SCHEDULING STRATEGIES FOR REAL-TIME TASKS IN THERMALLY CONSTRAINED PROCESSING ENVIRONMENTS

by

Rehan Ahmed

A dissertation submitted in partial fulfillment of

the requirements for the degree of

Doctor of Philosophy

(Electrical Engineering)

at the

UNIVERSITY OF WISCONSIN–MADISON

2015

Date of the final oral examination: 02/05/15

The dissertation is approved by the following members of the Final Oral Committee:

Parameswaram Ramanathan, Professor, Electrical and Computer Engineering
Kewal K. Saluja, Professor, Electrical and Computer Engineering
Mikko H. Lipasti, Philip Dunham Reed Professor, Electrical and Computer Engineering
Katherine Morrow, Associate Professor, Electrical and Computer Engineering
Yu Hen Hu, Professor, Electrical and Computer Engineering
Dan Negrut, Associate Professor, Mechanical Engineering

ACKNOWLEDGMENTS

Although, this dissertation is credited to me, I believe this is a culmination of efforts from a full range of individuals on many different and distinct fronts. I would like to take this opportunity to thank them.

First and foremost, I would like to express my deepest gratitude to my advisers Prof. Paramesh Ramanathan and Prof Kewal K.Saluja for their mentor-ship, guidance and thoughtful advice during the course of my graduate studies. This thesis would not have been accomplished without their support and guidance. I am very grateful to them for mentoring me during my doctoral studies.

I am also thankful to my committee members, Prof. Yu Hen Hu, Prof. Mikko H Lipasti, Prof. Katherine Morrow and Prof. Dan Negrut, for their valuable feedback and guidance on my work. My interaction with them has been an enriching experience.

During my studies here, I was fortunate to work under the guidance of Mark Allie as a teaching assistant. Apart from getting financial support, this experience has given me valuable insights into teaching methodology, ethics and the role of an instructor in a teaching environment. I will carry this experience with me during me academic career and am very grateful to Mark for being my mentor.

I would also like to give a shout-out to my grad school friends and colleagues: Spencer Millikan, Felix Lu, Ayoosh Bansal, Bhuvana Kakulooni, Chunhua Yao, Shoaib Altaf, Arslan Zulfiqar,

Syed Gilani, Vignyan Reddy and Atif Hashmi. Thank you for your help and support. My study at UW Madison has been a lively experience because of you.

No acknowledgement can be complete without expressing gratitude for my parents, Mohammad Saif Uddin Ahmed and Rehana Saif. That being said, no acknowledgement would do justice to their contribution in my life; as I am, and always intend to be, their reflection. I would also like to thank my brothers, Zeeshan and Aman, for their unwavering love and support. I was fortunate to find my life partner, Anum Rehan, during the course of my studies here. Her love and support have been invaluable for me and I look forward to sharing my life's journey with her.

Lastly, but by no measure the least, I want to thank God for giving me the ability to do this work and for being an unseen pillar of support in my life.

ABSTRACT

This dissertation is motivated by rapid and increasing rise in power density of modern processing platforms. High power densities cause thermal hotspots which degrade reliability, performance, and efficiency of a system. The research in this dissertation aims to alleviate these temperature related problems in the context of real-time computing (systems where tasks have deadlines). Traditionally, high temperature conditions are resolved by application of performance throttling schemes such as dynamic voltage and frequency scaling (DVFS) and decode throttling. However, these temperature management strategies cause performance degradation which may cause deadline violations in real-time systems. Depending on the criticality of a real-time system, deadline violations and other failures can have catastrophic consequences; including loss of life. Therefore, it is important to devise thermal management schemes specific to real-time systems; which is the subject of this dissertation.

Majority of the research on real-time systems during last four decades has focused on meeting deadline constraints of real-time tasks. Recently, work has been done on thermal constrained scheduling. However, most existing thermal aware schemes use a restrictive power model and employ DVFS to meet thermal constraints. More importantly, strong theoretical results for thermal feasibility of real-time tasks have been missing. In this dissertation, we counter these limitations and provide strong theoretical results pertinent to thermal feasibility of real-time tasks.

We propose the concepts of Thermal Impact/Utilization which are used to prove several important theoretical results. Specifically, we prove that thermal utilization of less than or equal to 1 is a necessary and sufficient condition for thermal feasibility of periodic real-time tasks on uni-core systems. This is similar to the computational feasibility condition proposed by Liu and Layland [1] in their seminal paper. Apart from periodic task scheduling, we also propose optimal scheduling algorithms for aperiodic tasks. In line with current architectural trends, the concept of Thermal Impact/Utilization is generalized to a multi-core processing platform and thermally optimal scheduling strategies for multi-core platforms are proposed. In contrast to other research, the scheduling strategies proposed in this work do not rely on DVFS to reduce system temperature. However, DVFS strategies are also proposed which enable further temperature reduction. We also evaluate the proposed multi-core solution on a hardware testbed. The theoretical results, simulations and hardware evaluations show that the proposed concepts can be used to significantly reduce system temperature while conforming to task deadline constraints. The application of the proposed concepts to other application domains, such as mobile and data-center computing, also has significant potential and can lead to improved user experience, system efficiency and operational costs.

TABLE OF CONTENTS

Page

ABSTRACT . . . iii

LIST OF TABLES . . . viii

LIST OF FIGURES . . . ix

1 Introduction . . . 1

2 Related Research . . . 9

2.1 Real-Time Applications and Scheduling . . . 9
2.2 Thermal Analysis . . . 12

3 Theoretical Foundation . . . 14

3.1 System Model . . . 14
3.1.1 Processor Model . . . 14
3.1.2 Thermal Model . . . 15
3.1.3 Simplified Uni-Core Model . . . 18
3.1.4 Task Model: Implicit Deadline Periodic Tasks . . . 20
3.2 Meeting Deadline Constraints for Implicit Deadline Periodic Tasks . . . 21
3.2.1 EDF Example . . . 22
3.3 Thermal Impact . . . 24
3.4 Necessary Condition for Thermal Feasibility . . . 26
3.5 Power Redistribution Algorithm . . . 29

4 Thermally Optimal Solution for Implicit Deadline Periodic Tasks on Uni-Core . . . 32

4.1 Generalized Processor Sharing as a Thermally Optimal Scheduling Discipline . . . 32
4.2 Worst-Case-Fair Weighted Fair Queuing . . . 35
4.3 Scheduling Example . . . 37
4.4 Validation Results . . . 38

Page

5 Thermal Extension of the Total Bandwidth Server . . . 43

5.1 Task Model: Aperiodic Tasks . . . 43
5.2 Total Bandwidth Server . . . 44
5.3 Thermally Constrained Total Bandwidth Server . . . 45
5.3.1 Theoretical Results . . . 46
5.3.2 Statistical Estimation of Response Time: . . . 48
5.3.3 Discretized T2BS . . . 49
5.4 Scheduling Example . . . 49
5.5 Results . . . 52

6 Thermal Utilization and Speed Scaling . . . 58

6.1 Thermal Utilization/Computation Utilization Tradeoff . . . 58
6.2 Thermal Utilization Minimization . . . 60
6.3 Simulation Results . . . 66

7 Thermal-Aware Multi-Core Scheduling . . . 71

7.1 Computational Feasibility . . . 72
7.2 Necessary Condition For Thermal Schedulability on Multi-Core . . . 74
7.2.1 Taskset Thermal Utilization . . . 76
7.3 Thermal Aware Multi-Core Scheduling . . . 78
7.3.1 TrUMPS . . . 79
7.3.2 TrUMPS^2 . . . 79
7.3.3 TrUMPS-C . . . 80
7.4 Results . . . 82
7.4.1 Simulation Setup . . . 83
7.4.2 Simulation Results - Fixed speed . . . 83
7.4.3 Simulation Results - Variable Speed . . . 86
7.5 Hardware Implementation/Analysis . . . 89
7.6 Computation Heterogeneity . . . 93

8 Generalized Multi-Core Scheduling Framework . . . 95

8.1 Non-Partitioned Scheduling of Periodic Tasks . . . 96
8.2 Adaptations of Scheduling Framework . . . 100
8.2.1 Constrained deadline periodic tasks . . . 100
8.2.2 Incorporation of Resource Constraints . . . 101
8.2.3 Incorporation of Precedence Constraints . . . 101

Appendix

Page

8.2.4 Per-Interval Speed Solutions 103
8.3 Results 104

9 Conclusion and Future Work 107

9.1 Dynamic reclaiming strategies 107
9.2 Dynamic Priority Exchange 108
9.3 Improved Thermal Models 108
9.4 Sufficient Conditions for WF2Q 109
9.5 Non-Preemptive Execution Model 109
9.6 Sporadic Task Scheduling 110
9.7 Mobile Computing 110
9.8 Data-Center Computing 111

APPENDICES

Appendix A: Proof of Theorem 6.3 112
Appendix B: Proof of Theorem 6.4 114

LIST OF REFERENCES 115

LIST OF TABLES

Table Page

1.1 Recent papers on real-time applications in RTSS, RTAS and ECRTS conferences considering thermal constraints. 5

3.1 Simple uni-core thermal model parameters. 19

5.1 Periodic/Aperiodic task parameters. 50

LIST OF FIGURES

Figure Page

1.1 Example schedule with two periodic tasks and one aperiodic task. 3

1.2 Core 1 temperature with/without aperiodic task execution on core 0. 3

3.1 Thermal model used by HOTSPOT. Figure taken from [2]. 16

3.2 Single RC thermal model. 18

3.3 RMA feasible processor utilization as a function of number of periodic tasks. 21

3.4 EDF schedule for 1 *hyperperiod*. 23

3.5 Executions of task τ. 25

3.6 Thermal profile of *Execution 1* and *Execution 2*. 25

3.7 Accumulated Thermal Impact of *Execution 1* and *Execution 2*. 26

3.8 Fraction of thermally feasible periodic tasksets under EDF and PRA scheduling policies. 30

4.1 EDF and WF2Q schedules for the example periodic taskset. 37

4.2 Steady state thermal profile for schedules shown in Figure 4.1 and GPS. 37

4.3 Computation and thermal utilization of periodic tasksets simulated; and their thermal feasibility with GPS scheduling policy. 39

4.4 Fraction of periodic tasksets which are thermally feasible using GPS, and EDF. Different periodic task frequencies simulated. 39

4.5 Fraction of thermally feasible periodic tasksets under WF2Q scheduling policy with various Execution Interval lengths. 41

Figure Page

4.6 Processor temperature as a result of schedule given by GPS, WF2Q, and EDF scheduling schemes for a periodic taskset. 41

5.1 Example schedule with two periodic and two aperiodic tasks for TBS and D-T2BS scheduling schemes. 50

5.2 Hyperperiod temperature profile for TBS and D-T2BS and T2BS scheduling schemes. 51

5.3 Response time for mixed aperiodic tasks with varying levels of average power consumption. 53

5.4 Mean response time, as a function of arrival rate and aperiodic task power consumption. 53

5.5 Maximum temperature, as a function of arrival rate and aperiodic task power consumption. 54

5.6 Histogram of temperature for T2BS and D-T2BS. 56

5.7 Histogram of response time difference between D-T2BS and T2BS. 56

6.1 Variation in computation and thermal utilization of a periodic task with speed. 59

6.2 Variation in computation and thermal utilization of a periodic tasks with speed. 61

6.3 Variation in thermal utilization of periodic taskset wrt task speeds. 61

6.4 Fractional reduction in thermal utilization of periodic tasksets after speed scaling. . . . 67

6.5 Thermal Utilization of Optimal, I-SeCTUM and Constant Speed schemes normalized to the thermal utilization without speed scaling. 68

6.6 Difference in thermal utilizations of I-SeCTUM and Constant Speed schemes from optimal speed assignment. 69

7.1 Computation and thermal utilization of periodic tasksets simulated; and their thermal feasibility using TrUMPS. 84

7.2 Difference in maximum temperature given by TrUMPS and the lower bound on temperature. 85

7.3 Maximum temperature given by the TrUMPS for all tasksets. 85

Appendix
Figure Page

7.4 Distribution of temperature differences between TrUMPS^2 temperature lower bound. . 86

7.5 Distribution of temperature differences between core speed assignment strategies and TrUMPS^2. 88

7.6 Difference in maximum temperature between core speed assignment strategies and TrUMPS^2. 89

7.7 Distribution of jitter in sleep/wake time of high priority threads. 90

7.8 Core temperatures when EDF and TrUMPS are used to schedule periodic tasks. 91

7.9 Histogram of temperature in steady state hyperperiod of EDF schedule. 91

7.10 Histogram of temperature in steady state hyperperiod of TrUMPS schedule. 91

7.11 Maximum temperature across all cores for EDF and TrUMPS scheduling schemes. . . 92

8.1 Interval assignment for the example periodic taskset. Minimum number of intervals. . 96

8.2 Intervals assignment for the example periodic taskset. Arbitrary interval assignment. . 97

8.3 Interval assignment for the constrained deadline periodic taskset. Minimum number of intervals. 100

8.4 Maximum temperature given by global scheduling scheme. 105

8.5 Difference in maximum temperature given by Global, TrUMPS and the lower bound on temperature. 106

Chapter 1

Introduction

Embedded computing systems are commonly used to monitor and control the dynamics of the underlying physical processes in many cyber-physical applications. The objectives are to enhance the performance, reliability, and safety of the applications. Examples include modern day automobiles, aerospace applications, smart electric grids, cyber-physical applications, monitoring health of patients, and next generation transportation systems. For instance, modern day automobiles contain embedded computing systems with 50–100 processors executing both critical and non-critical application tasks [3]. The processors in a typical Powertrain Control Module execute critical tasks that monitor and control key operational parameters such as engine speed and position, fuel injection settings, and the timing of next spark signal. Other processors may execute non-critical tasks such as those in the car's entertainment system. Both types of tasks usually have stringent real-time deadline constraints by which they must complete their execution. Failure to meet the deadline constraints may have severe consequences on performance, reliability, and/or safety of the vehicle.

