the current temperature T_{t^r} at t^r and find the moment of *NTA* (Figure 18(a)). With the temperature T_{t^r}, the time interval until *NTA*, and the thermal model, we then project the thermal profile of *RSS* (Figure 18(b)). Based on the projected thermal profile, we compute the total available clock-cycles $c_{[t^r,NTA]}$ in the interval under *RSS* scheme with the length of high speed execution Δt_H and the length of equilibrium speed execution Δt_E. By the simple subtraction of c^{pd} which is estimated in Figure 18(a) from $c_{[t^r,NTA]}$, we find the thermal slack c^{ap} (Figure 18(c)).

2. Thermal Slack Computation

Let $T_{ss}(s)$ denote the thermally steady-state temperature converged when the processor runs continuously for infinite time at a constant speed s. The time interval Δt_H in Figure 18(b) after which the temperature reaches T_c at speed s_H can be computed by Equation 3.1 and is expressed as follows:

$$\begin{aligned} T_{ss}(s_H) &= R \cdot \kappa s_H^{\alpha} \ , \\ \Delta t_H &= \frac{1}{b} \ln \left(\frac{T_{ss}(s_H) - T_r}{T_{ss}(s_H) - T_c} \right), \end{aligned} \tag{4.3}$$

where T_r is the monitored temperature at the moment of thermal prediction step (t^r in Figure 18). The value for Δt_H can be linearly approximated to lower the overhead of run-time computation as follows;

$$\Delta\tilde{t}_H = \frac{1}{b}\left(\frac{T_c - T_r}{T_{ss} - T_r}\right). \tag{4.4}$$

Observation 1. *If we denote by $\Delta\tilde{t}_H$ the value of Δt_H approximated by Eq. (4.4), then $\Delta\tilde{t}_H < \Delta t_H$. The Thermally-Aware Slack-Stealing (TASS) algorithm guarantees thermal safety of systems using the approximate value of Δt_H because it overestimate the temperature increase while aperiodic job executes and less amount of clock-cyles are assigned as slack to aperiodic tasks by the shorter length of maximum speed execution.*

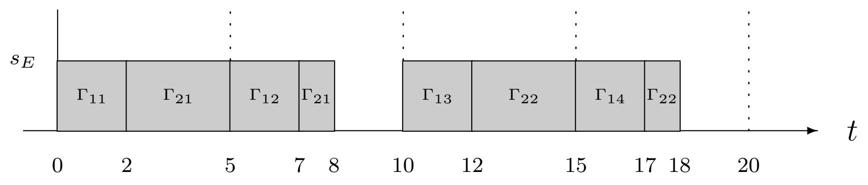
To simplify the run-time slack estimation, we decide to use $\Delta\tilde{t}_H$, the approximate value of Δt_H, which is thermally safe according to the Observation 1. In the remaining equations of this chapter, we use Δt_H for $\Delta\tilde{t}_H$ unless there is a need to distinguish those two terms. With the time length of the high speed duration Δt_H, the total available clock-cycles $c_{[t^r,NTA]}$ for the interval is computed according to the interval length between t^r and NTA as follows,

$$c_{t^r,NTA} = \begin{cases} s_H \cdot \Delta t_H + s_E \cdot (NTA - t^r - \Delta t_H) & \text{if } NTA > (t^r + \Delta t_H), \\ s_H \cdot (NTA - t^r - \Delta t_H) & \text{otherwise.} \end{cases}$$

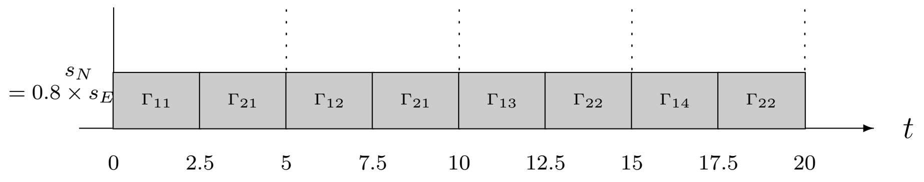
Since we use the speed s_N at which $U_e(\cdot) = 1$ for the periodic task execution while the TASS server is idle and the EDF schedule is applied, we guarantee that there is no deadline violation of periodic tasks. Considering the EDF scheduling with the nominal speed s_N, we also know that the least amount of periodic task workload to execute until NTA is $c^{pd} = s_N \cdot (NTA - t^r)$ as seen in Figure 18(a). The largest amount of aperiodic processing clock-cycles available until NTA, c^{ap}, is then simply obtained as follows,

$$c^{ap} = c_{t^r,NTA} - c^{pd}. \tag{4.5}$$

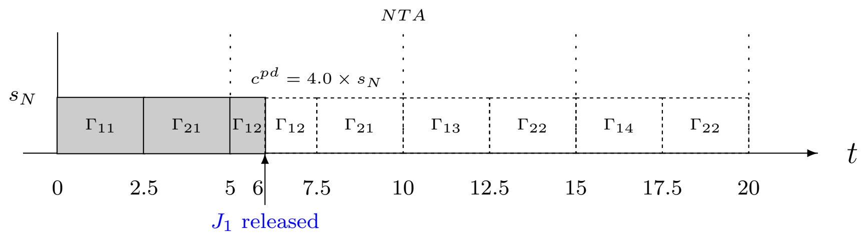
In the next section, we describe how the proposed TASS algorithm works for the mixed workload exectutions.



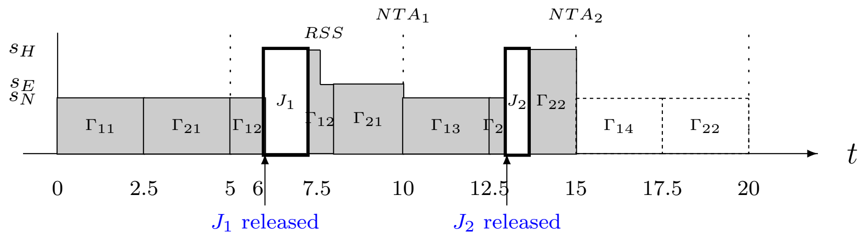
(a) EDF scheduling of periodic tasks at s_E



(b) EDF scheduling of periodic tasks at s_N



(c) An aperiodic job arrives at 6.0



(d) Speed control by TASS for aperiodic jobs

Fig. 19. The schedule of TASS algorithm: $\Gamma = \{(5,2),(10,4)\}$

Algorithm 1 Thermally-Aware Slack Stealing(TASS): rules and algorithm

Input: Γ and J (Periodic task set and Aperiodic jobs)

```
1:  Compute $s_N$ by the total utilization, $U_t$, of periodic task set
2:  At every arrival of periodic task instance, update $NTA$
3:  begin
4:  TASS is initialized to be inactive
5:  while System is running do
6:      if $J$ is in task queue then
7:          if TASS is inactive then
8:              wake up TASS
9:              $c^{pd} = s_N \cdot (NTA - t_{now})$
10:             /* for dynamic slack reclamation */
11:             if $\hat{c}^{pd} < c^{pd}$ then
12:                 $c^{pd} = \hat{c}^{pd}$
13:             end if
14:         end if
15:         TASS projects thermal envelope until NTA by RSS and computes $c_{t^r,NTA}$
16:         $c^{ap} = c_{t^r,NTA} - c^{pd}$
17:         if $c^{ap} > 0$ then
18:             /*execute aperiodic job*/
19:             s=$s_H$
20:         else
21:             if $c^{pd} > 0$ then
22:                 /*execute periodic job*/
23:                 s=$s_H$
24:                 decrease $c^{pd}$ at speed rate $s$
25:                 /*$s$ may be switched to $s_E$ according to the thermal condition*/
26:             else
27:                 /*CPU is idle*/
28:                 s=0
29:             end if
30:         end if
31:     else if $\Gamma$ is in task queue then
32:         if TASS is active then
33:             s = $\frac{c^{pd}}{NTA - t_{now}}$
34:             if s > $s_E$ then
35:                 /*execute periodic job*/
36:                 s = $s_H$
37:             end if
38:             decrease $c^{pd}$ at speed rate $s$
39:             /*$s$ may be switched to $s_E$ according to the thermal condition*/
40:         else
41:             /*execute periodic job*/
42:             s = $s_N$
43:         end if
44:     else
45:         s = 0
46:     end if
47:     if $t_{now} \geq NTA$ then
48:         suspend TASS
49:         $c^{pd} = 0$
50:     end if
51:     /* Job selected executes at the assigned speed $s$. */
52:     /* Speed can be switched to $s_E$ by RSS when $T_c$ is reached. */

53: end while
```

3. Speed Control

For the speed control, we focus on each time interval composed of two adjacent NTAs. The algorithm of Thermally-Aware Slack Stealing (TASS) aperiodic job server is explained in Algorithm 1 and the example is shown in Figure 19. At the beginning of NTA interval, if there is no aperiodic job ready in task queue so that the TASS server remains idle, the processor is assigned s_N to execute periodic task instance as described in lines 41-43 of Algorithm 1. When an aperiodic job is released at t^r, the TASS aperiodic job server becomes active and it projects a thermal profile based on the monitored current temperature, the thermal model, and the RSS scheme (line 15). The TASS aperiodic job server is active until the next NTA is reached. If there remains thermal slack c^{ap}, it executes the aperiodic job according to RSS strategy (lines 17-19). And the remaining periodic task workload is also executed either by the RSS scheme continuously or at the lowest possible constant speed according to the the amount of workload (lines 21-24 and 32-39). When the aperiodic job completes at t^f without using up c^{ap} in the interval, times that were supposed to be consumed for aperiodic job execution are reclaimed to lower the CPU speed for the remaining periodic task execution. Suppose TASS server is active in the NTA interval and there is no more aperiodic job ready in the aperiodic task queue. The lowest possible constant to finish the remaining periodic workload c^{pd} for the remaining time interval $[t^f, NTA]$ is computed as $s_{pd} = \frac{c^{pd}}{NTA-t^f}$. If the speed is less than s_E, we can safely assign the speed s_{pd} for the periodic workload without any concern about the violation of thermal constraint. Otherwise, we should make the speed control simply follow RSS scheme for thermal safety (lines 32-39).