Research over the past three decades has resulted in extensive theory and algorithms for meeting the real-time constraints of such embedded applications. The existing theory and the algorithms deal with different processing models (single core, multi-core, or multiprocessor), different application models (preemptive, non-preemptive, tasks with resource constraints, etc.), and different timing guarantees (hard, firm, or soft). However, certain important emerging challenges of modern day processors are not well-addressed by these extensive theory and algorithms.

One such emerging challenge is the need for thermal management in next generation processing systems. As technology scales, on-chip power consumption and power density are increasing rapidly. International Technology Roadmap for Semiconductors (ITRS) predicts that power densities will more than double over the next decade [4]. As power consumption and density increase, their on-chip spatio-temporal variations will create thermal hot spots that are detrimental to both processor performance and reliability. For example, studies have shown that a 10-15°C increase in temperature can more than double the probability of failure of the underlying semiconductor device [5]. In addition, increase in on-chip temperature also causes increase in device leakage currents, which increases on-chip power consumption, which in turn, increases the temperature further. This positive feedback between temperature and leakage currents can cause "thermal runaway" which is extremely detrimental to processor performance and reliability [6]. The challenge, therefore, is to schedule and complete the application tasks in such a way that the peak on-chip temperature stays below a pre-determined threshold and/or is kept as small as possible. To further reduce leakage currents, additional objectives may include minimizing the spatio-temporal average of the on-chip temperatures.

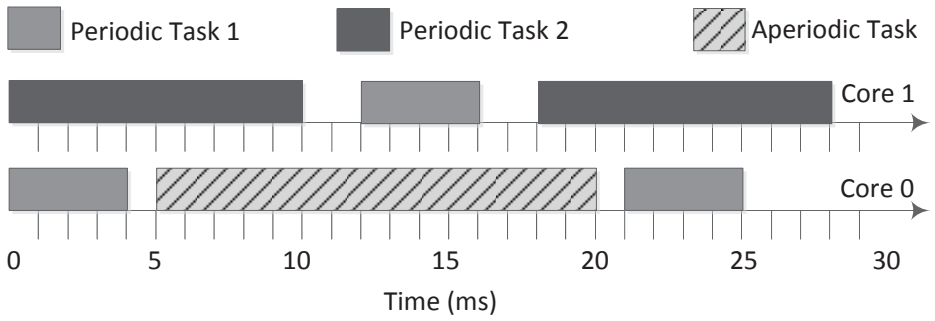


Figure 1.1: Example schedule with two periodic tasks and one aperiodic task.

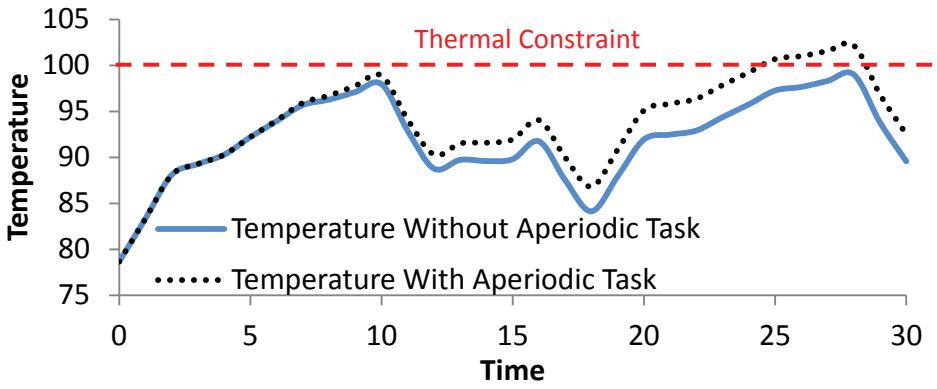


Figure 1.2: Core 1 temperature with/without aperiodic task execution on core 0.

Traditional approaches for on-chip thermal management often rely on scaling down the processor speed (through a combination of voltage and frequency scaling) to reduce its power consumption and thereby temperature. Since scaling processor speed increases the time required to complete a computation, meeting the real-time constraints of embedded applications becomes more challenging. The challenge is even more acute for multi-core embedded applications where effects such as *inter-core thermal interactions* and *thermal inertia* complicate the thermal constrained scheduling problem. These effects and their implications are explained below with an example.

Scheduling Example: Consider a processing system with two cores executing a real-time application with two periodic tasks. Periodic tasks form an infinite sequence and repeat after a fixed time period. The periodic task 1 has computation time of 4 time units and a period of 10 time units. Periodic task 2 has a computation time of 10 time units and a period of 15 time units. The application also consists of an aperiodic task with a computation time 15 time units. The schedule for both periodic and aperiodic tasks is shown in Figure 1.1. Figure 1.2 plots the temperature of core 1 for the following two scenarios.

(a) Aperiodic task is not executed. Periodic tasks are executed as shown in Figure 1.1.

(b) Aperiodic task is scheduled and executed as shown in in Figure 1.1

In the first case, where aperiodic task is not executed, the temperature profile of core 1 is low as shown by the solid line curve in Figure 1.2. However in the second case, when the aperiodic task is executed on core 0 from time 5 to time 20, core 1 temperature increases as shown by black dashed curve in Figure 1.2. The temperature increase of core 1 due to execution on core 0 is due to the *thermal interaction* between core 0 core 1. Furthermore, the temperature of core 1 remains high well after the aperiodic task finishes execution on core 0 (at time 20). This is because of *thermal inertia* (the delay in temperature increase on core 1 due to execution on core 0). Both of these factors complicate the thermal constrained scheduling problem. In this particular case, assuming a thermal constraint of 100°C, the execution of aperiodic task on core 0 causes thermal violation on core 1. Furthermore, thermal violation occurs well after the aperiodic task finishes execution (at and after time 25). Therefore, in a multi-core platform, while watching out for possible thermal

Table 1.1: Recent papers on real-time applications in RTSS, RTAS and ECRTS conferences considering thermal constraints.

Application	Processor	
Periodic	Single	[7][8][9] [10][11][12][13]
Periodic &Aperiodic		[14]
Periodic & Sporadic		[15]
Periodic	Multi	[16][17][18] [19][20] [21][22] [23]
Periodic & Aperiodic		[24]
Periodic & Sporadic		[22] [23]

violations during the execution of given task; we also need to consider the thermal effect of the execution well after the execution finishes.

Recently, thermal constrained scheduling problem has gotten attention and Table 1.1 categorizes state of the art recent works in this field that have appeared in premier real-time conferences. The scope of these works and their limitations are discussed in Chapter 2. Broadly speaking most of the exiting literature assumes a very simple constant power model and relies on speed scaling to manage temperature. This research attempts to address these issues.

The research proposed in this dissertation does not have the limitations of constant power model and high reliance on speed scaling. One of the foremost contributions of this research is the derivation of necessary and the sufficient conditions for schedulability of thermally-constrained embedded applications. For example, for single-core systems without thermal constraints, the

Liu and Layland schedulability bound is a seminal result that has been widely used to provide real-time guarantees [1]. Since then, widely used schedulability bounds have been derived for multiprocessor systems and a variety of application models [25]. However, such bounds do not exist for thermally-constrained real-time scheduling. The concept of thermal utilization, proposed in this research, plays an analogous role to the traditional computational utilization for dealing with thermal constraints. We propose thermally optimal scheduling algorithms for several real-time application and task models. Specifically, this thesis makes the following vital contributions:

a) Necessary Schedulability Conditions: We propose the concept of Thermal Utilization and prove that having thermal utilization ≤ 1 is a necessary condition for thermal feasibility of implicit deadline periodic tasks on uni-core systems. We also propose a lower bound for processor temperature for a given periodic taskset.

b) Sufficient Schedulability Conditions and Optimal Scheduling: We prove that Thermal Utilization of ≤ 1 is a sufficient condition for schedulability of implicit deadline periodic tasks on thermally constrained uni-core systems. We also prove that a scheduling algorithm inspired from Generalized Processor Sharing [26] is thermally optimal.

c) Aperiodic task Scheduling: We propose an optimal algorithm for the scheduling of aperiodic tasks along with periodic tasks on thermally constrained uni-core systems. The proposed algorithm is based on the Total Bandwidth Server [27] concept and is optimal in terms of being able to support maximum rate of aperiodic arrivals.

d) Speed Scaling Strategies: The optimal scheduling algorithms proposed in this dissertation do

not rely on speed scaling. However, speed scaling strategies are proposed that enable further reduction of processor temperature.

e) Multi-Core Scheduling: We extend the concepts of thermal utilization to a multi-core platform and prove necessary conditions for thermal schedulability. We also provide thermally optimal partitioned solutions for the scheduling of implicit deadline periodic tasks on multi-core. The proposed multi-core solution is also implemented and evaluated on a hardware testbed.

f) Executing Other Task Models: We present a general scheduling framework that can be used to schedule other task, constraint and execution models. The extensions explained include: scheduling of constrained deadline periodic tasks, introduction of resource constraints, and introduction of precedence constraints.

Organization: Chapter 2 overviews the research in the field of thermal constrained scheduling and thermal analysis. Chapter 3 covers the system, task and thermal models assumed in this work. It also introduces the concept of Thermal Impact/Utilization and presents necessary conditions for thermal schedulability of periodic real-time tasks on uni-core. Optimal scheduling algorithm for periodic real-time tasks on uni-core, along with a necessary and sufficient thermal schedulability condition is presented in Chapter 4. Chapter 5 covers an optimal scheduling strategy for aperiodic tasks in the presence of hard headline periodic real-time tasks. Chapter 6 explains how the concepts of thermal impact/utilization can be extended to accommodate speed scaling strategies. Periodic task scheduling problem is generalized to multi-core platform and a partitioned scheduling solution is proposed in Chapter 7. This chapter also includes the simulation results for multi-core solution

and the evaluation of proposed multi-core solution on a hardware testbed. A general scheduling framework which can be used to do thermal aware scheduling for other task models is explained in Chapter 8 followed by conclusion and future work.

Chapter 2

Related Research

2.1 Real-Time Applications and Scheduling

In this chapter we discuss the research work dealing with thermal aware scheduling and its limitations. As mentioned before, Table 1.1 in the introduction chapter succinctly categorizes recent work from literature on thermal-aware scheduling in real-time systems. In particular, the table categorizes papers published in the years 2004–2013 in three premier conferences (Real-Time Systems Symposium, Real-Time Technology and Applications Symposium, and Euromicro Conference on Real-Time Systems) based on assumed models.

Most of the existing works in uni-core context rely on scaling down the processor speed (through a combination of voltage and frequency scaling) to reduce its power consumption and thereby temperature. They can be broadly subdivided into either reactive schemes [10][12] or proactive schemes [7][28][9]. Reactive schemes reduce processor speed only when temperature gets above a certain threshold. Wang et al. perform schedulability analysis for a two speed reactive speed scaling scheme in [10]. They also perform delay analysis in [12]. Their scheme is improved upon by Chen et al. [7] by using proactive speed scheduling. In proactive schemes, apart from only reacting to a thermal triggers, speed can also be changed *proactively* by requests issued by

task/scheduler. Chen et al. also derive approximation bounds on the maximum temperature for Earliest Deadline First (EDF) schedule compared to a thermally optimal schedule in [19]. The 2-speed reactive solution is also improved upon by Chaturvedi et al. [29] who use an m-oscillating scheme to schedule periodic tasks. The processor speed oscillates between m levels to keep the system temperature below threshold. Thermally constrained scheduling of periodic tasks is also studied by Quan et al. [28][9]. They derive thermal feasibility checks and also formulate constructive speed scheduling algorithms for periodic tasks under thermal constraints.

Among existing schemes that consider multi-core model, Chantem et al. [30] use integer linear programming to minimize the maximum temperature for a given set of real-time tasks. They use an equivalent circuit model to estimate the transient core temperatures. Fisher et al. [22] use the thermal characteristics of processing cores to derive preferred speeds for processing elements in a homogeneous multi-core system to minimize system temperature. Hung et al. [31] focus on real-time task scheduling for processing elements without dynamic voltage scaling capability to target maximal power saving in a heterogeneous multi-core system. Liu et al. [32] propose a two-stage approach, in which they first utilize the retiming and software pipelining technique to transform a periodic dependent task graph into a set of independent tasks, and then iteratively adjust the task schedule and voltage selection by incorporating the dynamic voltage scaling and dynamic power management techniques.

In [17][18], Schor et al. perform worst case temperature analysis for real-time tasks running on multi-core systems. They use a more generalized task model by using the concepts of real-time calculus [33] to model arrivals and computation demands of real-time tasks. The worst case

computation methodology is further used in [21] to perform task and frequency assignment for real-time tasks executing on multi-core systems. The problem is formulated and solved as a binary optimization problem. Kumar et al. [13] perform worst case delay analysis on a stream of jobs following arrival and computation patterns based on real-time calculus. In their work, Hettiarachchi et al. [15] propose a new resiliency parameter for real-time systems with thermal constraints. Their work focuses on the design of real-time systems that operate in unpredictable thermal environments where the ambient temperature can change. This work is extended to multi-core processing environment in [23]. In [20], Yun et al. consider scheduling of periodic tasks on a multi-core system under soft thermal constraints. They design a proactive peak temperature manager which periodically estimates peak processor temperature. It uses machine learning to predict high temperature on a given core. If a core is predicted to overheat, dynamic power management techniques are applied to cool down the core without violating task timing constraints. Kumar et al. [16] address the problem of minimizing end-to-end delay on a set of real-time tasks with inter-dependencies; on a thermally constrained distributed system. They prove that the delay is minimized if all tasks are executed *as soon as possible* under the system thermal constraints.

Specifically, all of the mentioned works have the following primary limitations which are addressed by this work:

1. Majority of works consider dynamic power consumption to be only a function of processor speed. Therefore, it is assumed that power consumption is independent of the task executing on the system. This is clearly a restrictive power model, considering that different applications exhibit vastly different power characteristics [34]. Therefore, to guarantee *hard* thermal

constraints, the schemes which consider no power variation across tasks have to schedule tasks based on the *worst-case* power/thermal behaviour; which may be very pessimistic.