Figure 19 illustrates an example of TASS algorithm with the periodic task set $\Gamma = \{(5,2),(10,4)\}$. Since $U_t = 0.8$, the nominal speed s_N is set as $s_N = 0.8 \cdot s_E$.

With s_N, the effective utilization U_e becomes 1, and Figure 19(b) shows the EDF scheduling of the given task set at the speed s_N. At $t = 6$, we suppose an aperiodic job is released ($t^r = 6.0$), and TASS server becomes active to project thermal profile until NTA ($t = 10$) and computes $c^{pd}(= 4.0 \cdot s_N)$. In Figure 19(d), we notice how c^{ap} is estimated considering RSS thermal management mechanism. In Figure 19(d), we also see that another aperiodic job J_2 is released at $t = 13$ and serviced by TASS aperiodic job server. In this case, however, it shows that there may not be the speed switching to s_E by RSS scheme during the executions of aperiodic jobs and of the remaining periodic tasks until NTA depending on the thermal condition.

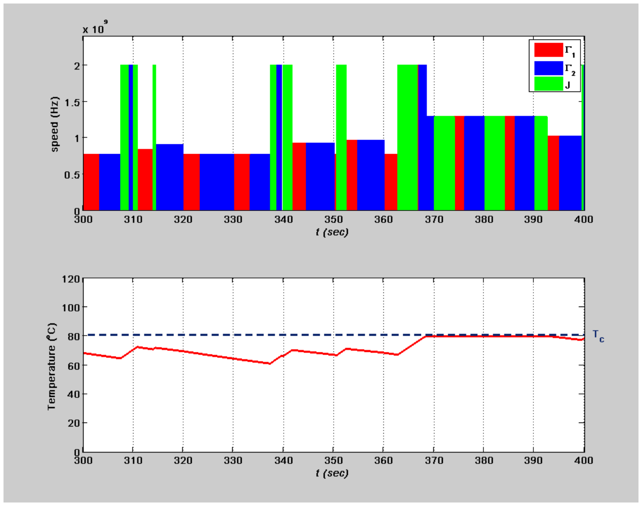


Fig. 20. Example of TASS speed control with temperature constraint $T_c = 80^oC$: $\Gamma = \{(10, 2), (20, 8)\}$, $\lambda = 0.1, \mu = 0.5$

Figure 20 shows an example of aperiodic job scheduling by the proposed thermally-aware speed control. In the example, the temperature never reaches over the critical level T_c, while the response time of aperiodic jobs are significantly reduced by transient overclocking.

Lemma 6. *Among all speed scaling scheme, the RSS minimizes the response times* *of aperiodic jobs, when two discrete speeds are used* [1].

Proof. Let ac^{ap} denote the actual workload amount of aperiodic job released at t^r. From the Figure 18, suppose $ac^{ap} \leq c^{ap}$, then we need to run the CPU at the highest speed possible to complete the aperiodic job at the earliest time. Thus, the RSS scheme is the best to minimize the response time of the job.

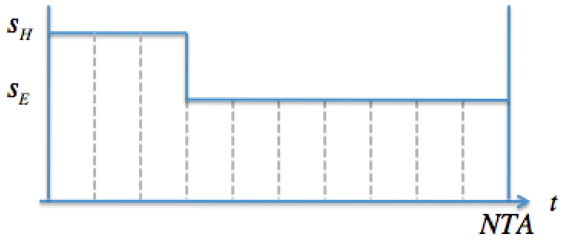
On the contrary, if $ac^{ap} > c^{ap}$, the aperiodic job cannot be finished by *NTA* and the available slack clock-cycles within $[t^r, NTA]$ should be maximized to minimize the remnant of the aperiodic job to execute in the following NTA intervals.

Let $\hat{c}$ denote the sum of the cycles by RSS in the interval. That is, $\hat{c} = c^{ap} + c^{pd}$. And let T_{RSS} be the temperature reached at *NTA* by RSS speed control. If there is any other speed assignment using the two speeds (s_H and s_E) that retrieves $\hat{c}$ clock-cycles and makes the final temperature T_{NTA} be lower than T_{RSS}, the speed control must be superior to the RSS scheme retrieving more available clock-cycles than $\hat{c}$ by the increase of high speed time length Δt_H, which increases T_{NTA} to be equal to T_{RSS}.

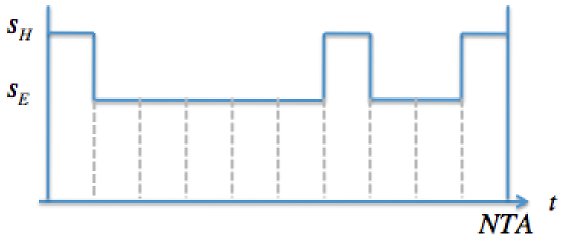
Assume that there is a way of speed assignment that is different from RSS scheme and it satisfies the condition described above. Since the workload amount by the speed assignment is assumed to be $\hat{c}$, total time lengths of s_H and s_E in the interval must be same as those of the RSS scheme, respectively. In this respect, the assumed speed assignment can be regarded as a different combination of speed assignment orderings from the RSS scheme, which is illustrated in Figure 21.

The comparisons of temperature increases by various orderings of speed assignment can be understood by the comparison described in the *Lemma* 1 in [21] with

[1] *1) We simplify the thermally-aware slack-stealing mechanism and alleviate the overhead of new speed computation by adopting two speed approach (RSS) for the aperiodic job execution. 2) We select two speeds s_H and s_E because the former is the fastest speed when system temperature is low and the latter is the thermally safe fastest possible speed when the temperature is at T_c.*



(a) RSS speed control



(b) non-RSS speed control

Fig. 21. RSS scheme vs. non-RSS scheme for same amount of $c_{[t^r,NTA]}$

a level of offset. It is because the temperature is an output of the power input in thermal-RC circuit model which is a linear-time-invariant system and can be superposed.

The *Lemma* 1 in [21] says that delaying some part of job execution increases temperature more at a specific later time instance. And this can be directly interpreted as that running the processor at high-speed earlier decreases the temperature lower at a moment after the execution of same amount of workload. For example, given a current temperature T_{t^r} and $T_{RSS}(= T_c)$, if we switch the speed assignment ordering from $< s_H, s_E >$ to $< s_E, s_H >$ for the same amount of cycles (the execution time length for s_E and s_H are Δt_E and Δt_H respectively) as shown in Figure 22, we should see the thermal violation at NTA, that is, $T_{NTA} > T_c$.

Based on the *Lemma*[21], we notice that there is no other speed assignment than RSS to lower the temperature below T_{RSS} at NTA obtaining $\hat{c}$ in the given interval.

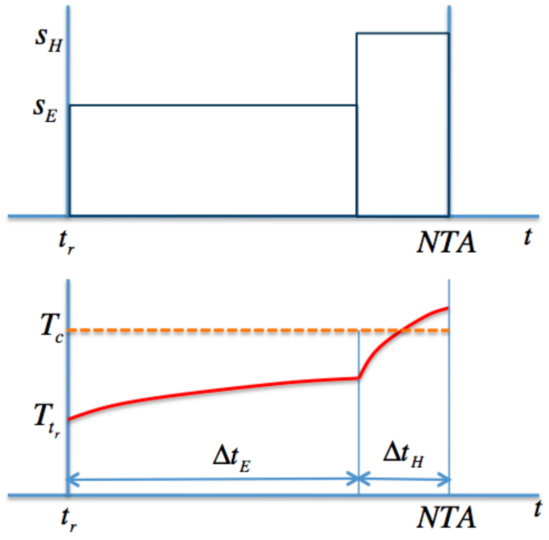


Fig. 22. Switch of speed assignment

This is the contradiction.

For both cases i) $ac^{ap} \leq c^{ap}$ and ii) $ac^{ap} < c^{ap}$, therefore, we see the RSS scheme is optimal for aperiodic job execution under thermal constraints. □

Lemma 7. *TASS algorithm guarantees both deadline constraints and the thermal constraint of thermally constrained hard real-time systems.*

Proof. For the deadline constraint to be satisfied in a hard real-time system with EDF scheduling strategy, periodic task workload assigned to every *NTA*-interval is required to complete within the interval. Since the periodic task workload is obtained from the EDF schedule with a constant nominal speed s_N that makes $U_e = 1$, the thermally-aware slack-stealing strategy that considers the workload amount guarantees the temporal feasible condition.

In every *NTA*-interval, if there is no aperiodic job ready and TASS is inactive, system temperature never increases to T_c because the proposed algorithm assigns s_N which is equal or smaller than s_E based on the assumption that the total utilization

U_t is equal or less than 1. When there is an aperiodic job released, *RSS* scheme is applied until the next *NTA* is reached or the aperiodic job is complete, whichever comes first.

For the execution of periodic workload while TASS server is active, too, unless the constant speed adjusted is larger than s_E, the system remains stable in thermal respective. For the case that the required constant speed for the remaining periodic workload until *NTA* is larger than s_E, the system is also thermally safe since the speed control follows *RSS*.

Therefore, both thermal constraint and deadline constraint are satisfied by proposed thermally-aware slack-stealing speed control scheme. □

Theorem 3. *TASS algorithm is optimal for minimizing the response times of aperiodic jobs in thermally constrained hard real-time systems.*

Proof. The task scheduling of mixed workload with TASS algorithm is feasible in the sense that periodics meet deadlines and thermal constraints are satisfied by the Lemma 7. And the Lemma 6 proves TASS minimizes the response times of aperiodic jobs. Thus, TASS is regarded as the optimal algorithm for aperiodic jobs in thermally constrained real-time systems. □

4. Dynamic Slack Reclamation

For thermal slack computation, we consider both the temperature level of microprocessor at the release moment of aperiodic job and the remaining periodic workload to execute.