2. Except for [29][30], all of the schemes rely on speed scaling to manage temperature. Therefore, these schemes cannot be applied to systems which are not capable of Dynamic Frequency Scaling or where speed change overheads are high.

3. None of the existing works present necessary and sufficient conditions for thermal schedulability of periodic tasksets.

2.2 Thermal Analysis

The existing research on characterizing the thermal profile of ICs can be broadly classified into three categories. The first category focuses on detailed accurate characterization of temperature variations in the IC. The associated algorithms divide the IC into a discrete grid of suitable size and solve a linear system of equations characterizing temperature relationships at the grid points. Papers differ on the solution methodology used to accurately solve these system of equations as fast as possible [35, 36, 37, 38]. There are two major drawbacks of this type of thermal analysis: (i) they are impractical for runtime use in a scheduler because finding a good schedule requires multiple runs of thermal analysis, and (ii) they depend on a detailed characterization of power consumption profile within the IC, which is often difficult to obtain.

The second category relies on an architectural level simulation to characterize temperature variations among different functional blocks in the processor. For example, Hotspot [39] thermal analysis tool solves a linear system of equations but at very coarse-grain. It requires power

consumption information for each functional block, and temperature difference equations are modelled at functional block level given their relative location in the IC floorplan. Studies have shown that tools such as Hotspot are fairly accurate at architectural level thermal profile characterization. Unfortunately, thermal simulations are also computationally expensive and hence cannot easily be performed by a scheduler at runtime.

Finally, the third category is tailored for use in schedulers [7, 8, 10, 9, 12, 32, 31, 22]. Methods in this category rely on simple analytic models to characterize the thermal impact of executing a given task. The underlying models usually rely on a simple uniform power consumption assumption. These methods sacrifice accuracy for conservativeness and speed. As a result, the scheduling decisions are usually pessimistic, e.g., a scheduler may excessively delay an aperiodic real-time task due to the conservative processor temperature estimates.

In this work, we use the second and third model for all evaluations/simulation results. However, a specific model is not necessary for the applicability of results presented in this work. As will be explained in later chapters, the only assumption we make for the thermal model is that it is linear in terms of power consumption of tasks. All three methodologies for temperature analysis/evaluation satisfy linearity assumption. Furthermore, the proposed schemes do not compute temperature at runtime. Therefore, there is no temperature computation overhead incurred by the scheduler.

Chapter 3

Theoretical Foundation

In this chapter, we first specify the system, task and execution model assumed in this dissertation. That is followed by introduction of the concept of thermal impact. Thermal impact is the fundamental concept on which several theoretical results in this dissertation are based. In this chapter, Thermal Impact/Utilization is used to find a *Necessary Condition* for thermal schedulability of implicit deadline periodic tasks on uni-core. This implies that:

For a given periodic taskset executing on a uni-core, if the condition presented in this chapter is not satisfied; then no scheduling algorithm will be able to meet thermal constraints of the system

3.1 System Model

3.1.1 Processor Model

The processor has a set M of m identical cores for some $m \geq 1$. Some of the results in this dissertation are specific to uni-core system, i.e, $m = 1$. However, several results are applicable and effective on multi-core systems, i.e, where $m > 1$. We define $\widehat{\Delta}$ as the thermal constraint of

The work covered in this Chapter appeared in [11] and [40]
Rehan Ahmed, Parameswaran Ramanathan, Kewal K. Saluja, "On thermal utilization of periodic task sets in uni-processor systems" RTCSA 2013
Rehan Ahmed, Parameswaran Ramanathan, Kewal K. Saluja, "Temperature Minimization Using Power Redistribution in Embedded Systems", VLSI Design 2014

the processor. The temperature of each core m has to be less than $\widehat{\Delta}$ for safe operation. The power consumed by a core has two components: (i) dynamic power due to computational activities of a task, and (ii) leakage power due to leakage currents. Studies have shown that leakage currents are a strong function of temperature. However, within a typical operating temperature of a processor, power due to leakage currents can be adequately modelled as a linear function of temperature [41]. As a result, if task τ is executing on core i at time t, then the total power consumed by core i at time t can be written as

$$P_{\text{tot},i}(t, \Psi(t)_i) = P_\tau + \delta\Psi(t)_i + \rho, \tag{3.1}$$

where P_τ is the dynamic power due to computational activities of task τ, $\Psi(t)_i$ is the temperature of core i at time t and where ρ and δ are modelling constants. Each core is assumed to be DVFS capable with speed varying between $[s_{\min}, s_{\max}]$. Furthermore, it is assumed that voltage/frequency of each core can be modified independently.

3.1.2 Thermal Model

The power consumption on a core causes its temperature to increase. Due to thermal coupling between them, the power consumption on a core also causes temperature increases on other cores of a multi-core system. Studies have shown that thermal effects of power consumption can be modelled as linear resistance-capacitance (RC) circuits [39][22][30]. Accurate thermal models of multi-core processors typically include multiple discrete thermal elements. For example, HOTSPOT [39], uses the thermal model shown in Figure 3.1 which models individual architectural units, heat spreader, interface layer, heatsink and cooling due convection. In this model,

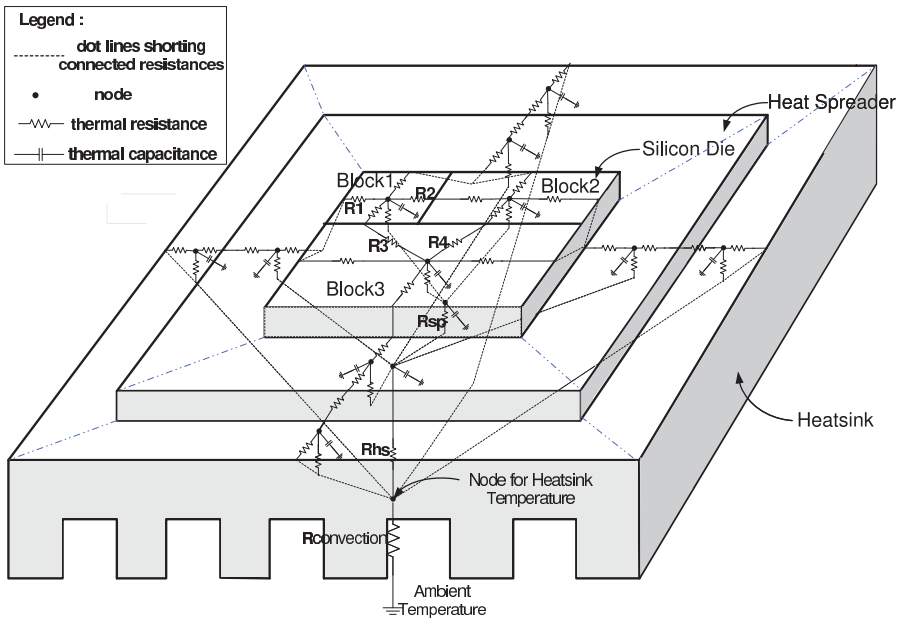


Figure 3.1: Thermal model used by HOTSPOT. Figure taken from [2].

Power consumption is modelled as current and temperature is analogous to voltage. Transfer of heat between different elements is modelled as current passing through a *thermal resistor*. The delay in increase of temperature for a given element is modelled by a *thermal capacitance*. Using the well-known Fourier law, thermal behaviour of a processor can be written as set of differential equations, which in turn can be succinctly rewritten in matrix form as follows.

$$\boldsymbol{C} \circ \boldsymbol{\Psi}(t)' = \boldsymbol{P_{\mathbf{tot}}}(t) - \boldsymbol{A}\boldsymbol{\Psi}(t), \tag{3.2}$$

where $\boldsymbol{C}$ is a vector characterizing the thermal capacitances of different elements in the model, $\boldsymbol{\Psi}(t)$ is a vector of temperature at time t of all the thermal elements in the model, $\boldsymbol{P}(t)$ is a vector of total power consumption of the different elements in the model at time t, and $\boldsymbol{A}$ is matrix which is constructed using the thermal conductances between the different elements in the model in the

following manner:

$$A_{i,j} = \begin{cases} -G_{i,j} & \text{if } i \neq j \\ \sum_{j} G_{i,j} & \text{if } i = j \end{cases} \tag{3.3}$$

$G_{i,j}$ is the thermal conductance between elements i and j and $G_{i,i}$ is assumed to be 0.

Let $\mathbf{\Psi}_{\mathrm{idle},ss}$ denote a vector of steady state temperatures when all cores are idle, i.e., when the dynamic power consumption on every core is zero. Note that the power consumption of each core is still non-zero because of static power component. Define a new variable $\mathbf{\Theta}(t) = \mathbf{\Psi}(t) - \mathbf{\Psi}_{\mathrm{idle},ss}$. In other words, $\mathbf{\Theta}(t)$ is a column vector of temperature at time t of all the thermal elements in the model expressed as a relative increase with respect to the idle steady-state temperatures. Following a few algebraic simplifications, one can show that

$$\boldsymbol{C} \circ \mathbf{\Theta}(t)' = \boldsymbol{P_{dyn}}(t) - (\boldsymbol{A} - \delta \boldsymbol{I})\mathbf{\Theta}(t). \tag{3.4}$$

Note that, the above equation only contains dynamic power consumptions. Leakage power is accounted for by expressing temperature as relative increase with respect to idle steady-state temperatures and subtracting δ from diagonals of A matrix. For simplicity of presentation, the rest of this dissertation assumes that all temperatures are expressed relative to the idle steady-state temperatures unless specified otherwise. The thermal constraint of the processor relative to steady-state temperature is a vector $\mathbf{\Delta}$ such that:

$$\mathbf{\Delta} = \widehat{\Delta} \cdot \mathbf{1} - \mathbf{\Psi}_{\mathrm{idle},ss}$$

where $\mathbf{1}$ is a column vector in which all entries are 1.

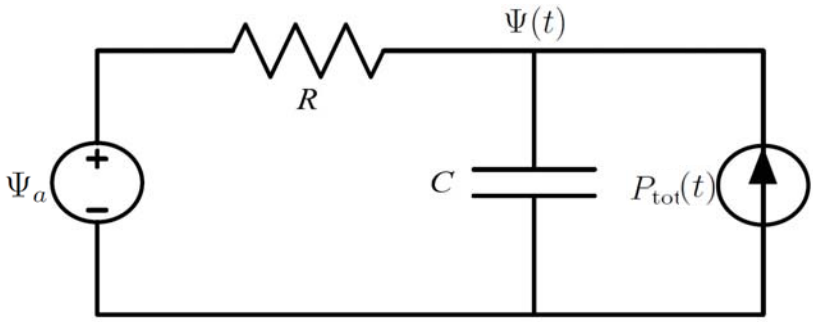


Figure 3.2: Single RC thermal model.

3.1.3 Simplified Uni-Core Model

For some portions of this thesis (specifically the results for the uni-core section and examples), we will revert to a simple uni-core model where the processor is modelled as a single resistance-capacitance pair as shown in Figure 3.2. In this model, we can compute temperature by the following differential equation:

$$C\Psi(t)' = P_{\text{tot}}(t) + \frac{\Psi_a - \Psi(t)}{R} \tag{3.5}$$

Where Ψ_a is the ambient temperature. The idle steady state temperature ($\Psi_{\text{idle,ss}}$) in this case can be computed by setting $\Psi(t)' = 0$ and $P_{\text{tot}}(t) = \delta\Psi(t) + \rho$: $\Psi_{\text{idle,ss}} = \frac{R\rho+\Psi_a}{1-R\delta}$. For notational brevity, let $\beta = \frac{1}{RC} - \frac{\delta}{C}$ and, $\Theta(t) = \Psi(t) - \frac{R\rho+\Psi_a}{1-R\delta}$. Following few algebraic simplifications, Equation 3.5 simplifies to:

$$C\Theta(t)' = P_{\text{dyn}(t)} - \beta\Theta(t) \tag{3.6}$$

R	Thermal Resistance	0.36
C	Thermal Capacitance	0.8
δ	As defined in Equation 3.1	0.001
ρ	As defined in Equation 3.1	0.1
Ψ_a	Unadjusted ambient temperature	40°C
$\widehat{\Delta}$	Unadjusted threshold temperature	75°C
β	Cooling coefficient in Equation 3.7	3.471
ζ	Unit Thermal Impact	0.36
$\Psi_{\text{idle,ss}}$	Unadjusted idle steady state temperature	40.05°C
Δ	Threshold temperature	35°C

Table 3.1: Simple uni-core thermal model parameters.

Assuming that power is constant in the interval [0,t), the differential equation has the following solution:

$$C\Theta(t) = \frac{P}{\beta}(1 - e^{-\beta t}) + \Theta(0)e^{-\beta t} \tag{3.7}$$

Equation 3.7 will be used extensively in this document to evaluate processor temperature. In this simplified uni-core model, modelling parameters and corresponding thermal model variables given in Table 3.1 will be used for computation of processor temperature

3.1.4 Task Model: Implicit Deadline Periodic Tasks

The initial chapters of this dissertation deal with the scheduling periodic, preemptive, independent tasks with implicit deadlines. Let Γ denote the set of tasks in the application. Each task τ is characterized by three parameters: (i) worst-case computation time C_τ, (ii) period T_τ, and worst-case computational activity a_τ. Furthermore, all tasks are assumed to be in-phase. Each task instance also has a hard deadline equal to the arrival time of the next instance of the corresponding periodic task (implicit deadline). Due to the computational activity of a task, the processor consumes a certain amount of dynamic power whenever the task executes. The dynamic power consumed by a task instance is assumed to be proportional to its worst-case computational activity. In general, the dynamic power consumed by a task instance is also a strong function (typically cubic) of the speed of the processing core at which the execution occurs. We assume that C_τ is the worst-case computation time of the task τ at speed 1 and worst-case computation time is inversely proportional to speed of execution i.e. computation time at speed s_τ is C_τ/s_τ. The worst-case dynamic power consumption of task τ at speed s_τ is:$P_\tau = a_\tau s_\tau^3$.

In this work Π represents the periodic schedule. The value of $\Pi(t)$ represents a 2 element tuple with elements representing the task being executed at time t and the speed at which it is being executed. For a multi-core system, $\mathbf{\Pi}(t)$ is a vector with tuples representing the task/speed values of each core. The length of schedule Π is one hyperperiod (L); where hyperperiod is defined as the Least Common Multiple (LCM) of all periodic tasks in Γ. In this work, we assume that the periodic schedule is *cyclic* i.e. $\mathbf{\Pi}(t) = \mathbf{\Pi}(t + kL)$ for all t and all integers k.

3.2 Meeting Deadline Constraints for Implicit Deadline Periodic Tasks

The seminal work for the scheduling of periodic tasks was done in 1973 by Liu and Layland [1]. The key contributions of this work was the formulation of optimal algorithms for computational feasibility of implicit deadline periodic tasks. Liu and Layland proposed two important scheduling algorithms. First of these algorithms was a *static priority* algorithm called Rate Monotonic Algorithm. In this algorithm periodic tasks were assigned priorities proportional to the execution rates ($\frac{C_\tau}{T_\tau s_\tau}$). A higher priority task is allowed to preempt execution a lower priority task. The authors prove in [1] that:

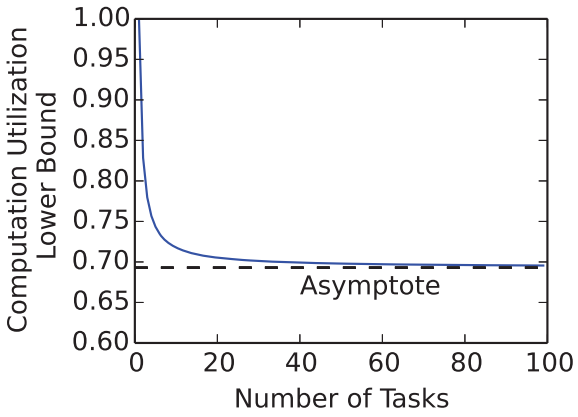


Figure 3.3: RMA feasible processor utilization as a function of number of periodic tasks.

Theorem 3.1. ***(Theorem 2 from [1])****If a feasible priority assignment exists for some task set, the rate monotonic priority assignment is feasible for that task set.*

This implies that RMA is optimal static priority assignment algorithm for implicit deadline periodic tasks on uni-core. The authors also prove that:

Theorem 3.2. ***(Theorem 5 from [1])*** *If a set of n tasks are scheduled in fixed priority order, then RMA will guarantee all deadlines if processor utilization $U \leq n(2^{1/n} - 1)$.*

Note that the bound in Theorem 3.2 is a *sufficient condition* and not a *necessary condition* i.e. if $U > n(2^{1/n} - 1)$, then RMA may still be able to meet all deadlines for periodic tasks in Γ. The expression $n(2^{1/n} - 1)$ has an asymptotic value of 0.693 (as number of periodic tasks reaches ∞) and it is plotted in Figure 3.3 as a function of number of periodic tasks.

To achieve higher processor utilization, Liu and Layland also propose a *dynamic priority* algorithm called Earliest Deadline First (EDF). As the name suggests, in EDF, a task instance is assigned priority inversely proportional to its deadline (task instances with earlier deadlines are given higher priority). It is proven in [1] that:

Theorem 3.3. ***(Theorem 7 in [1])*** *For a given set of taskset Γ, the Earliest Deadline First scheduling algorithm is feasible if and only if:* $\sum_{\tau \in \Gamma} \frac{C_\tau}{T_\tau} \leq 1$

Note that this is a *necessary and sufficient* condition implying that if $\sum_{\tau \in \Gamma} \frac{C_\tau}{T_\tau} > 1$, then no scheduling algorithm can meet deadlines of all periodic tasks. Therefore EDF is *optimal* for meeting deadlines of implicit deadline periodic tasks on uni-core.

In the context of this dissertation, EDF is a very important scheduling algorithm. The thermal aware scheduling schemes presented in this dissertation are compared against EDF as a baseline. Therefore, in the interest of reader's understanding, we now present a scheduling example that illustrates the functioning of EDF.

3.2.1 EDF Example

The periodic taskset considered constitutes two periodic tasks:

Task 1 ($C_1 = 0.1, T_1 = 0.25$)

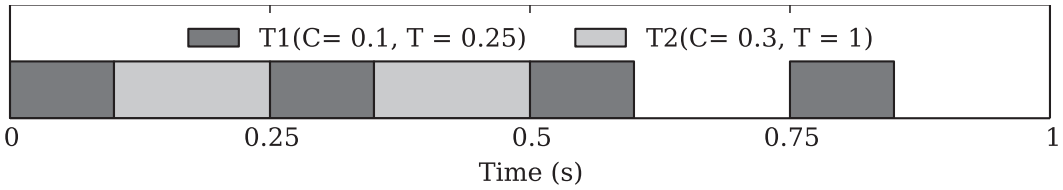


Figure 3.4: EDF schedule for 1 *hyperperiod*.

Task 2 ($C_2 = 0.3, T_2 = 1$)

In this example, we use $J_{i,j}$ to represent j^{th} instance of periodic task i, $D_{i,j}$ to represent its absolute deadline. The EDF schedule for one hyperperiod is shown in Figure 3.4. $J_{1,1}$ and $J_{2,1}$ arrive simultaneously at time 0. $D_{1,1} = 0.25$ and $D_{2,1} = 1.0$. Since $D_{1,1} < D_{2,1}$, $J_{1,1}$ is scheduled for execution starting at time 0 until its completion at time 0.1. At time 0.1, since task $J_{2,1}$ is the only ready task, it starts execution. However, at time 0.25 when $J_{2,1}$ has completed 0.15 out of 0.3 units of its execution, $J_{1,2}$ arrives with $D_{1,2} = 0.5$. Since $D_{1,2} < D_{2,1}$, $J_{2,1}$ is preempted and $J_{1,2}$ executes in the interval [0.25,0.35]. After completion of $J_{1,2}$ at time 0.35, $J_{2,1}$ executes and completes its execution at time 0.5. After this $J_{1,3}$ and $J_{1,4}$ are executed in intervals [0.5, 0.6] and [0.75, 0.85] respectively. $J_{1,5}$ and $J_{2,2}$ arrive simultaneously at time 1.0. 1.0 is the hyperperiod (LCM(0.25, 1)=1) and entire EDF schedule from in the interval [0, 1.0) repeats in the interval [1.0, 2.0) and all future hyperperiods.

Even though EDF is optimal in terms meeting deadline constraints for periodic tasks, it is not a *Thermal Aware* scheduling algorithm. Therefore, EDF cannot be used to safely execute real-time tasks in a thermally constrained system. In the next sections of this chapter, we will cover the theoretical foundation used to devise thermal aware scheduling algorithms.

3.3 Thermal Impact

Thermal Impact gives a measure of the total temperature effect, execution of a given task will have on a system. Given a task τ with computation time C_τ; suppose that $\eta_\tau(t)$ represents the processor temperature at time t when τ is executed without preemption starting at time 0 with $\Theta(0) = 0$. *Accumulated Thermal Impact* of task τ at time t ($\mathrm{ATI}_\tau(t)$) is defined as the cumulative thermal effect until t. Specifically:

$$\mathrm{ATI}_\tau(t) = \int_0^t \eta_\tau(u)du \tag{3.8}$$

Total Thermal Impact of task τ is defined as the "net" thermal effect of execution of τ.

$$\mathrm{TTI}_\tau = \lim_{t\to\infty} \mathrm{ATI}_\tau(t) \tag{3.9}$$

The concept of thermal impact is particularly useful in preemptive scheduling of tasks. This is because, due to linearity of RC thermal model, the value of TTI_τ remains unchanged after preemption of task at arbitrary number of points. To understand these concepts further, consider the following example:

Consider a task τ with computation time of 1s. In this example, we will consider the temperature effect of two possible executions of τ. As shown in Figure 3.5, in *Execution 1*, τ is executed contiguously between time 0 and time 1 sec. In *Execution 2*, τ is preempted once before completing execution at time 1.5s. The thermal profile of τ for the two executions is shown in Figure3.6. Based on the two thermal profile, the thermal impact of τ can be evaluated and is shown in Figure 3.7. Note that, as expected, two executions have different thermal profiles. Consequently, the value of the accumulated thermal impact also varies temporally. However, note that the value of

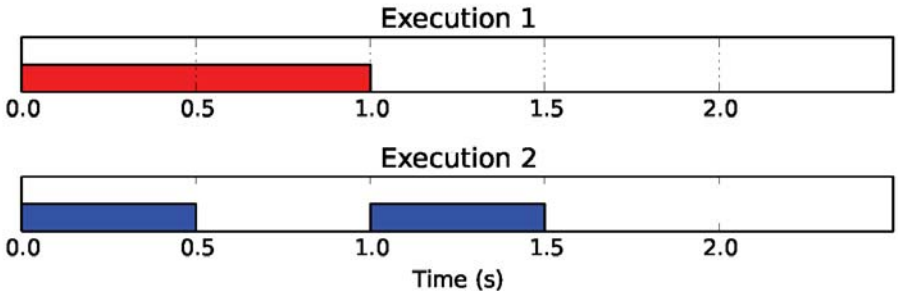


Figure 3.5: Executions of task τ.

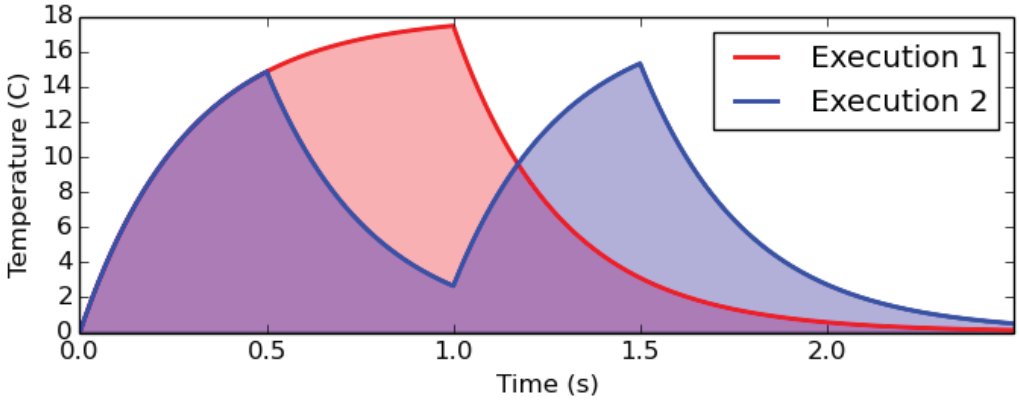


Figure 3.6: Thermal profile of *Execution 1* and *Execution 2*.

accumulated thermal impact saturates after some time since the cooling temperature slowly decays to 0. This value is the *Total Thermal Impact*. As illustrated in Figure 3.7, the value of Total Thermal Impact for the two executions is same. We will now use this basic result to find a general expression for Total Thermal Impact and see how it can be used to schedule periodic real-time tasks.

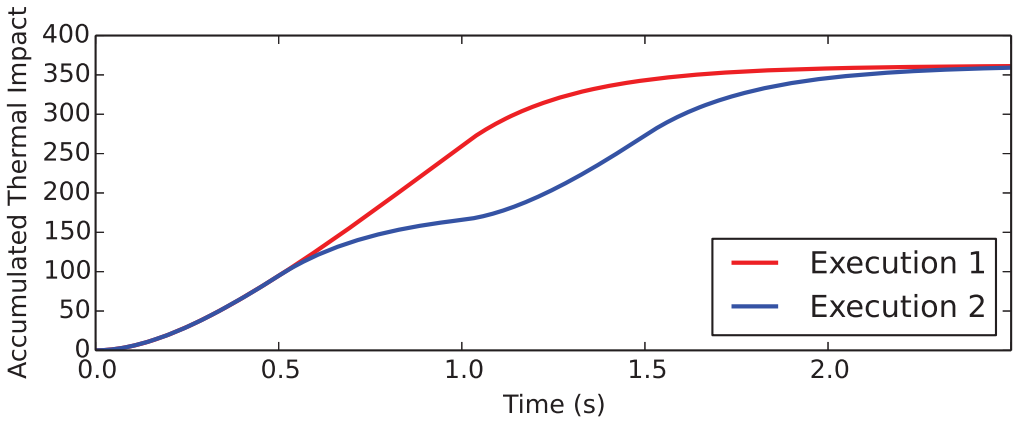


Figure 3.7: Accumulated Thermal Impact of *Execution 1* and *Execution 2*.

3.4 Necessary Condition for Thermal Feasibility

Definition 3.4. *Suppose that a single instance of a unit power task (Dynamic power consumption = 1W) executes for one second on a uni-core processor starting at an initial reference temperature of 0°C. Its power consumption will cause the temperature of the core to increase. Let* $\theta(t)$ *denote the temperature of the core at time* t *as a result of this execution. Then,* Unit Thermal Impact *(UTI) of the core is defined as:*

$$\zeta = \int_0^\infty \theta(t)dt.$$

Note that, due to the linearity of the system, if a single instance of a task with power P executes for C seconds on the core, then its *Total Thermal Impact* on the core will be $P \cdot C \cdot \zeta$. Similarly, due to linearity of the system, it follows from superposition principle that total thermal impact of executing more than one instance is the sum of the total thermal impacts of each instance.

Thermal Steady State: As defined in Section 3.1 let $\Pi(t)$ represent the cyclic periodic schedule. Also, let $\eta_{\Pi}(t)$ represents the thermal profile of the periodic schedule $\Pi(t)$. It was proved in [28, 11, 24] that if we repeatedly execute a periodic schedule Π, the temperature at the start of the hyperperiod increases monotonically until the temperature at the start and end of hyperperiod are approximately equal. This scenario where the temperature at the start of hyperperiod stops increasing is called *thermal steady state*. We call the temperature at the start of hyperperiod at thermal steady state Θ_{Π}.