The slack computation described in section 2 is static because it considers only the worst-case periodic workload. It is known that, in many cases, the instances of real-time tasks complete earlier than the worst-case scenario. The earlier completion of the instances of real-time tasks contributes to more slack achievement in two ways. Less workload execution decreases temperature level after it is finished [21]. This lower temperature eventually delays the moment of hitting the critical temperature T_c and increases the length of high speed duration Δt_H in the equation (4.3). Furthermore, the early completion of periodic jobs means less remaining amount of periodic workload than the worst-case scenario. Let the sum of the remaining worst-case periodic workload be denoted as $\hat{c}^{pd}(t^r)$ at the release instant t^r of aperiodic task. Since $\hat{c}^{pd}(t^r)$ is typically less than c^{pd} by early completions of jobs in the NTA interval, we can safely increase the thermal slack by replacing c^{pd} with $\hat{c}^{pd}(t^r)$ in the equation (4.5) without violating the deadline constraints of systems. By the increase of Δt_H due to the lowered temperature level and by the decrease of remaining periodic workload to complete until NTA, the slack c^{ap} for aperiodic workload increases.

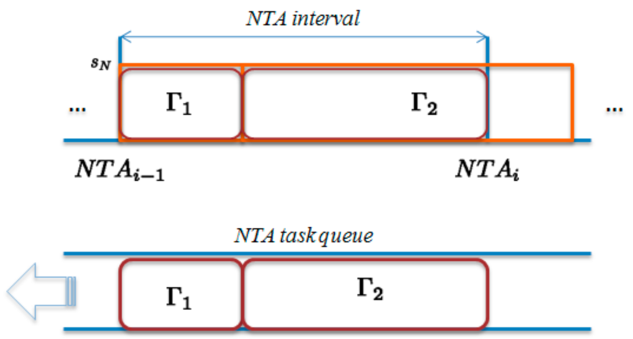


Fig. 23. Example of an NTA interval and the NTA task queue

While the temperature decreases naturally by the early completions of periodic workload, the worst-case periodic workload amount needs to be properly updated and estimated at run-time according to the completions of periodic tasks. For the management of the worst-case periodic workload in NTA intervals, we propose NTA *task queue* which is a *virtual* periodic task queue. The NTA task queue holds the periodic workload scheduling information at speed s_N, which is based on the workload information from actual periodic task queue. The NTA task queue is populated at every beginning moment of NTA intervals and it accepts the workload scheduled only for a single *NTA* interval. Thus, even when a periodic job is supposed to execute for more than one *NTA* interval, only the portion of the workload which is supposed to execute within the current *NTA* interval is queued in the NTA task queue. Figure 23 illustrates one NTA interval and the corresponding NTA task queue populated at the beginning instant of the NTA interval. In the figure, we see only a portion of Γ_2 is queued in the NTA task queue.

The management of worst-case workload by NTA task queue is as follows:

- NTA task queue is populated only at the beginning moment of NTA intervals based on the periodic task information in the actual periodic task queue of systems.

- The periodic task execution is based on NTA task queue. That is, if NTA task queue is empty, the system scheduler is not allowed to execute any periodic workload even when the actual periodic task queue is backlogged.

- While periodic task executes, NTA task queue is updated accordingly. When a periodic job is complete and removed from the actual task queue, the job is also removed from NTA task queue.

- For the run-time slack computation, the sum of worst-case periodic workload during a NTA interval is equal to the total sum of workload in NTA task queue.

Since the NTA task queue reflects early completions of periodic workload within each NTA interval, we are able to reclaim the unused clock-cycles of periodic tasks at run-time accordingly, which can reduce the response times of aperiodic workload significantly.

This way of dynamic slack reclamation approach is sub-optimal under the thermal constraints due to the possible non-work-conserving task scheduling. It should be very efficient at run-time, however, with low computational overhead and with the fact that we focus simply on every single NTA interval.

D. Experimental Evaluation

1. Simulation Model

We use the similar simulation environment as that used in [60]. Aperiodic tasks are generated by the exponential distribution using inter-arrival time ($1/\lambda$) and service time ($1/\mu$) with parameters λ and μ. Changing the values of μ and λ, we control the workload ($\rho = \lambda/\mu$) of aperiodic tasks under a fixed utilization of periodic tasks, U_t. There are four periodic tasks in Table III of which the total utilization is 0.4 as in [60].

Table III. Periodic Task Set

Task Set (second)

Task	Period	WCET
Γ_1	6	0.5
Γ_2	8	1.0
Γ_3	14	2.1
Γ_4	18	3.1
U_t	0.4	

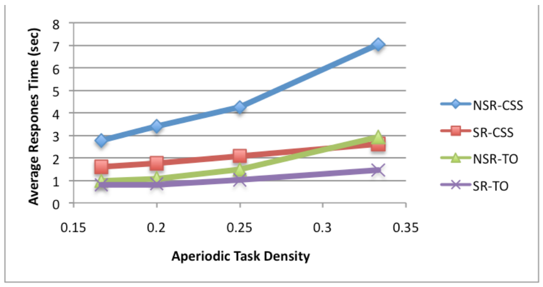
The actual execution time of each periodic task instance is generated by a normal distribution function in the range of [BCET, WCET], where BCET is the best-case execution time. The mean and the standard deviation were set to $\frac{(WCET+BCET)}{2}$ and $\frac{(WCET+BCET)}{6}$, respectively [61].

In the experiment, the time delay of speed scaling overhead is assumed to be negligible. For the simple comparisons of our proposed approach, we fix the maximum speed s_H to be 2.0 GHz. And the scaled (letting the ambient temperature $T_a = 0$) critical temperature T_c is set to be 80^oC.

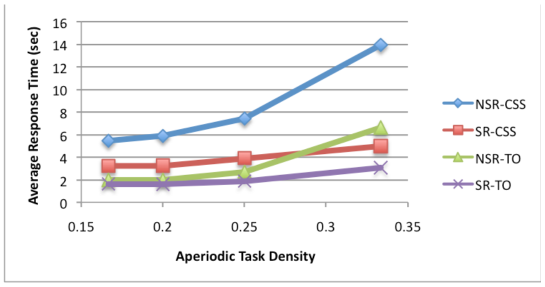
To evaluate our proposed thermally-aware transient overclocking algorithm, we also implemented non-slack-reclaiming static transient overclocking solution (NSR-TO) and a constant speed (s_E) slack stealing algorithm. For the constant speed slack stealing algorithm, we compute slack supposing the application of s_E both to aperiodic job and to periodic job until NTA. For the constant speed slack stealing solution as well, both the slack-reclaiming (SR-CSS) and non-slack-recaliming (NSR-CSS) mechanisms are considered and the results are displayed for the comparison.

Figure 24 shows the results of the average response times of aperiodic jobs with same periodic task set described in in Table III and changing the aperiodic task densities ($\rho = \frac{\lambda}{\mu}$). We fix BCET/WCET ratio $\gamma_{b/w}$ of periodic tasks at 0.1. For all experiments, the plots show that SR-TO scheme reduces the average response time the most, which is very evident that SR-TO minimizes the aperiodic response time by exploiting low thermal level and dynamic temporal slack reclamation. We also notice that the slack-reclaiming (SR) schemes are much less sensitive to the increase of aperiodic task density than the non-slack-reclaiming (NSR) schemes. That is because SR schemes are still capable of getting more clock-cycles as slack by the dynamic slack reclamation, while NSR schemes simply lose the available slacks according to the increase of total system utilization with aperiodic jobs.

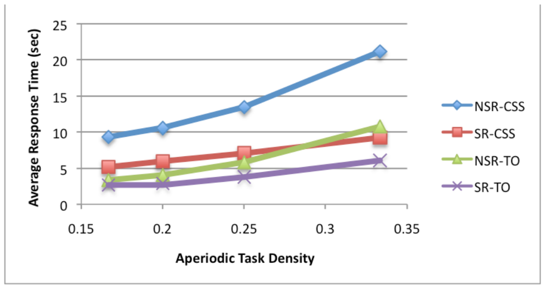
In Figure 24, we show the increase of average response times of aperiodic jobs



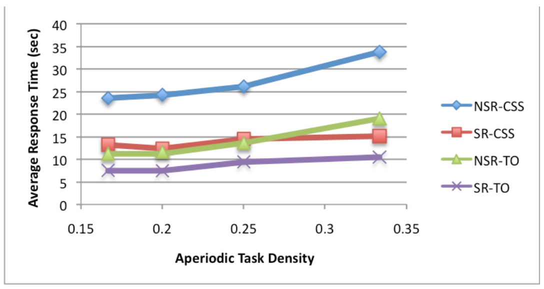
(a) $\frac{1}{\mu} = 1.0$



(b) $\frac{1}{\mu} = 2.0$



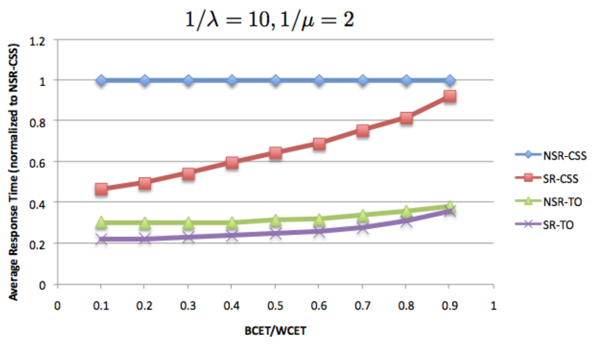
(c) $\frac{1}{\mu} = 4.0$



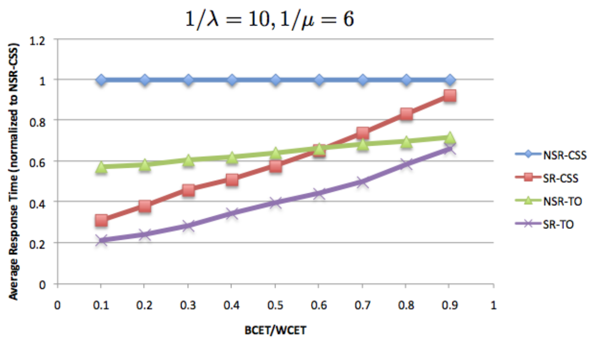
(d) $\frac{1}{\mu} = 8.0$

Fig. 24. Comparison of average response times according to various aperiodic task density

according to the increase of the ratio between BCET and WCET ($\gamma_{b/w}$). Along with the increase of $\gamma_{b/w}$, the difference of average response times between NSR schemes and SR schemes decrease since the larger $\gamma_{b/w}$ value means smaller temporal and thermal slack to be reclaimed dynamically. In Figure 25, for the longer aperiodic jobs (which present larger workload), we see more clearly the importance of dynamic slack reclamations. For the short aperiodic job executions in low utilized systems, the thermal-awareness plays an important in the reduction of response time of aperiodic jobs (Figure 25(a)). For the long aperiodic job, on the contrary, the dynamic slack reclamation appears to be more important (Figure 25(b)). This is because the highly utilized systems result in higher average system temperature, and the higher the average system temperature is, the less advantage of thermal-awareness there is.



(a) short aperiodic job executions, $\frac{1}{\mu} = 2.0$



(b) long aperiodic job executions, $\frac{1}{\mu} = 6.0$

Fig. 25. Comparison of average response times of aperiodic task

E. Conclusion

In this chapter, we have presented an *Online Thermally-Aware Transient Overclocking* scheme for mixed workload in hard real-time systems. Under the thermally constrained hard real-time systems, conventional ways of aperiodic job executions with no thermal consideration may result in either the thermal violation or very longer delays of aperiodic job execution. If we apply the fastest thermally safe constant speed for the aperiodic job execution as a very conservative way, it does not reduce the response times of aperiodic jobs sufficiently because *thermal slack* is not used. On the contrary, when the fastest processor speed is applied without the consideration of thermal conditions, the aperiodic job executions should lead to critically high temperature.

By the proposed *Thermally-Aware Slack-Stealing* algorithm, we compute the slack for aperiodic jobs online efficiently considering both the temporal and thermal constraints. A static approach which expects only the worst-case periodic workload is first described and the dynamic approach of the slack reclamation method is presented.

Simulations results have shown that the Thermally-Aware Transient Overclocking can effectively reduce the average response times of aperiodic workload and that the thermal-awareness is more important in the minimization of aperiodic jobs when the system is less utilized.

CHAPTER V

EFFICIENT CALIBRATION OF THERMAL MODELS BASED ON APPLICATION BEHAVIOR

A. Introduction

In Chapter II, we described a number of dynamic thermal management (DTM) approaches that have been proposed, ranging from dynamic voltage and frequency scaling (DVS) to clock gating and to architecture-level mechanisms that balance the load across functional units or cores on the processor chip. *Reactive* DTM approaches trigger appropriate thermal control actions (e.g. DVS or clock gating) whenever readings from thermal sensors exceed a particular level [4, 21, 62, 63, 64]. *Proactive* DTM, on the other hand, allows for optimal thermal control by predicting thermal trajectories ahead of time as a function of CPU frequencies [65] and task executions [66]. While proactive DTM has been shown to allow for better utilization of computational resources compared to reactive schemes [65], it is also significantly more difficult to implement in practice. Most notably, its effectiveness relies on the accuracy of the *thermal model* that underlies the prediction of the effects of speed scaling and task execution on CPU temperature.

The thermal behavior of microelectronic circuits is typically modeled using variations of an RC circuit [67]. Examples of applications of such RC models can be found in [35] for web farms, in [65] for optimal speed control, and in [68] as part of the HotSpot thermal simulation.

A major impediment when putting proactive thermal management into practice is the need to develop a thermal model that is appropriate for the platform at hand. Generic thermal models only loosely capture the thermal behavior, due to variability in fabrication, environmental effects, or the need for special configurations of either cooling devices or other aspects of packaging. The models must therefore be appropriately calibrated before use. Figure 26 illustrates the effect of different cooling environments (high/low fan speeds vs. external fan) on the thermal trajectory. Note that the thermal increase rate and the highest temperature level varies for different types of cooling support even with the workload and the CPU remaining the same. When the thermal management must deal with such variabilities, effective configu-

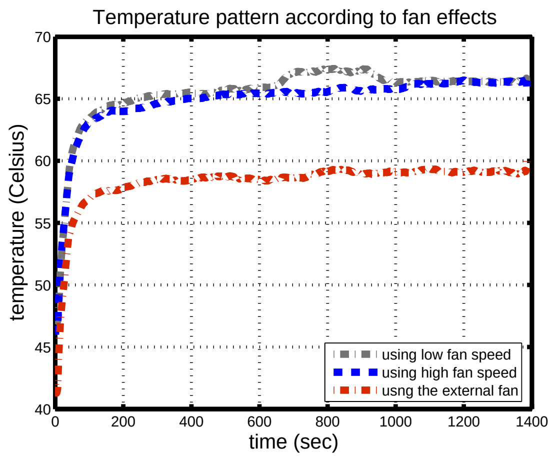


Fig. 26. Thermal trajectories with various cooling environments

ration and efficient calibration methods are needed to accurately parameterize the thermal models that drive the predictive thermal control. Unfortunately, as thermal models describe the relation between input power and thermal behavior, they rely on the availability of a measurement of input power for parameterization purposes. Since such measurements are typically not available at processor level in most systems, we need to find alternative ways for thermal model calibration.

In this chapter we describe an indirect methodology for parameterization of thermal models, which relies on the off-line analysis of the thermal behavior for a reference application, for which we measure both the detailed utilization behavior (through

PAPI performance measure counters) and the thermal behavior (through thermal sensors on the chip). We determine and later exploit a linear relation between energy consumption and utilization level to calibrate the thermal model for new *target applications* by comparing relative utilization levels. This results in accurate thermal model parameters without the need for power measurements.

This chapter is organized as follows: In Section B we give a brief overview of the thermal model used in this paper and of its RC circuit representation. In Section C we describe our indirect approach to estimate the parameters of the thermal model based on measurements of a reference application (`462.libquantum` from the SPEC CPU 2006 benchmark suite in our case). We describe how we measure and use relative utilization levels to calibrate the thermal model for new applications. Section D describes how we expand both the thermal model and the calibration approach to multicore processors, and Section E validates our approach with experimental results. Finally, we conclude with a summary and an outlook in Section F.

B. System Model

Effective prediction of the thermal trajectory depends first on a faithful thermal model and then on the accurate estimation and calibration of the specific parameters of the model for the system at hand. In this section we describe our thermal model and then proceed to elaborate on parameterization approaches that rely on power measurements. In the following sections we will describe methods to estimate model parameters at run time without the need to measure power.

1. Thermal Model

Thermal modeling of microprocessor circuits has traditionally applied variations of the simple Fourier's Law of heat conduction (e.g., [67, 62]). We assume that the environment has a fixed temperature, and that temperature is scaled so that the ambient temperature is zero. We define $T(t)$ and $P(t)$ as the temperature and the power input at time t, respectively. Fourier's Law is then expressed as follows:

$$T'(t) = \frac{P(t)}{C} - bT(t) \quad , \tag{5.1}$$

where the parameters R and C are the *thermal resistance* and *capacitance*, respectively, and capture the thermal characteristics of the processor chip under consideration. The term $b = 1/RC$ represents the *power dissipation rate* and is the inverse of thermal time constant of the system.

Simple thermal situations such as described in Equation (5.1) are often represented as RC circuits. Figure 27 shows the thermal RC circuit for a single-core pro-

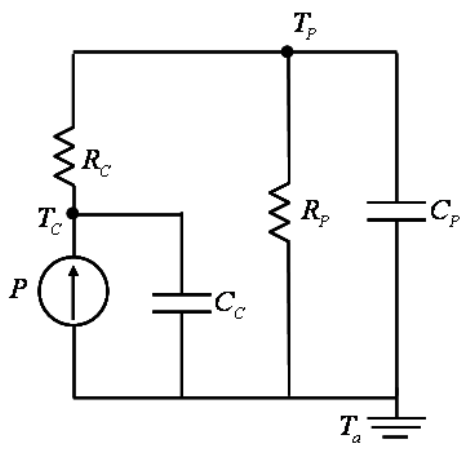


Fig. 27. Lumped RC thermal circuit model for a single-core processor

cessor, and it includes both the processor core and the packaging. We assume that the initial temperature is T_0, i.e., $T(t_0) = T_0$. If P is a constant power input level during a task execution interval, then the core's temperature can be approximated by the thermal circuit model and Equation (5.1) as follows:

$$\begin{aligned} T_c(t) &= T_{ss.c}(1 - e^{-b_c(t-t_0)}) + (T_{c.0} - T_{p.0})e^{-b_c(t-t_0)} \\ &+ T_{ss.p}(1 - e^{-b_p(t-t_0)}) + T_{p.0}e^{-b_p(t-t_0)}, \end{aligned} \tag{5.2}$$

where $T_{ss.c} = R_c P$ and $T_{ss.p} = R_p P$ are steady-state temperatures of core and package respectively, b_p is the power dissipation rate of the package, and b_c is that of the core itself. The above approximation is derived from the fact that the dissipation rate b_c of the core is much larger than that of the package, b_p. This causes the core's temperature increase to be dominated by the core's own thermal characteristic immediately after the input power is first applied, and then the core's temperature becomes dependent on the package's thermal characteristic [65]. Hence, the temperature increases very quickly at the beginning and it slows down its increasing rate according to the increase of the package temperature.

C. Thermal Prediction Based on Relative Utilization

Given a sampled thermal trajectory $T(t)$ of a system, the thermal parameters $T_{ss.c}$, $T_{ss.p}$, b_c, and b_p can be infered with help of Equation (5.2) if the input power P is known. In most systems, accurate power measurements are not available, and methods must be developed to capture the effect of input power onto the thermal behavior of the system without relying on absolute values for power levels. In the following we will describe how we use *relative utilization levels* to predict the thermal behavior of systems with the help of off-line measurement of thermal trajectories. In order to keep the following discussion simple, we limit ourselves to constant-speed processors, where the power consumption is application dependent. The results can be easily extended to the case of discrete processor's speed levels, where the power consumption is defined by the application and by the speed control module of the system.