Before stating the theoretical results in this section, we define the following function

$$\mathrm{Cool}(\Theta_{\mathrm{init}}, t) = \Theta(t) \text{ Where } \Theta(0) = \Theta_{\mathrm{init}} \text{ and processor is } \textit{idle} \text{ for } t > 0 \tag{3.10}$$

Theorem 3.5. *Suppose that all tasks are executed as per* $\Pi(t)$ *and* $\Theta(0) = \Theta_{\Pi}$*, then*

$$\int_0^L \Theta(t) = L \cdot \zeta \sum_{\tau \in \Gamma} \frac{P_\tau C_\tau}{T_\tau s_\tau}, \tag{3.11}$$

where s_τ *is the speed at which all instances of task* τ *are executed.*

Proof: given a periodic task τ with period T_τ, the hyperperiod will have L/T_τ instances of τ. Let the start time of the instance i be $\mathrm{st}_{\tau,j}$. The integral of processor temperature during hyperperiod at thermal steady state can be computed using the following expression:

$$\begin{aligned}
\int_0^L \Theta(t)dt &= \int_0^L \mathrm{cool}(\Theta_{\Pi}, t)dt + \sum_{\tau \in \Gamma} \sum_{\{j \in \mathbb{N}^0 | j < L/T_\tau\}} \left(\int_0^{L - \mathrm{st}_{\tau,j}} \eta_\tau(t)dt \right) \\
&= \int_0^{\infty} \mathrm{cool}(\Theta_{\Pi}, t)dt + \sum_{\tau \in \Gamma} \sum_{\{j \in \mathbb{N}^0 | j < L/T_\tau\}} \left(\int_0^{\infty} \eta_\tau(t)dt \right) \\
&\quad - \int_L^{\infty} \mathrm{cool}(\Theta_{\Pi}, t)dt - \sum_{\tau \in \Gamma} \sum_{\{j \in \mathbb{N}^0 | j < L/T_\tau\}} \left(\int_{L - \mathrm{st}_{\tau,j}}^{\infty} \eta_\tau(t)dt \right)
\end{aligned}$$

By superposition, $\mathrm{cool}(\Theta_\Pi, t) + \sum_{\tau\in\Gamma} \sum_{\{j\in\mathbb{N}^0 | j<L/T_\tau\}} \Big(\eta_\tau(L - \mathrm{st}_{\tau,j})\Big)$ represents the temperature at the end of hyperperiod assuming that initial temperature is Θ_Π. By definition of thermal steady state, the temperature at the end of hyperperiod is Θ_Π. Therefore:

$$\int_L^\infty \mathrm{cool}(\Theta_\Pi, t)dt + \sum_{\tau\in\Gamma} \sum_{\{j\in\mathbb{N}^0 | j<L/T_\tau\}} \Big(\int_{L-\mathrm{st}_{\tau,j}}^\infty \eta_\tau(t)dt\Big) = \int_0^\infty \mathrm{cool}(\Theta_\Pi, t)dt$$

Therefore:

$$\int_0^L \Theta(t)dt = \sum_{\tau\in\Gamma} \sum_{\{j\in\mathbb{N}^0 | j<L/T_\tau\}} \Big(\int_0^\infty \eta_\tau(t)dt\Big)$$

By definition 3.4 the total thermal impact of one instance of periodic task τ is: $\frac{P_\tau C_\tau \zeta}{s_\tau}$. Given that periodic task τ has L/T_τ instances in the hyperperiod, the integral of processor temperature at thermal steady state is therefore equal to:

$$\int_0^L \Theta(t)dt = L \cdot \zeta \sum_{\tau\in\Gamma} \frac{P_\tau C_\tau}{T_\tau s_\tau}$$

■

Note that, the value of the integral in the above theorem is only a function of the task characteristics, the speed at which task is executed and UTI of the processor. In particular, it is independent of the schedule Π. We call the value of integral in Equation 3.11 the Hyperperiod Thermal Impact of periodic taskset Γ (HTI(Γ)) for a fixed task-to-speed assignment. The term $\sum_{\tau\in\Gamma} \frac{P_\tau C_\tau}{T_\tau s_\tau}$ represents the average dynamic power consumption of the processor due to scheduling of all periodic tasks in Γ, during one hyperperiod. This average power consumption is represented by P_Γ. Therefore: $\mathrm{HTI}(\Gamma) = L \cdot \delta \cdot P_\Gamma$

Theorem 3.6. *Suppose that all tasks are executed as per* $\Pi(t)$ *and* $\Theta(0) = \Theta_\Pi$. *Let* $\phi(\Pi)$ *be the corresponding maximum temperature during the periodic schedule, i.e.,* $\phi(\Pi) = \max_{0\le t\le L}(\Theta(t))$.

Then,

$$\phi(\Pi) \geq \zeta \sum_{\tau \in \Gamma} \frac{P_\tau C_\tau}{T_\tau s_\tau} = \zeta \cdot P_\Gamma. \tag{3.12}$$

Proof: From Theorem 3.5, we know that at thermal steady state, integral of temperature during the hyperperiod is $L \cdot \zeta \sum_{\tau \in \Gamma} \frac{P_\tau C_\tau}{T_\tau s_\tau}$. The average temperature during the hyperperiod is therefore given by: $\zeta \sum_{\tau \in \Gamma} \frac{P_\tau C_\tau}{T_\tau s_\tau}$. The maximum temperature within the hyperperiod has to be at least as high as the average temperature across the hyperperiod. Hence the theorem. ■

This theorem specifies a lower bound on the maximum temperature of periodic taskset Γ at thermal steady state. We now define *Thermal Utilization* of a taskset Γ as follows.

$$\Upsilon(\Gamma) = \frac{\zeta}{\Delta} \sum_{\tau \in \Gamma} \frac{P_\tau C_\tau}{T_\tau s_\tau} = \frac{\zeta}{\Delta} P_\Gamma. \tag{3.13}$$

Based on this definition of thermal utilization and Theorem 3.6, it follows that $\Upsilon(\Gamma)$ must be less than or equal to 1 in order for a periodic taskset Γ to have a thermally feasible schedule. Otherwise, a lower bound on the maximum core temperature for any schedule of this taskset will be greater than Δ, which in turn violates the thermal constraint of the processor. In other words, if the thermal utilization of a periodic taskset is greater than 1, there is no feasible schedule which meets the thermal constraint of the processor. Therefore, $\Upsilon(\Gamma) \leq 1$ is a necessary condition for thermal feasibility.

3.5 Power Redistribution Algorithm

Based on the lower bound on temperature, we proposed a heuristic for minimizing processor temperature when it is scheduling of implicit deadline periodic tasks [40], called Power Redistribution Algorithm (PRA). We only give overview of PRA here since a superior scheduling scheme

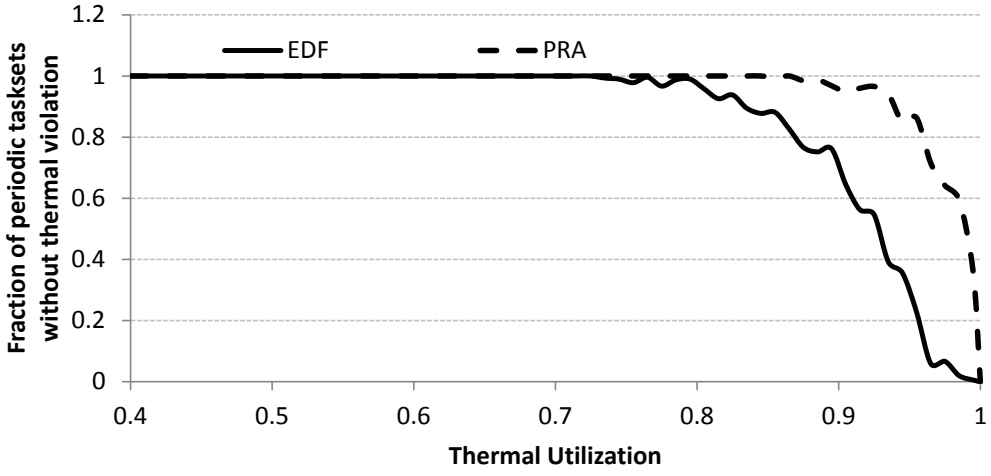


Figure 3.8: Fraction of thermally feasible periodic tasksets under EDF and PRA scheduling policies.

based on GPS/WF2Q was proposed later and is covered in the next chapter. PRA works by interleaving periodic tasks with different power consumption and slack intervals such that the system temperature is minimized. For non real-time applications, the basic interleaving strategy is studied in [34]. The intuition behind the task interleaving scheme is that tasks with low *thermal footprint* (Activity) are executed when the system temperature is high while tasks with high *thermal footprint* are executed when the system temperature is low; where thermal footprint can be considered to be the thermal effect of executing the task.

At a given time t, PRA first computes the time by which the execution of all ready periodic task instances, including partial instances, can be delayed without causing any deadline misses. We propose two different algorithms to compute this *slack* value in this work. If *slack* is available, PRA chooses a task for execution in the time interval $t, t + \epsilon$ which brings temperature at time $t + \epsilon$ closest to a calculated target temperature. The value target temperature is computed using

Theorem 3.6. If *Slack* is not available, then a ready task with earliest deadline is executed to avoid deadline miss.

Figure 3.8 plots the fraction of periodic tasksets that were feasible using EDF and PRA scheduling policies, as a function of thermal utilization. As specified in Theorem 3.6 and Equation 3.13, a thermal utilization of ≤ 1 is a necessary condition for thermal feasibility. Therefore, after thermal utilization of 1, the fraction of thermally feasible periodic tasks for both PRA and EDF becomes 0 in Figure 3.8. However, PRA is able to feasibly schedule several more high thermal utilization periodic tasksets compared to EDF. This illustrates the efficacy of PRA in in terms of minimizing maximum temperature.

In the next chapter, we present necessary and sufficient conditions for schedulability of implicit deadline periodic tasks. We also present thermally optimal scheduling strategies inspired by Generalized Processor Sharing.

Chapter 4

Thermally Optimal Solution for Implicit Deadline Periodic Tasks on Uni-Core

In this chapter we prove necessary and sufficient conditions for thermal schedulability of implicit deadline periodic tasks on uni-core and prove that a scheduling algorithm based on Generalized Processor Sharing [26] is thermally optimal. This implies that:

For a given periodic taskset executing on a uni-core, If GPS based algorithm can not meet the thermal constraints of the system; then no other scheduling algorithm will be able to meet thermal constraints of the system

We also present a discrete approximation of GPS and empirically validate the theoretical results.

4.1 Generalized Processor Sharing as a Thermally Optimal Scheduling Discipline

We now prove that a scheduling scheme inspired by Generalized Processor Sharing (GPS) [26] achieves this lower bound on temperature. Generalized Processing Sharing is a service policy for entities sharing a common resource. In the context of this chapter, the different entities sharing a common resource are the individual periodic tasks in periodic taskset Γ. The shared resource in

The work covered in this Chapter appeared in [42]
Rehan Ahmed, Parameswaran Ramanathan, Kewal K. Saluja, "Necessary and Sufficient Conditions for Thermal Schedulability of Periodic Real-Time Tasks", ECRTS 2014

this case is the core itself with a resource capacity of 1. GPS assumes that service is fluid. In the context of this work, this implies that all periodic tasks make progress concurrently at rate equal to $\frac{C_\tau}{T_\tau s_\tau} \quad \forall \tau \in \Gamma$. This is an idealized discipline and will be approximated by Worst Case Fair Weighted Fair Queuing. The explanation of Worst-Case-Fair Weighted Fair Queuing is covered in the next subsection. Note that the version of GPS considered in this work is different from traditional GPS. In traditional GPS any unused system capacity (if $\sum_{\tau \in \Gamma} \frac{C_\tau}{T_\tau s_\tau} < 1$), is distributed among the active task streams. In our version, the unused system capacity is not distributed and each task is executed at constant rate equal to $\frac{C_\tau}{T_\tau s_\tau}$

Theorem 4.1. *The lower bound on maximum temperature for periodic taskset* Γ *at thermal steady state, identified by Theorem 3.6, is achieved if and only if the processor temperature is constant throughout hyperperiod and is equal to* $\frac{\mathrm{HTI}(\Gamma)}{L}$

Proof: We prove this by contradiction.

Case1 $\Theta(t_1) < \frac{\mathrm{HTI}(\Gamma)}{L}$**:** If the processor temperature at time t_1, $(\Theta(t_1)) < \frac{\mathrm{HTI}(\Gamma)}{L}$, the processor temperature at some other time t_2 will have to be greater than $\frac{\mathrm{HTI}(\Gamma)}{L}$ for the integral of processor temperature during hyperperiod to equal $\mathrm{HTI}(\Gamma)$. In this case, we do not achieve the lower bound on temperature since $\Theta(t_2) > \frac{\mathrm{HTI}(\Gamma)}{L}$.

Case2: $\Theta(t_1) > \frac{\mathrm{HTI}(\Gamma)}{L}$**:** If the processor temperature at time t_1, $\Theta(t_1) > \frac{\mathrm{HTI}(\Gamma)}{L}$, then we do not achieve the lower bound on temperature because of temperature at t_1.