The application of non-linear regression of Equation (5.2) on a measured thermal trajectory $T(t)$ yields estimations for $T_{ss.c}$, $b_c = 1/R_cC_c$, $T_{ss.p}$, and $b_p = 1/R_pC_p$. We note that the thermal model cannot be derived directly from the measurements, since *(a)* we do not know the input power P, and *(b)* we would have to factorize the results in order to get to the individual parameters. We therefore use an *indirect* approach - based on *relative utilizations* - to predict the thermal trajectory without knowledge of the input power level. We base our approach on the following observation:

Observation 2. *Assume a system with a constant-speed processor, and two applications* Γ_1 *and* Γ_2 *that execute workload amounts* X_1 *and* X_2 *(measured in number of instructions) during a given interval* $[0, t]$*, respectively. Let* E_1 *and* E_2 *be the energy consumed by* Γ_1 *and* Γ_2 *during the same interval. The energy consumption ratio* $\rho_e = E_1/E_2$*, is equal to the utilization ratio* $\rho_x = \frac{X_1/t}{X_2/t} = X_1/X_2$.

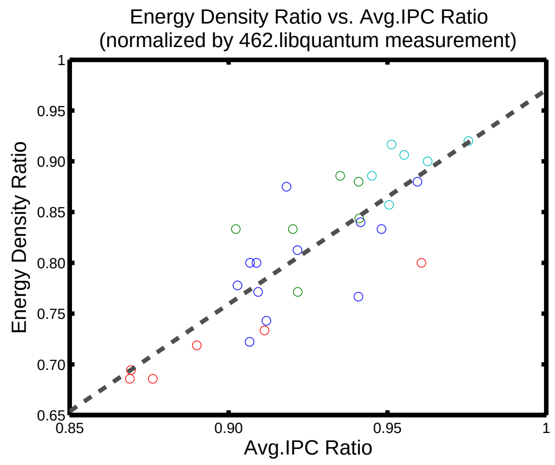


Fig. 28. Energy density vs. average IPC

The above observation is supported by experimental results. Figure 28 shows that the energy ratio during a given interval is proportional to the average utilization ratio for the same interval. The plot is derived from 30 sampled executions from 5 different benchmarks[1]. Li *et al.* make a similar observation about the relationship between the average power and the utilization (IPC) in [69]. The energy density (average power) was computed from Equation (5.2) and the dashed line shows a linear regression of the data.

The simple relation between utilization ratio and energy consumption ratio can be used to estimate the parameters $T_{ss.p}^{target}$ and $T_{ss.c}^{target}$ of a new application Γ_{target} as follows:

Calibration:

Step 1: Measure the thermal trajectory $T_{ref}(t)$ of the reference application Γ_{ref}. (In our case we use the SPEC CPU 2006 benchmark `462.libquantum` because of its constantly high utilization level.)

Step 2: Determine the parameters $T_{ss.c}^{ref}$, $T_{ss.p}^{ref}$, $b_{c.ref}$, and $b_{p.ref}$ from $T_{ref}(t)$, for example through nonlinear regression.

Step 3: Measure the average utilization level IPC_{ref} of the reference application Γ_{ref}.

Run-Time Estimation:

Step 4: Measure the utilization level IPC_{target} of the target application Γ_{target}.

Step 5: Predict the thermal trajectory $T_{target}(t)$ using Equation (5.2) with parameters $T_{ss.c}^{ref} \times \frac{IPC_{target}}{IPC_{ref}}$, $T_{ss.p}^{ref} \times \frac{IPC_{target}}{IPC_{ref}}$, $b_{c.ref}$, and $b_{p.ref}$.

[1]We measure the utilization level using the `PAPI_ipc` function of the $PAPI$ performance monitoring library. At 1-second intervals, we call `PAPI_ipc` to retrieve the current instruction and cycle counter, respectively. From these values we can determine a measure for the utilization of the processor.

(In the following we will use the notation $b_{c/p}$ to denote both $b_{c/p}$ and $b_{c.ref/p.ref}$.) We note that Step 1 to Step 3 form the calibration of the thermal prediction system and are performed off-line to determine a reference level. They may need to be performed when a system is deployed to account for environmental factors affecting cooling. For systems with multiple speed levels this calibration may need to be run once for each processor speed level. For systems with dynamic active heat dissipation mechanisms (e.g. fans with multiple speed levels) the calibration may need to be repeated as well to account for the multiple levels of thermal dissipation. Step 4 and Step 5 are executed at run time to exercise the model for the target application.

We will describe later how the accuracy can be further improved by using smaller discrete time intervals to better track application dynamics. The following equation or inequality shows how the temperature can be approximated in terms of energy consumption. That is,

$$\begin{aligned} T(t) &= \frac{P}{C}\int_0^t e^{-b(t-x)}dx + T_0 e^{-bt} && (5.3)\\ &\leq \frac{P}{C}\int_0^t dx + T_0 e^{-bt} && (5.4)\\ &= \frac{P\cdot t}{C} + T_0 e^{-bt} = \frac{E(t)}{C} + T_0 e^{-bt} \\ \Longleftrightarrow \quad & T(t) \leq \frac{E(t)}{C} + T_0 e^{-bt} \quad , && (5.5)\end{aligned}$$

where Equation (5.3) is the solution of the Fourier Law in Equation (5.1) for a constant input power P. For the transition from Equation (5.3) to Equation (5.4) we take advantage of the fact that $\int_0^t e^{-b(t-x)}dx = 1/b(1-e^{-bt}) \leq 1/b \cdot bt$. As a result, the temperature can be approximated with help of the energy consumed during the time interval $[0, t]$, which we denote by $E(t)$. Similarly, we approximate the term $(1-e^{-bt})$ in Equation (5.2) by the term bt, which allows for the following simplifications of the equation:

$$\begin{aligned} T_c(t) &\leq \frac{E(t)}{C_c} + (T_{c.0} - T_{p.0})e^{-b_c t} + \frac{E(t)}{C_p} + T_{p.0}e^{-b_p t} \\ &= R_c b_c E(t) + (T_{c.0} - T_{p.0})e^{-b_c t} \\ &\quad + R_p b_p E(t) + T_{p.0}e^{-b_p t} \quad . \end{aligned} \tag{5.6}$$

Thus, if we discretize a task execution period with the interval of length Δ, the temperature variation is expressed by the energy consumption as follows in a recursive way,

$$\begin{aligned} T_c((i+1)\Delta) &\approx R_c b_c E(\Delta) + (T_c(i\Delta) - T_p(i\Delta))e^{-b_c\Delta} \\ &\quad + R_p b_p E(\Delta) + T_p(i\Delta)e^{-b_p\Delta}. \end{aligned} \tag{5.7}$$

We note that b_c and b_p have been previously determined, and therefore $e^{-b_{c/p}\Delta}$ are constants.

Given the above thermal approximation based on the energy consumption, we are able to estimate the thermal trajectory of any task execution as a result of monitored IPC values. The selection of the interval length Δ depends on the thermal parameters b_c and b_p so that the error between the estimated and the real temperature values be acceptably small. At the i^{th} interval, the temperature error by the energy approximation is described as follows,

$$\begin{aligned} error(i\Delta) &= R_c b_c E(\Delta) + R_p b_p E(\Delta) \\ &\quad + (T_c((i-1)\Delta) - T_p((i-1)\Delta))e^{-b_c\Delta} \\ &\quad + T_p((i-1)\Delta)e^{-b_p\Delta} \\ &\quad - R_c P(1 - e^{-b_c i\Delta}) - R_p P(1 - e^{-b_p i\Delta}) \\ &\leq R_c b_c P\Delta + R_p b_p P\Delta. \end{aligned} \tag{5.8}$$

Thus, suppose for example that $\Delta = \frac{1}{b_p 20} = \frac{1}{b_c 10}$, $R_c P = T_{ss.c} = 20$, and $R_p P = T_{ss.p} = 15$, the maximum error is $2.75^o C$. And the error decreases with decreasing Δ.

1. Temperature Predictions for Tasks with Dynamic Utilization

The temperature approximation method represented in Equation 5.7 can be easily extended to predict temperatures in systems where tasks display dynamic (i.e., time-varying) utilization levels. We exploit the relation between utilization level and energy consumption previously determined in Step 2 of the calibration. Given the estimated average energy consumption E_{ref} of the reference application, Equation 5.7 for the thermal trajectory can be modified as follows:

$$\begin{aligned} T_c((i+1)\Delta) \approx\ & R_c b_c E_{ref} \cdot \rho_{ipc,i} + (T_c(i\Delta) - T_p(i\Delta))e^{-b_c \Delta} \\ & + R_p b_p E_{ref} \cdot \rho_{ipc,i} + T_p(i\Delta)e^{-b_p \Delta} \quad , \end{aligned} \tag{5.9}$$

where $\rho_{ipc,i} = \frac{IPC_{target(i)}}{IPC_{ref}}$ is the utilization ratio during interval i.

Although the selection of energy as a control factor and the local linearization of the model result in an overestimation in terms of the temperature increase, we will show in Section E that the error is acceptable for the thermal estimation by the appropriate choice of timing discretization length. In addition, by safely overestimating the thermal trajectory, this methodology can be applied in thermal management systems that require conservative handling of thermal behavior.

Figure 29 shows the temperature measurement of a benchmark, `462.libquantum` and nonlinear regression of the measurement to estimate thermal parameters of processor. The figure also shows the thermal approximation by the proposed utilization-based approach for `462.libquantum` which is the reference for thermal estimations of other applications shown in Section E.