Therefore, processor temperature has to be constant at $\frac{\mathrm{HTI}(\Gamma)}{L}$ for us to achieve lower bound on maximum temperature for periodic taskset Γ at thermal steady state. ■

Corollary 4.2. *The lower bound on maximum temperature for periodic taskset* Γ*, is achieved if and only if the processor power consumption is constant throughout hyperperiod and is equal to* P_Γ

Proof: From Theorem 4.1, the lower bound on maximum temperature is achieved for periodic taskset Γ if temperature is constant at $\frac{\text{HTI}(\Gamma)}{L} = \zeta \cdot P_\Gamma$. This constant temperature can only be achieved with constant power consumption; since any variation in power consumption would cause variation in temperature. The energy consumption during one hyperperiod for periodic taskset Γ can be computed as $L \cdot P_\Gamma$. Therefore, to achieve lower bound on maximum temperature, power consumption must be constant at P_Γ. ■

Lemma 4.3. *If computational utilization of a periodic taskset* $\Gamma \leq 1$ *and GPS algorithm with weight for task* $\tau \in \Gamma$ *equal to* $\frac{C_\tau}{T_\tau s_\tau}$ *is used for scheduling, all periodic task instances of task* τ *will finish exactly at their absolute deadlines.*

Proof: For a periodic task τ, GPS guarantees a rate of execution equal to its weight ($\frac{C_\tau}{T_\tau s_\tau}$) if the sum of all weights in the system is less than system capacity ($\sum_{\tau \in \Gamma} \frac{C_\tau}{T_\tau s_\tau} \leq 1$). Since the computation utilization of taskset $\Gamma \leq 1$, GPS will guarantee execution rate of exactly $\frac{C_\tau}{T_\tau s_\tau}$ $\forall \tau \in \Gamma$. Now suppose that we have a periodic task instance j of task τ with arrival time t_1 and deadline $t_1 + T_\tau$. By definition, GPS will execute j for exactly $\frac{C_\tau}{T_\tau s_\tau}(t_2 - t_1)$ where $t_1 < t_2 \leq t_1 + T_\tau$. For a periodic task instance to complete, it needs to execute for C_τ / s_τ units of time. At time $t + T_\tau$, GPS will schedule instance j for exactly $\frac{C_\tau}{T_\tau s_\tau}((T_\tau + t_1) - t_1) = C_\tau / s_\tau$ time units. Therefore, instance j will complete exactly at its deadline. Also, since the next instance of periodic task τ arrives at time $t_1 + T_\tau$, periodic task τ is always *Backlogged*. ■

Theorem 4.4. *If computational and thermal utilization of a periodic taskset* Γ *are* ≤ 1*, then Generalized Processor Sharing (GPS) algorithm with weight for task* τ *equal to* $\frac{C_\tau}{T_\tau s_\tau}$ *finds a schedule for* Γ *that meets all deadline constraints and keeps the core temperature at or below threshold* Δ*.*

Proof: By Lemma 4.3, there will be no timing violations if periodic taskset Γ is scheduled using GPS and computation utilization of $\Gamma \leq 1$. Now, consider any time interval $(t, t+h) \in [0, L)$. By Lemma 4.3, we know all all periodic tasks are always backlogged. Therefore, by definition GPS will ensure that each task $\tau \in \Gamma$ executes for $h \cdot \frac{C_\tau}{T_\tau s_\tau}$ time units in this interval. The average power of all task instances executed in this interval is $\sum_{\tau \in \Gamma} \frac{C_\tau P_\tau}{T_\tau s_\tau} = P_\Gamma$. Now if we make the length of this interval infinitesimally small (i.e. $h \to 0$), we can consider the power consumption constant at P_Γ throughout the hyperperiod. By Corollary 4.2, a constant power of P_Γ achieves the lower bound on maximum temperature for periodic taskset Γ. This constant temperature is equal $\frac{\mathrm{HTI}(\Gamma)}{L} = \zeta \cdot P_\Gamma$ (Theorem 4.1). Therefore, if Thermal Utilization of periodic taskset ($\frac{\zeta}{\Delta} P_\Gamma \leq 1$), maximum temperature given by GPS schedule $\leq \Delta$; hence the theorem. ■

4.2 Worst-Case-Fair Weighted Fair Queuing

WF2Q [43] is a scheduling scheme which approximates GPS when the system is not fluid (processor is not infinitely preemptible). We use this scheme in the analysis of results presented in Sections 4.4 and 5.5. In the context of this work, WF2Q will assign each task τ a weight of $\frac{C_\tau}{T_\tau s_\tau}$. WF2Q makes scheduling decisions for contiguous time periods called *Execution Intervals*. Let $E_{\tau,\mathrm{WF2Q}}(t)$ be the net execution completed for task τ at time t under WF2Q scheduling policy. Likewise, let $E_{\tau,\mathrm{GPS}}(t) = \frac{C_\tau}{T_\tau s_\tau} \cdot t$ be the net execution completed for task τ at time t under

GPS scheduling policy. Also let $B(t)$ be the set of tasks at time t such that $B(t) = (\tau \in \Gamma : E_{\tau,\mathrm{WF2Q}}(t) \leq E_{\tau,\mathrm{GPS}}(t))$ i.e. GPS has completed at least as much execution as Wf2Q at time t for all tasks in $B(t)$. Then at time t WF2Q will pick a task $i \in B(t)$ to execute during time $[t, t+$*execution interval*$)$ such that the scheduled execution has the earliest finish time under GPS scheduling policy. i.e. $i = \underset{j \in B(t)}{\operatorname{argmin}}\big((E_{\mathrm{j,WF2Q}}(t) + \textit{execution interval}) \cdot \frac{T_j}{C_j}\big)$

On a uni-core system, WF2Q yields a schedule which is P-fair [44]; i.e. the execution done for each periodic task will have an absolute difference of less than 1 *execution interval* from its execution in a GPS schedule [43] at all times. Therefore, WF2Q scheduling scheme will not cause any deadline misses if computation utilization is $\leq$1. Other scheduling algorithms which yield a P-fair schedule [34] may also be used with similar results. Note that WF2Q scheduling has a high preemption overhead since a context switch occurs after every execution interval. Furthermore, if periodic tasks are assigned different speeds, the processor speed also changes after each execution interval. Therefore, the length of execution interval has to be chosen large enough to mitigate the effect of these overheads on a given system.

The discretization of fluid scheduling scheme leads to a temperature profile that oscillates around the temperature profile of the fluid scheduling scheme executing the same tasks. At different time instances the temperature may be higher or lower than the temperature of the fluid scheme. This variation is a decreasing function of scheduling interval length and power variation across periodic/aperiodic tasks. Temperature variance because of discretization is illustrated in the example covered in next section and visually shown in Figure 4.2.

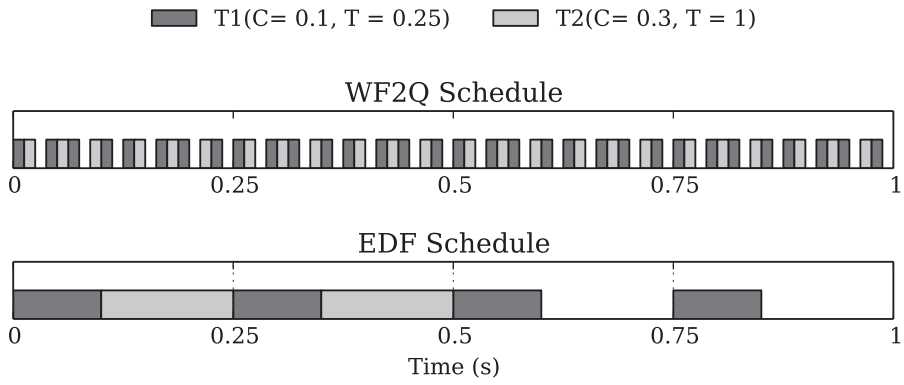


Figure 4.1: EDF and WF2Q schedules for the example periodic taskset.

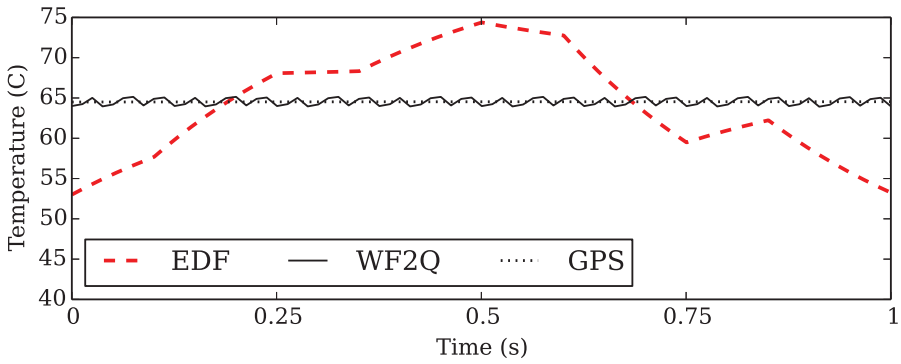


Figure 4.2: Steady state thermal profile for schedules shown in Figure 4.1 and GPS.

4.3 Scheduling Example

To illustrate the proposed concepts, we use the periodic taskset introduced in Section 3.2.1 as an example. As before, the periodic taskset considered constitutes two periodic tasks:

Task 1 ($C_1 = 0.1, T_1 = 0.25, a_1 = 80$)

Task 2 ($C_2 = 0.3, T_2 = 1, a_2 = 120$)

The speeds at which both periodic tasks are executed is assumed to be 1. The hyperperiod for this

periodic taskset is LCM(0.25,1) = 1sec. Figure 4.1 shows the EDF schedule for one hyperperiod. Figure 4.1 also shows the WF2Q schedule for one hyperperiod with execution interval length = 12.5 msec. Figure 4.2 shows the temperature profile of the core during hyperperiod at thermal steady state (initial temperature = Θ_Π) for both EDF and WF2Q schedules. As shown in the figure, the maximum temperature for WF2Q is lower than the maximum temperature for EDF. However, the area under the curves for both EDF and WF2Q are the same. This area can be computed using Equation 3.11 and 3.7 as:

$$\begin{aligned}\int_o^L \Psi(t) &= L.\left(\sum_{\tau\in\Gamma}\frac{C_\tau P_\tau \zeta}{T_\tau} + \Psi_{\text{idle,ss}}\right)\\ &= (4\times 0.1\times 80 + 1\times 0.3\times 120)\zeta + 40.05\end{aligned}$$

using the thermal parameters, the value of $\zeta = \frac{1}{\beta C} = 0.360$. Therefore,

$$\int_o^L \Psi(t) = 68\times 0.36 + 40.05 = 64.53$$

Figure 4.2 also shows the temperature profile of a fluid (GPS) schedule. With a GPS schedule, the temperature remains constant at $\sum_{\tau\in\Gamma}\frac{P_\tau C_\tau}{T_\tau} + \Psi_{\text{idle,ss}} = 64.53°$C. This is the lower bound on maximum temperature which can be achieved for the periodic taskset under consideration.

4.4 Validation Results

The simulations conducted in this section use the simplified thermal model defined in Section 3.1.3. We use UUniFast algorithm [45] to generate more than 10,000 periodic tasksets with computation utilizations varying between 0.6-1.0. The power consumption of the periodic tasks in a periodic taskset is uniformly distributed between 30W and 250W. Tasksets are selected such that

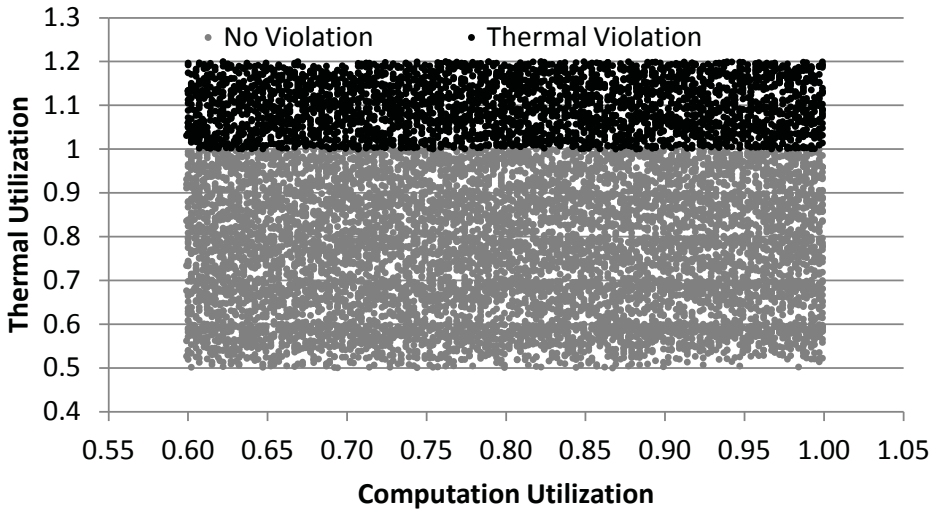


Figure 4.3: Computation and thermal utilization of periodic tasksets simulated; and their thermal feasibility with GPS scheduling policy.

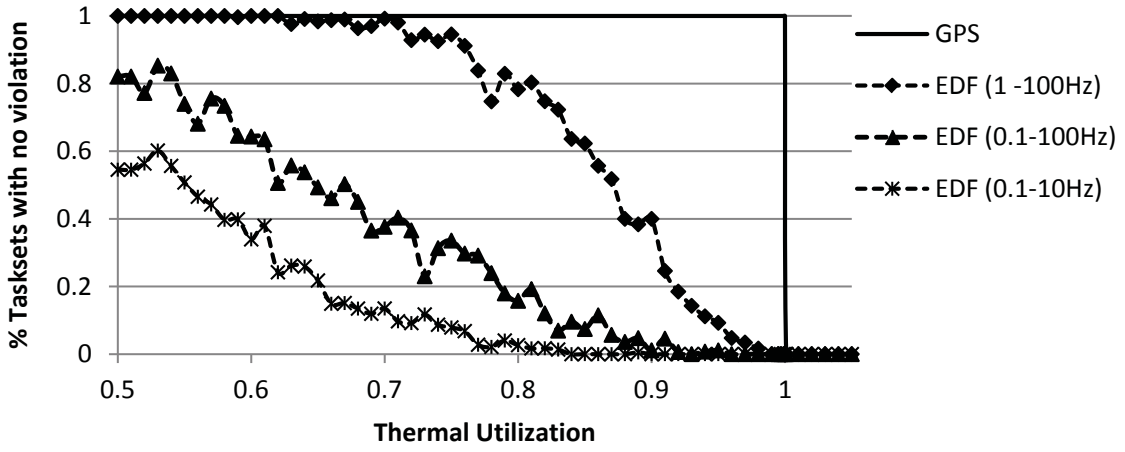


Figure 4.4: Fraction of periodic tasksets which are thermally feasible using GPS, and EDF. Different periodic task frequencies simulated.

their thermal utilization varies between 0.6-1.2. It is assumed that the speed of all periodic tasks is fixed at 1.

Figure 4.3 shows the computation and thermal utilizations of all periodic tasksets. It shows the periodic tasksets which did not have any thermal violations using GPS, as gray dots. The periodic tasksets which had thermal violations are shown as black dots. As Theorem 4.4 states, a thermal utilization of ≤ 1 is both necessary and sufficient condition for thermal feasibility of a periodic taskset. Therefore, all periodic tasksets with thermal utilization ≤ 1 have no thermal violations. Figure 4.3 also shows that all periodic tasksets with thermal utilization > 1 violated thermal constraint.