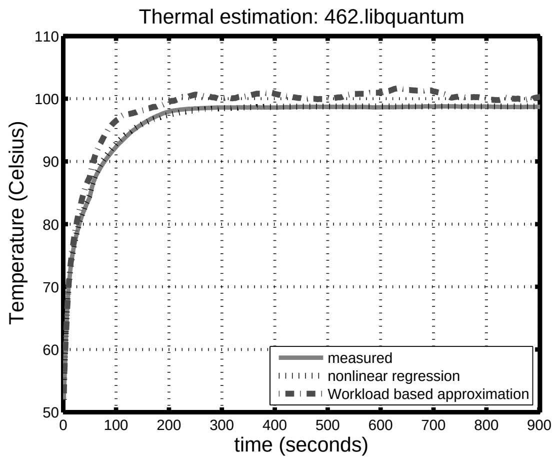


Fig. 29. Thermal parameter estimation & utilization based temperature approximation

D. Extensions of Thermal Prediction Model for Multicore

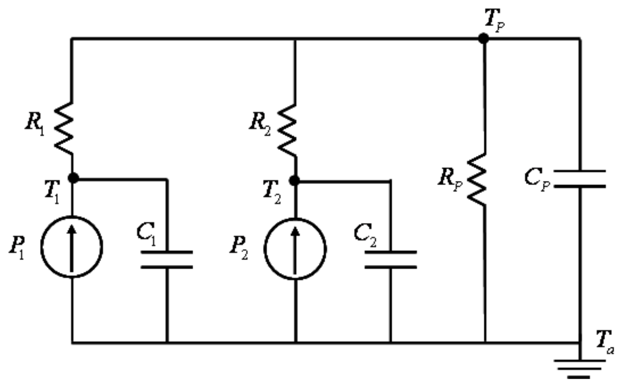


Fig. 30. A lumped RC thermal circuit model for dual core processors

In this section, we derive the estimation of thermal trajectories in dual-core system. For the multicore systems, the RC thermal circuit can be derived as shown in Figure 30. Each core's thermal trajectory can be expressed as follows,

$$\begin{aligned} T_1(t) &= R_p(P_1+P_2)(1-e^{-b_p t}) + T_{p.0}e^{-b_p t} \\ &\quad + R_1P_1(1-e^{-b_1 t}) + (T_{1.0}-T_{p.0})e^{-b_1 t} \\ T_2(t) &= R_p(P_1+P_2)(1-e^{-b_p t}) + T_{p.0}e^{-b_p t} \\ &\quad + R_2P_2(1-e^{-b_2 t}) + (T_{2.0}-T_{p.0})e^{-b_2 t}, \end{aligned} \tag{5.10}$$

where C_p, b_p, and $T_{p.0}$ are thermal parameters and the initial temperature of the packaging - for example the heat-sink contacts shared by both cores. Similarly to the single-core case, we estimate the parameters as described in Section C based on the thermal trajectory caused by a reference application (`462.libquantum` in our case) on each core. Given the symmetric arrangement of cores in many architectures, one may be tempted to use a system-level model for each core. However, due to variations in fabrication and external asymmetries caused by the layout and the integration of the system with external cooling mechanisms, each core is likely to have different thermal characteristics. It is advantageous, therefore, to derive the thermal parameters separately for each core. Similarly to Equation (5.6), for the single-core case, Equation (5.10) can be approximated in terms of energy consumption as well, i.e.,

$$\begin{aligned} T_1(t) &= R_1b_1E_1 + (T_{1.0}-T_{p.0})e^{-b_1 t} \\ &\quad + R_pb_p(E_1+E_2) + T_{p.0}e^{-b_p t} \\ T_2(t) &= R_2b_2E_2 + (T_{2.0}-T_{p.0})e^{-b_2 t} \\ &\quad + R_pb_p(E_1+E_2) + T_{p.0}e^{-b_p t}. \end{aligned} \tag{5.11}$$

We note that Equation(5.10) indicates how the thermal behavior of each core is affected by the other core's input power and the temperature level throughout the package.

E. Results and Analysis

1. Experimental Setup

We evaluate the proposed thermal model in a real-world CMP product. In order to estimate each core's working temperature individually, we develop a specific device driver for accessing the Digital Thermal Sensor (DTS) in our multicore system at runtime. The trigger point of these thermal sensors is not programmable by software since it is set during the fabrication of the processor [70]. To validate our thermal model, we conduct our experiments using the system described in Table IV.

Table IV. Experimental Systems Description

	System
The number of cores	4 cores
Processor	Intel Quad Core Q6600
Memory Size	1 GB
Operating System.	SUSE 10.3 (Kernel Version: 2.6.27)

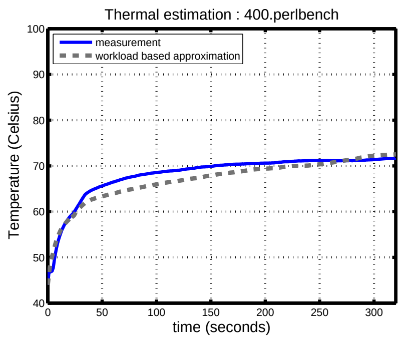
2. Experimental Results

Figure 31 shows the measured temperatures and the comparisons with the predicted thermal trajectories. The results show how the approximations match the measured thermal trajectories.

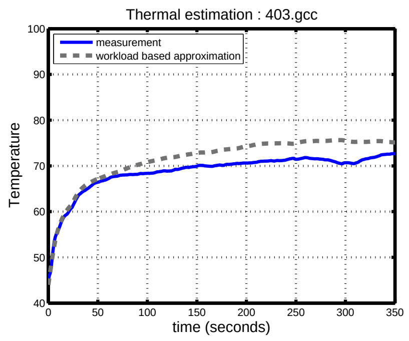
We define the error as:

$$error = \sqrt{\frac{\sum_{i=0}^{(t_f/\Delta)} \mid T_m(i\Delta) - T_a(i\Delta) \mid^2}{(t_f/\Delta)}}, \tag{5.12}$$

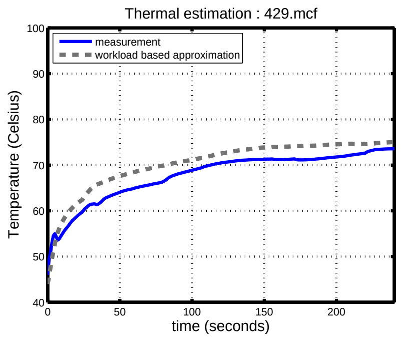
where T_m is the measured temperature, T_a is the estimated value, and t_f is the measurement interval length in seconds. Table V shows the average error, which is found to be at most 3.5^oC for these benchmarks. Note that we should be able to decrease the error when a shorter monitoring time interval is selected for IPC.



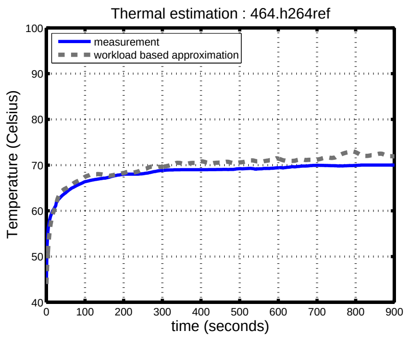
(a) 400.perlbench



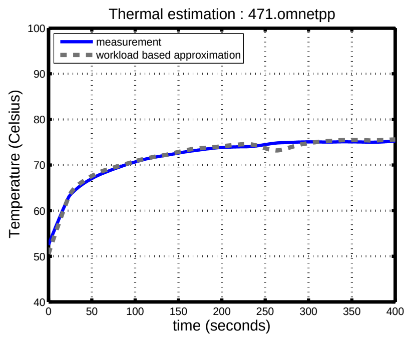
(b) 403.gcc



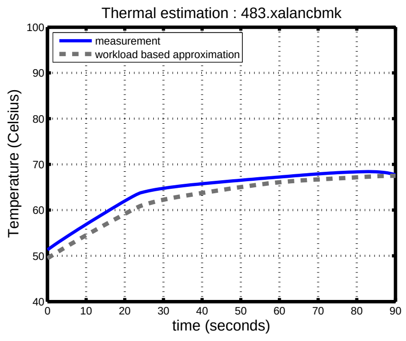
(c) 429.mcf



(d) 464.h264ref



(e) 471.omnetpp



(f) 483.xalancbmk

Fig. 31. The temperature estimation in Intel Quad-Core Q9650 processor

Figure 32 shows the experimental result for the dual-core case. In the experiment, two different benchmarks, `464.h264ref` and `462.libquantum` were executed on *core 1* and *core 2* concurrently. By the measured IPCs from each processor core,

Table V. Average Error of Utilization Based Thermal Estimation

Benchmark	Avg.Error (^oC)
400.perlbench	2.09
403.gcc	3.47
429.mcf	3.49
464.h264ref	1.73
471.omnetpp	0.98
483.xalancbmk	2.74

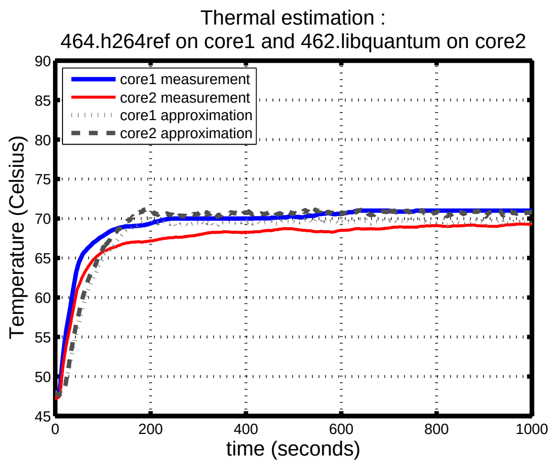


Fig. 32. The temperature estimation of two cores: 464.perlbench & 462.libquantum

we estimated the thermal trajectory of each core based on the Equation (5.11) and compared the results with measured values.

F. Conclusion and Future Work

Many of today's commercial computing systems and embedded devices do not have support for monitoring of power and energy. While this is rarely a problem for energy awareness (when in doubt, reduce speed), it becomes critical when attempting to predict the thermal behavior of a system. In this paper we develop a thermal model and a methodology for the efficient calibration of its parameters that does not rely on explicit information about input power. Rather, we closely monitor the thermal behavior of a well-known reference application on the given computing platform, and we then take advantage of the tight relationship between processor utilization and power consumption to predict the thermal trajectory for an unknown application at run time. The benefits of this methodology are numerous: First, the thermal characteristic of the computing platform can be explored after deployment time. For example, the system may run the reference application during configuration or even boot time and derive the thermal parameters shortly before run time. Next, it is easy to run this step for different levels of active heat dissipation if so desired. For example, the system may have a combination of on-package fans and case fans, which may be operated independently of each other. For each combination of such operating fans, the thermal characteristics of the system can be captured by the model and the parameters derived separately, thus enabling the effective prediction of thermal trajectories with dynamic active heat dissipation mechanisms. Finally, utilization information is readily available at run time, for example through Instructions Per Cycle (IPC) counter in PAPI.

In order to verify our thermal prediction model, we experiment with other benchmark applications for the prediction of temperature variations based on the IPC data monitored at every second. The experimental results show that the predictions are very close to the measurement of actual temperature values measured via Digital Thermal Sensor (DTS) embedded in each core despite the overestimation error that results from the energy-based approximation.