Figure 4.4 shows the fraction of periodic tasksets which were accepted for each thermal utilization. The figure shows the idealized GPS results (All periodic tasksets with thermal utilization ≤ 1 do not cause thermal violations to occur). Figure 4.4 also shows the acceptance curves for Earliest Deadline First scheduling algorithm. We conducted EDF simulations with three different sets of periodic tasksets. In the first set of results, the computation times and periods of all periodic tasksets are time scaled such that periodic tasks have frequencies varying between [1Hz, 100Hz]. For the second set of results, the time scaling factors are kept such that all periodic tasks have frequencies varying between [0.1Hz, 100Hz]. For the third set of results, periodic tasks have frequencies varying between [0.1Hz, 10Hz]. As Figure 4.4 shows, periodic tasksets with high frequency periodic tasks have a higher thermal acceptance ratio as compared to low frequency periodic tasksets when EDF is used as a scheduling algorithm. This is because, when periodic tasks have low frequency, it is likely that EDF will schedule a high power task for a large time duration;

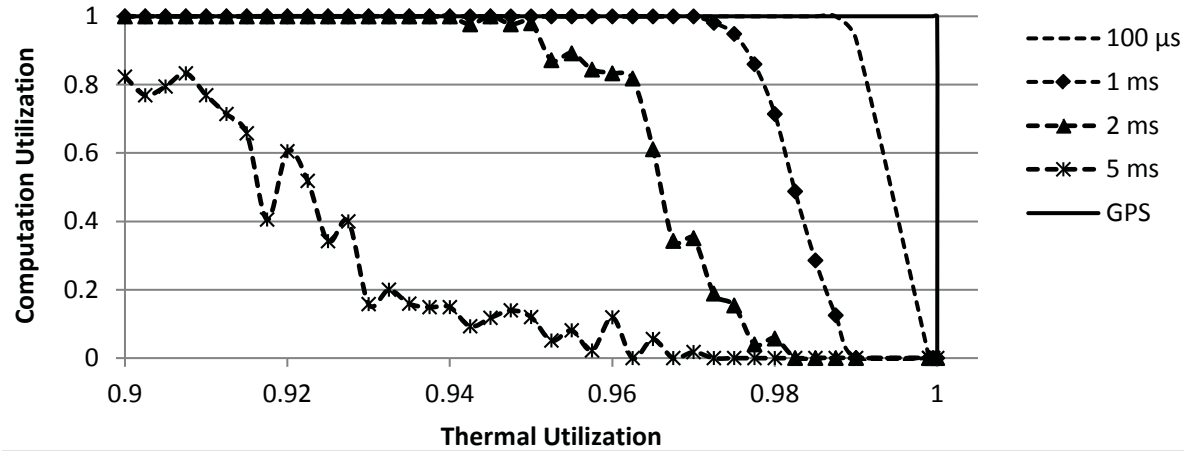


Figure 4.5: Fraction of thermally feasible periodic tasksets under WF2Q scheduling policy with various Execution Interval lengths.

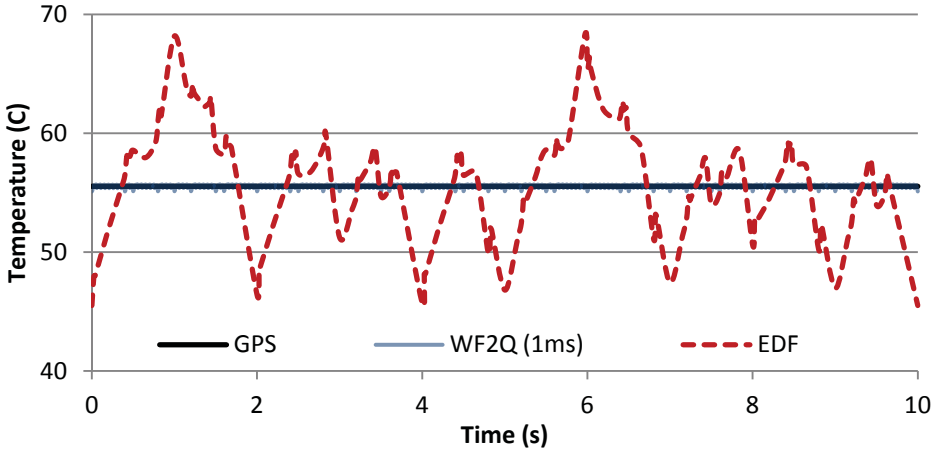


Figure 4.6: Processor temperature as a result of schedule given by GPS, WF2Q, and EDF scheduling schemes for a periodic taskset.

causing the processor temperature to exceed threshold temperature. However, when periodic tasks have high frequency, regular preemptions will occur; making the execution of high power tasks for large duration of time less likely. Therefore, high frequency periodic tasksets have a higher acceptance ratio as compared to low frequency periodic tasksets. On the contrary, the acceptance ratio of GPS scheduling strategy does not depend on the frequency of periodic tasks.

Figure 4.5 compares the acceptance ratios of periodic tasksets with thermal utilizations > 0.9 and with various lengths of WF2Q execution intervals. We conducted simulations for execution intervals of length $100\mu s$, $1ms$, $2ms$ and $5ms$. It is evident from the figure that the acceptance ratio is higher for WF2Q with shorter execution intervals. This is because WF2Q with a short execution interval emulates GPS more closely compared to WF2Q with a long scheduling interval. Figure 4.6 shows the temporal variation in temperature when GPS, WF2Q and EDF schedules for a periodic taskset are executed. As shown in the figure, EDF schedule has the highest variation in temperature. It also has the highest maximum temperature among all scheduling schemes (68.32°C). The plots for GPS and WF2Q schedules are almost overlapping; with temperature remaining almost constant at around 55.5° C. The execution interval for WF2Q schedule result shown in Figure 4.6 is $1ms$.

In the next chapter, we present an optimal scheduling algorithm for aperiodic tasks executing on thermally constrained uni-core processing environments. The proposed strategy is a thermal extension of the Total Bandwidth Server [27] and utilizes the GPS inspired optimal scheduling solution presented in this chapter.

Chapter 5

Thermal Extension of the Total Bandwidth Server

This chapter covers a scheme for scheduling aperiodic tasks with hard deadline periodic tasks in thermally constrained uni-core systems. We thermally extend Total Bandwidth Server (TBS) [27] scheduling scheme. The proposed scheme is optimal in terms of being able to support maximum rate of aperiodic task arrivals.

5.1 Task Model: Aperiodic Tasks

Aperiodic tasks do not conform to any specific arrival pattern and are not deadline constrained. An aperiodic task j is characterized by (i) release time R_j, (ii) worst-case computation time C_j, worst-case dynamic power consumption P_j. It is assumed that these parameters are known when an aperiodic task arrives on the system. The application considered in this chapter is composed of both periodic tasks (model covered in Section 3.1.4) and aperiodic tasks. It is assumed at all aperiodic tasks are executed at speed 1.

The work covered in this chapter appears in [46]
Rehan Ahmed, Ayoosh Bansal, Bhuvana Kakulooni, Parameswaran Ramanathan, Kewal K.Saluja, "Thermal Extension of the Total Bandwidth Server", VLSI Design 2015.

5.2 Total Bandwidth Server

Total Bandwidth Server (TBS) is a scheme for scheduling aperiodic tasks with hard deadline periodic tasks. The functioning of TBS is simple. When j^{th} aperiodic task arrives at time R_j with computation requirement C_j, it is assigned a deadline D_j such that:

$$D_j = max(R_j, D_{j-1}) + C_j/U_A$$

where D_{j-1} is the deadline assigned to the previous aperiodic task and U_A is the computation utilization/system bandwidth assigned for the scheduling of aperiodic tasks. Once deadline (D_j) is assigned, TBS uses Earliest Deadline First (EDF) algorithm to schedule periodic and aperiodic tasks.

TBS scheme has the ability to strictly bound the amount of execution aperiodic tasks receive within any time interval. Consider a time interval $[t_1, t_2]$ and let C_{ape} be the total amount of aperiodic task execution for aperiodic tasks which arrive and have deadlines with in this interval. In [27], Spuri et al. prove that $C_{\text{ape}} \leq (t_2 - t_1)U_A$. Furthermore, they prove that there will be no timing violations if $U_A + U_P \leq 1$, where U_P is the total utilization of all periodic tasks. To guarantee maximum service for aperiodic tasks and no deadline violations for periodic tasks, TBS sets $U_A + U_P = 1$. However, TBS is not a thermal-aware scheduling scheme. Therefore, using TBS in a thermally-constrained system may lead to thermal violations. We will illustrate this in section 5.4. In the following section, we propose a new algorithm inspired by TBS called Thermally Constrained Total Bandwidth Server (T2BS).

5.3 Thermally Constrained Total Bandwidth Server

Thermally Constrained Total Bandwidth Server (T2BS) schedules all periodic/aperiodic tasks at set rates assuming a fluid system. T2BS assigns incoming aperiodic tasks deadlines based on both their computation requirement and power consumption such that: i) There are no timing violations for periodic tasks ii) There are no thermal violations during scheduling of aperiodic tasks iii) There are no thermal violations in the future, due to execution of periodic tasks.

In T2BS, aperiodic tasks are assigned computation utilization limit (U_A^C) and thermal utilization limit (U_A^T). Furthermore, we also define Computation Utilization of periodic tasks as $U_P^C = \sum_{\tau \in \Gamma} \frac{C_\tau}{T_\tau s_\tau}$ and Thermal Utilization of periodic tasksets as $U_P^T = \Upsilon(\Gamma) = \frac{\zeta}{\Delta} \sum_{\tau \in \Gamma} \frac{P_\tau C_\tau}{T_\tau s_\tau}$.

When j^{th} aperiodic task arrives, at time R_j, it is assigned a deadline equal to:

$$D_j = \max(D_j^C, D_j^T) \quad \text{where} \tag{5.1}$$

$$D_j^C = \max(D_{j-1}, R_j) + C_j / U_A^C \tag{5.2}$$

$$D_j^T = \max(D_{j-1}, R_j) + \frac{\zeta C_j P_j}{\Delta U_A^T} \tag{5.3}$$

In both Equations 5.2 and 5.3, D_{j-1} is the deadline of the previous aperiodic task. In T2BS, all periodic tasks are scheduled using GPS with execution rates equal to $\frac{C_\tau}{T_\tau s_\tau} \forall \tau \in \Gamma$. Aperiodic task j is scheduled using GPS in the interval $[\max(D_{j-1}, R_j), D_j]$ with execution rate equal to $\frac{\mathrm{C_j}}{\mathrm{D_j - max(D_{j-1}, R_j)}}$. We use a fluid scheduling scheme to present the theoretical results in this section. In section 5.3.3, we review Worst Case Fair Weighted Fair Queuing as a non-fluid approximation of GPS.

5.3.1 Theoretical Results

Theorem 5.1. *There will be no timing violations for periodic tasks and no thermal violations if*

$$U_P^C + U_A^C \leq 1 \quad \textit{and} \quad U_P^T + U_A^T \leq 1$$

and T2BS scheduling strategy is used.

The following text will prove Theorem 5.1. We also prove the tightness of results presented in Theorem 5.1 at the end of this section.

Lemma 5.2. *All periodic tasks meet their deadline constraints under T2BS if* $U_P^C + U_A^C \leq 1$

Proof: To prove no deadline violations, it is sufficient to prove that the net execution rate requirement within any interval does not exceed 1 (Theorem 4.4). Specifically, periodic tasks will not miss their deadlines if:

$$U_P^C + \frac{C_{\text{ape\_T2BS}}}{t_2 - t_1} \leq 1 \quad \forall\{(t_1, t_2) | t_1 < t_2\}$$

where $C_{\text{ape\_T2BS}}$ is the amount of computation done for aperiodic tasks between time interval $[t_1, t_2]$ by algorithm T2BS. From Equation 5.1 and 5.2, we know that execution rate of aperiodic tasks $\leq U_A^C$ i.e. $\frac{C_{\text{ape\_T2BS}}}{t_2 - t_1} \leq U_A^C$ Therefore, $U_P^C + U_A^C \leq 1$ guarantees that there will be no deadline violations for periodic tasks. ■

Lemma 5.3. *Processor power consumption is* $\leq \Delta/\zeta$ *at all times if* $U_P^T + U_A^T \leq 1$

Proof: At any time t, we can divide the processor dynamic power consumption into two components.

$$P_{dyn}(t) = \frac{U_P^T \Delta}{\zeta} + P_{dyn\_A}(t)$$

The first term is power consumption due to the execution of periodic task streams. This is calculated based on the definition of thermal utilization in Equation 3.13. $P_{dyn\_A}(t)$ is the power consumption due to execution of aperiodic tasks. $P_{dyn\_A}(t)$ can vary depending on the following three cases:

i) No active aperiodic task: $P_{dyn\_A}(t) = 0$

ii) $D_j^C \geq D_j^T$**:** From Equation 5.2, rate of execution of aperiodic task is equal to U_A^C and resultant power consumption is $P_{dyn\_A}(t) = U_A^C P_j$. By Equations 5.2 and 5.3

$$\begin{aligned} \frac{C_j}{U_A^C} &\geq \frac{\zeta C_j P_j}{\Delta U_A^T} \quad \text{Therefore,} \\ P_{dyn\_A} &\leq \frac{U_A^T \Delta}{\zeta} \end{aligned}$$

iii) $D_j^C < D_j^T$**:** From Equation 5.3, the rate of execution of aperiodic task is equal to $\frac{U_A^T \Delta}{\zeta P_j}$. Therefore, $P_{dyn\_A} = \frac{U_A^T \Delta}{\zeta}$

Considering all three cases, $P_{dyn\_A} \leq \frac{U_A^T \Delta}{\zeta}$. Therefore, $P_{dyn}(t) \leq \frac{U_P^T \Delta}{\zeta} + \frac{U_A^T \Delta}{\zeta}$. Therefore, if $U_P^T + U_A^T \leq 1$, $P_{dyn}(t) \leq \Delta/\zeta \quad \forall t$. ■

Furthermore, by definition of Thermal Utilization in Equation 3.13 and Theorem 4.4, if processor power consumption is upper bounded by Δ/ζ, processor temperature will be $\leq \Delta$. Therefore, Lemma 5.2 and 5.3 lead to Theorem 5.1.