In the future work, we plan to extend the model to address IO-intensive applications, by better monitoring both reference and run-time application. This requires better information to be available about the proportion of computation-intensive instructions vs. total number of instructions at every monitoring interval. Similarly, the model must be extended to better reflect the thermal reality on the chip. In addition to IPC, other system information, like cache-misses or memory accesses, need to be considered for general applications.

CHAPTER VI

CONCLUSIONS AND FUTURE WORK

In this chapter, we summarize the major results of this research and discuss future directions of this work.

A. Conclusions

The high level of power density and the resulting heat generation has become a critical problem in the design of modern processors. Since many static hardware-level thermal solutions (e.g., fans, heat-sinks, coolants, and etc.) can be expensive and hard to implement, in particular on small-sized embedded systems, various run-time *Dynamic Thermal Management* (DTM) mechanisms have been researched. Most DTM mechanisms are equivalent to the speed control of the processor and take advantage of the fact that the power consumption is the source of heat generation, and the power is regarded as a convex function of processor speed.

In our study, we focused on the dynamic speed control for the scheduling of mixed workloads on thermally-constrained hard real-time systems. A mixed workload is composed of a periodic task set and a sequence of aperiodic jobs. In such a system, we say that the workload can be feasibly schedulable if all the deadline constraints of periodic task instances are satisfied without the maximum safe temperature of the processor being exceeded. To minimize the response time of aperiodic jobs, we presented *Transient Overclocking* as an approach by which we apply the maximum CPU speed to the execution of aperiodic jobs. However, uncontrolled transient overclocking would lead to excessive CPU temperatures. Thus, the speed control has two contradictory directions to achieve two objectives: While the use of higher speed minimizes the response times of aperiodic jobs at the cost of temperature increase, the use of lower processor speeds keeps the CPU temperature at lower levels. Our aim in the study is to propose an appropriate thermally-aware speed control mechanism to execute the mixed workload to minimize the response times of aperiodic workloads while satisfying both the deadline constraints of periodic tasks and the thermal constraint of the system.

Our attention is also on the derivation of a correct thermal model that can in practice be efficiently applied for the thermal management. Thermally-aware speed control requires a specific thermal model. Given such a model, we still need to calibrate it to account for various cooling environments and different thermal characteristics of each system. Without a properly calibrated thermal model, thermal trajectories are predicted incorrectly, and any predictive thermally-aware speed control will work incorrectly. In this study, therefore, we also propose an efficient way of thermal model calibration based on application behaviors.

For this work, in Chapter II, we reviewed briefly the power model and the thermal model on which the speed control is based. For the mixed-workload executions, the traditional aperiodic task scheduling algorithms were also reviewed. The various thermal management mechanisms and thermal modelings were summarized as well.

In Chapter III, we first described how transient overclocking can be applied to safely reduce response times for aperiodic jobs in the presence of hard real-time periodic tasks. We then proposed a design-time method to allocate overclocking budget to aperiodic workloads and the performance was compared by the discrete event-based simulator. The proposed design-time budget based method has the advantage of low overhead at run-time. Without the need to monitor the temperature of microprocessors at run-time, the scheme provides a safe overclocking budget to aperiodic jobs.

In Chapter IV, we devised the *on-line thermally-aware transient overclocking* scheme for mixed workloads. By the design-time method which always assumes the worst-case workload, the thermal slack and the temporal slack are not efficiently used for the reduction of aperiodic job response times. We proposed, therefore, *Thermally-Aware Slack-Stealing* algorithm to derive the maximum instantaneous slack for aperiodic job execution without ruining the deadline constraints of periodic tasks or overheating the microprocessor. With the application of the conventional *energy-aware DVS scheme* to periodic task executions and through the thermal prediction approximation, we could compute the available slack conveniently with low computational overhead. We showed the proposed *on-line thermally-aware transient overclocking* method outperforms other constant speed schemes guaranteeing the thermal and temporal constraints.

In Chapter V, we describe an indirect methodology for parameterization of thermal models. Based on the analysis of a reference application along with the measurement of its utilization behavior and the thermal behavior, we calibrate the thermal model for other applications by comparing relative utilization levels. We verified the proposed approach of thermal model calibration by some experiments with various benchmark applications for the prediction of temperature variations.

B. Future Work

Our research goal is to devise a thermally-aware dynamic speed control scheme for mixed workload in real-time systems. The feasible dynamic speed control should be based on the correct thermal model. We thus proposed an efficient methodology of thermal model calibration, too. We believe that proposed dynamic speed control scheme based on the thermal model calibrated will be very useful for the design of practical thermally-aware real-time systems. Some issues remain open for further research. For either off-line or on-line transient overclocking scheme, there is no accurate analytical tool to help engineers' speed control decision, for example, the performance comparisons of various schemes were dependent on simulations. Although most researches and the evaluation of task scheduling algorithms for mixed workload depend on simulations, analytical tools and models are worth being studied and developed. We believe they will help the design of optimal thermally-aware speed control and the task scheduling algorithm at the design-time.

For the thermal model calibration methodology, we need to extend the approach to apply for the IO-intensive applications, too. Besides the measurement of IPC values for applications, there should be the consideration of other metrics like the cache-miss ratios or the number of memory accesses to correctly estimate the system operation effects on the microprocessor temperature.

REFERENCES

[1] F. J. Pollack, “New microarchitecture challenges in the coming generations of cmos process technologies,” in *Proc. 32nd International Symposium on Microarchitecture*, 1999, p. 2.

[2] S. Borkar, “Design challenges of technology scaling,” *IEEE Micro*, vol. 19, no. 4, pp. 23–29, Jul-Aug 1999.

[3] G. H. Loh, Y. Xie, and B. Black, “Processor design in 3D die-stacking technologies,” *IEEE Micro*, vol. 27, no. 3, pp. 31–48, 2007.

[4] D. Brooks and M. Martonosi, “Dynamic thermal management for high-performance microprocessors,” in *Proc. 7th International Symposium on High-Performance Computer Architecture*, 2001, pp. 171–182.

[5] V. Tiwari, D. Singh, S. Rajgopal, G. Mehta, R. Patel, and F. Baez, “Reducing power in high-performance microprocessors,” in *Proc. 35th Design Automation Conference (DAC)*, 1998, pp. 732–737.

[6] E. Rotem, A. Naveh, M. Moffie, and A. Mendelson, “Analysis of thermal monitor features of the Intel Pentium M processor,” in *the First Workshop on Temperature-aware Computer Systems*, 2004.

[7] K. Skadron, M. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan, “Temperature-aware microarchitecture: Extended discussion and results,” Tech. Rep. CS-2003-08, Department of Computer Science, University of Virginia, 2003.

[8] C. L. Liu and J. W. Layland, “Scheduling algorithms for multiprogramming in a hard-real-time environment,” *J. ACM*, vol. 20, no. 1, pp. 46–61, 1973.

[9] J. P. Lehoczky and S. Ramos-Thuel, “An optimal algorithm for scheduling soft-aperiodic tasks in fixed priority preemptive systems,” in *Proc. 13th Real-Time Systems Symposium*, 1992, pp. 110–123.

[10] J. K. Strosnider, J. P. Lehoczky, and L. Sha, “The deferrable server algorithm for enhanced aperiodic responsiveness in hard real-time environments,” *IEEE Transactions on Computers*, vol. 44, no. 1, pp. 73–91, 1995.

[11] J-J Chen, S. Wang, and L. Thiele, “Proactive speed scheduling for real-time tasks under thermal constraints,” in *Proc. 15th Real-Time and Embedded Technology and Applications Symposium*, 2009, pp. 141–150.

[12] T. Chantem, R. P. Dick, and X. S. Hu, “Temperature-aware scheduling and assignment for hard real-time applications on MPSoCs,” in *Proc. 11th Design, Automation and Test in Europe*, 2008, pp. 288–293.

[13] J. M. Rabaey, A. Chandrakasan, and B. Nikolic, *Digital Integrated Circuits: A Design Perspective*, Upper Saddle River, NJ: Prentice Hall, 2002.

[14] N. Bansal, T.Kimbrel, and K. Pruhs, “Dynamic speed scaling to manage energy and temperature,” in *Proc. 45th Symposium on Foundations of Computer Science*, 2004, pp. 520–529.

[15] N. Bansal and K. Pruhs, “Speed scaling to manage temperature,” in *Proc. 22th Symposium on Theorectical Aspects of Computer Science*, 2005, pp. 460–471.

[16] J. Liu, *Real-Time Systems*, New Jersey: Prentice Hall, 2000.

[17] B. Sprunt, “Aperiodic task scheduling for real-time systems,” Ph.D. dissertation, Carnegie Mellon University, Dept. of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburg, PA, 1990.

[18] M. Spuri and G. C. Buttazzo, "Scheduling aperiodic tasks in dynamic priority systems," *Real-Time Systems*, vol. 10, no. 2, pp. 179–210, 1996.

[19] L. Abeni and G. C. Buttazzo, "Integrating multimedia applications in hard real-time systems," in *Proc. 19th Real-Time Systems Symposium*, 1998, pp. 4–13.

[20] R. I. Davis, K. W. Tindell, and A. Burns, "Scheduling slack time in fixed priority preemptive systems," in *Proc. 14th Real-Time Systems Symposium*, 1993, pp. 222–231.

[21] S. Wang and R. Bettati, "Reactive speed control in temperature-constrained real-time systems," in *Proc. 18th Euromicro Conference on Real-Time Systems*, 2006, pp. 73–95.

[22] R. Rao, S. Vrudhula, C. Chakrabarti, and N. Chang, "An optimal analytical solution for processor speed control with thermal constraints," in *Proc. 11th International Symposium on Low Power Electronics and Design*, 2006, pp. 292–297.

[23] S. Wang and R. Bettati, "Delay analysis in temperature-constrained hard real-time systems with general task arrivals," in *Proc. 27th Real-Time Systems Symposium*, 2006, pp. 323–334.

[24] P. Pillai and K. G. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems," in *Proc. 18th Symposium on Operating Systems Principles*, 2001, pp. 89–102.

[25] H. Aydin, R. Melhem, D. Mosse', and P. M. Alvarez, "Dynamic and aggressive

scheduling techniques for power-aware real-time systems," in *Proc. 22nd Real-Time Systems Symposium*, 2001, pp. 95–105.

[26] A. Qadi, S. Goddard, and S. Farritor, "A dynamic voltage scaling algorithm for sporadic tasks," in *Proc. 24th Real-Time Systems Symposium*, 2003, pp. 52–62.

[27] Y. Liu and A. K. Mok, "An integrated approach for applying dynamic voltage scaling to hard real-time systems," in *Proc. 9th Real Time Technology and Applications Symposium*, 2003, pp. 116–123.

[28] S. Saewong and R. Rajkumar, "Practical voltage-scaling for fixed-priority rt-systems," in *Proc. 9th Real Time Technology and Applications Symposium*, 2003, pp. 106–114.

[29] G. Quan, L. Niu, X. S. Hu, and B. Mochocki, "Fixed priority scheduling for reducing overall energy on variable voltage processors," in *Proc. 25th Real-Time Systems Symposium*, 2004, pp. 309–318.

[30] F. Zhang and S. T. Chanson, "Power-aware processor scheduling under average delay constraints," in *Proc. 11th Real-Time and Embedded Technology and Applications Symposium*, 2005, pp. 202–212.

[31] K. Skadron, M. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan, "Temperature-aware microarchitecture," in *Proc. 30th International Symposium on Computer Architecture*, 2003, pp. 2–13.

[32] A. Cohen, L. Finkelstein, A. Mendelson, R. Ronen, and D. Rudoy, "On estimating optimal performance of cpu dynamic thermal management," *Computer Architecture Letters*, vol. 2, pp. 6, 2003.

[33] J. Srinivasan and S. Adve, "Predictive dynamic thermal management for multimedia applications," in *Proc. 17th International Conference on Supercomputing*, 2003, pp. 109–120.

[34] K. Skadron, T. Abdelzaher, and M. R. Stan, "Control-theoretic techniques and thermal-RC modeling for accurate and localized dynamic thermal management," in *Proc. 8th International Symposium on High-Performance Computer Architecture*, 2002, pp. 17–28.

[35] A. Ferreira, D. Mosse, and J.C. Oh, "Thermal faults modeling using a RC model with an application to web farms," in *Proc. 19th Euromicro Conference on Real-Time Systems*, Pisa, Italy, 2007, pp. 113–124.

[36] C. C. N. Chu and M. D. F. Wong, "A matrix synthesis approach to thermal placement," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 17, no. 11, pp. 1166–1174, 1998.

[37] G. Chen and S. Sapatnekar, "Partition-driven standard cell thermal placement," in *Proc. 7th International Symposium on Physical Design*, 2003, pp. 75–80.

[38] B. Goplen and S. Sapatnekar, "Thermal via placement in 3D ICs," in *Proc. 9th International Symposium on Physical Design*, 2005, pp. 167–174.

[39] E. Rohou, E. Rohou, and M. D. Smith, "Dynamically managing processor temperature and power," in *Proc. 2nd Workshop on Feedback-Directed Optimization*, 1999.

[40] J. Donald and M. Martonosi, "Techniques for multicore thermal management: Classification and new exploration," in *Proc. 33rd International Symposium on Computer Architecture*, 2006, pp. 78–88.

[41] S. Zhang and K. S. Chatha, "Approximation algorithm for the temperature-aware scheduling problem," in *Proc. 18th International Conference on Computer Aided Design*, 2007, pp. 281–288.

[42] W. Kim, D. Shin, H-S Yun, J. Kim, and S. L. Min, "Performance comparison of dynamic voltage scaling algorithms for hard real-time systems," in *Proc. 8th Real Time Technology and Applications Symposium*, 2002, pp. 219–228.

[43] T-Y Wang and C. C-P Chen, "3-D Thermal-ADI: A linear-time chip level transient thermal simulator," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 21, no. 12, pp. 1434–1445, 2002.

[44] T-Y Wang and C. Chen, "SPICE-compatible thermal simulation with lumped circuit modeling for thermal reliability analysis based on modeling order reduction," in *Proc. 5th International Symposium on Quality Electronic Design*, 2004, pp. 357–362.

[45] H. Qian, "Random walks in a supply network," in *Proc. 40th Design Automation Conference*, 2003, pp. 93–98.

[46] K. Skadron, M. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan, "Temperature-aware microarchitecture: Modeling and implementation," *ACM Transactions on Architecture and Code Optimization*, vol. 1, no. 1, pp. 94–125, 2004.

[47] H. Su, F. Liu, A. Devgan, E. Acar, and S. Nassif, "Full chip leakage estimation considering power supply and temperature variations," in *Proc. 8th International Symposium on Low Power Electronics and Design*, 2003, pp. 78–83.

[48] Y. Zhan and S. S. Sapatnekar, "Fast computation of the temperature distribution

in VLSI chips using the discrete cosine transform and table look-up," in *Proc. 10th Asia and South Pacific Design Automation Conference*, 2005, pp. 87–92.

[49] M. Huang, J. Renau, S-M Yoo, and J. Torrellas, "A framework for dynamic energy efficiency and temperature management," in *Proc. 33th International Symposium on Microarchitecture*, 2000, pp. 202–213.

[50] L. Kleinrock, *Queueing Systems*, New York: John Wiley and Sons, 1975.

[51] T-H Lin and W. Tarng, "Scheduling periodic and aperiodic tasks in hard real-time computing systems," in *Proc. ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, 1991, pp. 31–38.

[52] V. V. Ragulsky and V. G. Sidorovich, "On the availability of a free-space optical communication link operating under various atmospheric conditions," in *SPIE*, 2003.

[53] P. Gupta and P. R. Kumar, "The capacity of wireless networks," *IEEE Trans. on Information Theory*, vol. 46, no. 2, pp. 388 – 404, 2000.

[54] G. Bollella, B. Brosgol, P. Dibble, S. Furr, J. Gosling, D. Hardin, M. Turnbull, R. Belliardi, D. Holmes, and A. Wellings, "JSR001: Real-time specification for Java," Java Community Process, http://www.jcp.org/en/jsr/detail?id=1; accessed July 25, 2010.

[55] T. Mann, M. Deters, R. LeGrand, and R. K. Cytron, "Static determination of allocation rates to support real-time garbage collection," in *Proc. 9th Languages, Compilers, and Tools for Embedded Systems*, 2005, pp. 193–202.

[56] C. Moser, D. Brunelli, L. Thiele, and L. Benini, "Real-time scheduling with regenerative energy," in *Proc. 18th Euromicro Conference on Real-Time Systems*,

2006, pp. 261–270.

[57] R. Jejurikar and R. K. Gupta, “Dynamic slack reclamation with procrastination scheduling in real-time embedded systems,” in *Proc. 42nd Design Automation Conference*, 2005, pp. 111–116.

[58] T. Ishihara and H. Yasuura, “Voltage scheduling problem for dynamically variable voltage processors,” in *Proc. 3rd International Symposium on Low Power Electronics and Design*, 1998, pp. 197–202.

[59] L. Yuan and Gang Qu, “Analysis of energy reduction on dynamic voltage scaling-enabled systems,” *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 24, no. 12, pp. 1827–1837, 2005.

[60] D. Shin and J. Kim, “Dynamic voltage scaling of mixed task sets in priority-driven systems,” *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 25, no. 3, pp. 438–453, 2006.

[61] H. Aydin and Q. Yang, “Energy - responsiveness tradeoffs for real-time systems with mixed workload,” in *Proc. 10th Real-Time and Embedded Technology and Applications Symposium*, 2004, pp. 74–83.

[62] N. Bansal and K. Pruhs, “Speed scaling to manage energy and temperature,” *Journal of ACM*, vol. 54, no. 1, pp. 1–39, 2007.

[63] A. Kumar, L. Shang, L-S Peh, and N. K. Jha, “HybDTM: A coordinated hardware-software approach for dynamic thermal management,” in *Proc. 43rd Design Automation Conference*, 2006, pp. 548–553.

[64] Y. Ahn and R. Bettati, “Transient overclocking for aperiodic task execution in hard real-time systems,” in *Proc. 20th Euromicro Conference on Real-Time*

Systems, July 2008, pp. 102–111.

[65] R. Rao and S. Vrudhula, "Performance optimal processor throttling under thermal constraints," in *Proc. International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, 2007, pp. 257–266.

[66] J. Yang, X. Zhou, M. Chrobak, Y. Zhang, and L. Jin, "Dynamic thermal management through task scheduling," in *Proc. 8th International Symposium on Performance Analysis of Systems and Software*, 2008, pp. 191–201.

[67] J. E. Sergent and A. Krum, *Thermal Management Handbook*, New York: McGraw-Hill, 1998.

[68] M. R. Stan, K. Skadron, M. Barcella, W. Huang, K. Sankaranarayanan, and S. Velusamy, "HotSpot: A dynamic compact thermal model at the processor-architecture level," *Microelectronics Journal*, vol. 34, no. 12, pp. 1153–1165, 2003.

[69] T. Li and L. K. John, "Run-time modeling and estimation of operating system power consumption," *SIGMETRICS Perform. Eval. Rev.*, vol. 31, no. 1, pp. 160–171, 2003.

[70] "Intel 64 and IA-32 Architectures Software Developer's Manual," http://www.intel.com/products/processor/manuals/index.htm; accessed July 25, 2010.

VITA

Youngwoo Ahn received his B.S. degree in electrical engineering from Seoul National University, Korea in 1997 and his M.S. degree from Seoul National University, Korea in 1999. During 1999-2004, he worked as a research engineer at LG Electronics in Korea. He also worked as a researcher at Electronics and Telecommunication Research Institute in Korea from 2004 to 2005. He graduated with his Ph.D. in electrical and computer engineering from Texas A&M University in August 2010. His research interests lie primarily in real-time operating systems, especially in designing and analyzing task scheduling under resource constraints. He is also interested in low-power system designs. He may be contacted at:

Department of Electrical and Computer Engineering

Texas A&M University

TAMU 3112

College Station, TX 77843-3112

U.S.A.

Phone: (979) 209-9387

Email: youngwoo.ahn@gmail.com