Theorem 5.4. *No scheduling algorithm can guarantee timing feasibility if* $U_P^C + U_A^C > 1$

Proof: As a worst case scenario, assume that aperiodic tasks arrive periodically with computation utilization equal to U_A^C. This arrival pattern conforms to the maximum computation bandwidth

assigned to aperiodic tasks. In this scenario, if $U_P^C + U_A^C > 1$, no scheduling algorithm can guarantee timing constraints for periodic tasks; since the required computation bandwidth is greater than available computation bandwidth. Hence the theorem. ■

Theorem 5.5. *No scheduling algorithm can guarantee thermal feasibility if* $U_P^T + U_A^T > 1$

Proof: As a worst case scenario, assume that aperiodic tasks arrive periodically with thermal utilization equal to U_A^T. This arrival pattern conforms to the maximum thermal bandwidth assigned to aperiodic tasks. In this scenario, if $U_P^T + U_A^T > 1$, then, by Theorem 3.6 and Equation 3.13, no scheduling algorithm can guarantee thermal feasibility. Hence the theorem. ■

5.3.2 Statistical Estimation of Response Time:

T2BS uses GPS to schedule all periodic/aperiodic tasks. Therefore, all periodic/aperiodic tasks finish at their deadline. If the arrival of aperiodic tasks forms a Poisson Process, then we can use this property to determine the average response time of aperiodic task using Pollaczek-Khinchin (P-K) mean value formula [47]. By Equation 5.1, the service time (finish time - start time) of aperiodic tasks is $\max(\frac{C_j}{U_A^C}, \frac{\zeta C_j P_j}{\Delta U_A^T})$. Using queuing theory, the mean response time is equal to:

$$W = E(S) + \frac{\lambda(E(S)^2 + \text{Var}(S))}{2(1 - \lambda E(S))} \tag{5.4}$$

Where $E(S)$ is the mean service time, Var(S) is the variance of service time and λ is the mean arrival rate of aperiodic tasks assuming a Poisson Process. We also define the Utilization of aperiodic tasks:

$$U_A = \lambda E(S) \tag{5.5}$$

For a stable system, U_A has to be < 1. If $U_A \geq 1$, the mean response time of aperiodic tasks is unbounded.

5.3.3 Discretized T2BS

Discretized T2BS (D-T2BS) is a variant of T2BS where Worst Case Fair Weighted Fair Queuing (WF2Q) [43] as explained in Section 4.2 is used instead of GPS to schedule periodic and aperiodic tasks. To review, WF2Q is a scheduling scheme which approximates GPS when the system is not fluid (processor is not infinitely preemptible). In the context of this chapter, D-T2BS will assign each periodic task execution rate equal to $\frac{C_\tau}{T_\tau s_\tau} \forall \tau \in \Gamma$. Aperiodic task j is scheduled in an interval $[\max(D_{j-1}, R_j), D_j]$ with execution rate equal to $\frac{\mathrm{C_j}}{\mathrm{D_j - \max(D_{j-1}, R_j)}}$. Scheduling decisions are made for contiguous time periods called *Execution Intervals*. Because WF2Q closely approximates GPS, the absolute difference in response time of D-T2BS and T2BS is upper bounded by:

$$|W_{j,T2BS} - W_{j,D-T2BS}| \leq \frac{\text{Execution Interval}}{\min(U_A^C, \Delta U_A^T/(\zeta P_j))} \tag{5.6}$$

As explained in Section 4.2, The discretization of fluid scheduling scheme leads to a temperature profile that oscillates around the temperature profile of the fluid scheduling scheme executing the same tasks. Temperature variance because of discretization is illustrated in the example covered in next section and visually shown in Figure 5.2.

5.4 Scheduling Example

This section covers an example which illustrates the working of T2BS algorithm. The application consists of two periodic (P1, P2) and two aperiodic (A1,A2) tasks. Task characteristics are given in Table 5.1 The speeds of all tasks, periodic and aperiodic, are assumed to be 1. Based

Task	Computation	Period	Power	Release
P1	100 ms	250 ms	80 W	-
P2	300 ms	1000 ms	120 W	-
A1	150 ms	-	60 W	0 ms
A2	100 ms	-	120 W	100 ms

Table 5.1: Periodic/Aperiodic task parameters.

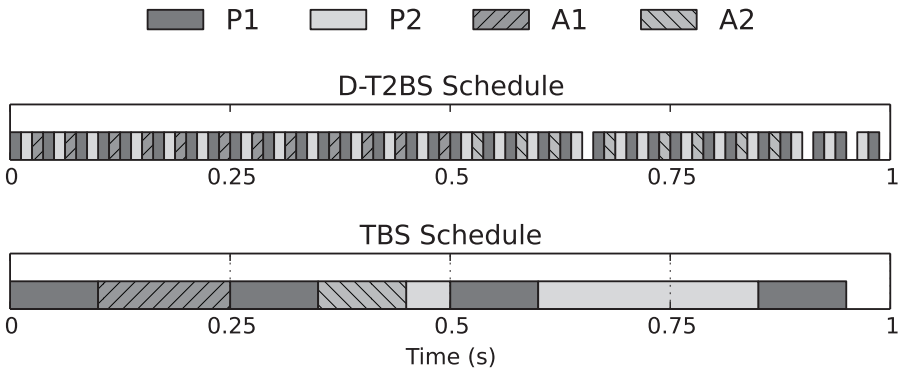


Figure 5.1: Example schedule with two periodic and two aperiodic tasks for TBS and D-T2BS scheduling schemes.

on the these task parameters, $U_P^C = 0.7$, $U_A^C = 1 - U_P^C = 0.3$. Furthermore, $U_P^T = 0.7$, $U_A^T = 1 - U_P^T = 0.3$ with the assumption that $\zeta = 0.36, \Delta = 34.95°C$ and $\widehat{\Delta} = 75°C$. We will now go over how TBS schedules all tasks. When Task A1 arrives at time 0, it is assigned a deadline equal to $C_{A1}/U_A^C = 500ms$. It is then scheduled using EDF along with the periodic tasks. A2 arrives at time 100ms and is assigned a deadline equal to $max(D_{A1}, R_{A2}) + C_{A1}/U_A^C = 500+330 = 830ms$. It is also scheduled using EDF. The schedule using TBS is given in Figure 5.2.

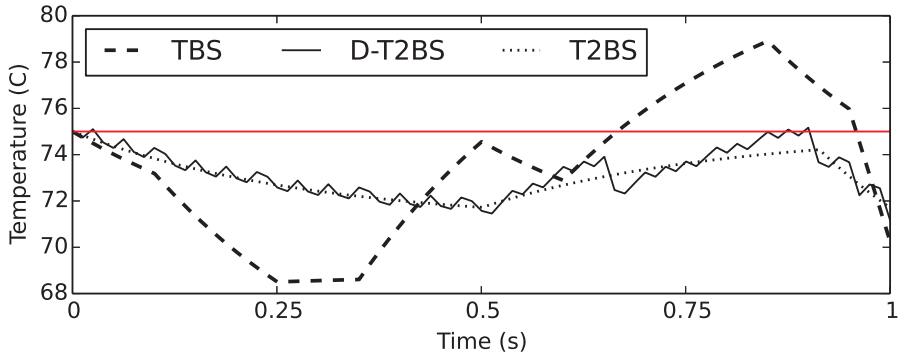


Figure 5.2: Hyperperiod temperature profile for TBS and D-T2BS and T2BS scheduling schemes.

In contrast to TBS, T2BS assigns aperiodic task deadlines differently. Based on Equations 5.2 and 5.3: $D^C_{A1} = 500ms,\ D^T_{A1} = 309ms,\ D^C_{A2} = 833ms,\ D^T_{A2} = 912ms$. Therefore, $D_{A1} = 500ms$ and $D_{A2} = 912ms$. Notice that A2 has a later deadline in T2BS compared to its deadline in TBS. This is because A2 has a high power consumption and T2BS decreases its execution rate to keep processor temperature below threshold during its execution. T2BS executes periodic tasks P1 and P2 at constant rates equal to 0.4 and 0.3 throughout the hyperperiod. Aperiodic task A1 is executed at rate 0.3 in interval $[0, 500ms]$ and task A2 is executed at rate 0.243 in interval $[500ms, 912ms]$. These execution rates lead to a power consumption of 86 W in interval $[0, 500ms]$, 97.047 W(Δ/ζ) in interval $[500ms, 912ms]$ and 68 W in interval $[912ms, 1000ms]$. Figure 5.2 shows the unadjusted temperature for TBS and T2BS schedules assuming that initial unadjusted temperature is $75°C$. For TBS schedule, the maximum unadjusted processor temperature ($78.9°C$) is higher than $\widehat{\Delta} = 75°C$ causing thermal violation. In contrast, the temperature for T2BS always remains below threshold.

Figures 5.1 and 5.2 also show the D-T2BS schedule with execution interval length of length $12.5ms$ and its corresponding temperature. As shown in Figure 5.2, maximum unadjusted temperature is lower for D-T2BS scheme ($75.16°C$) compared to TBS. However, it is slightly higher than threshold. The increase in temperature for D-T2BS scheme is because of discretization of fluid scheduling scheme in T2BS. To counteract for this temperature increase, we could either decrease the execution interval length or decrease U_A^T such that $U_P^T + U_A^T < 1$.

5.5 Results

In this section, we analyse the performance of T2BS and D-T2BS algorithms through a series of experiments/simulations. We show that the response time of the T2BS algorithm closely follows the theoretical value computed using Equation 5.4. Furthermore, we demonstrate through extensive simulations that T2BS never exceeds threshold temperature. The performance of D-T2BS is also compared against T2BS.

Simulation Setup: We use the same thermal model as used in the example section (Section 5.4). In our experiments $U_P^C = 0.6$ and $U_P^T = 0.65$. Therefore, $U_A^C = 0.4$ and $U_A^T = 0.35$. Aperiodic arrivals follow a Poisson process. In this section J will be used to to represent the set of all aperiodic tasks. For succinctness, we define power consumption $P_S = \frac{U_A^T \Delta}{\zeta U_A^C} = 84.92W$. Given an aperiodic task j; if $P_j > P_S$, then $D_j = D_j^T$; otherwise $D_j = D_j^C$. P_S will be used for explanation of results in this section. Without loss of generality, the computation time of all aperiodic tasks is assumed to be 100ms. For all simulations conducted, simulation points where aperiodic task

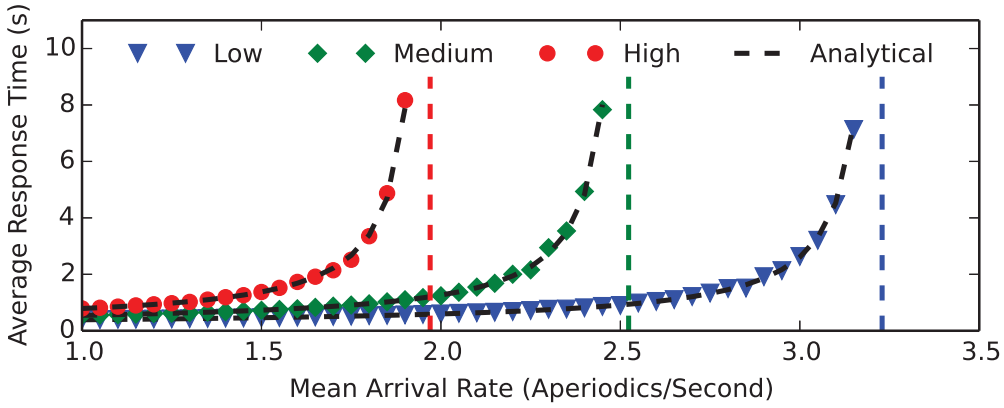


Figure 5.3: Response time for mixed aperiodic tasks with varying levels of average power consumption.

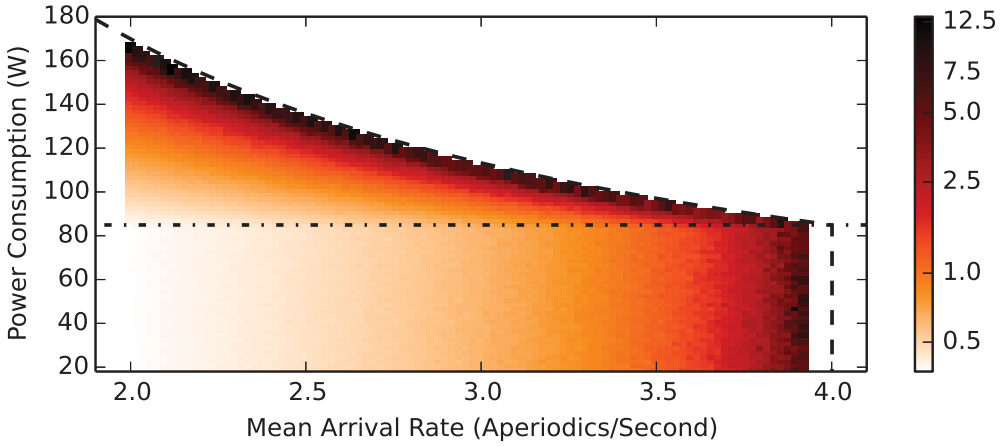


Figure 5.4: Mean response time, as a function of arrival rate and aperiodic task power consumption.

utilization $U_A = \lambda E(S) < 0.98$ are simulated. As explained in section 5.3, $U_A \geq 1$ leads to an unstable system where aperiodic task response times grow unbounded.

Simulation Results: In the first experiment, we analyse the effect of aperiodic task power consumption and arrival rate on response time. In this experiment, aperiodic tasks are chosen from a set of aperiodic task types; with each type having different power consumption. By varying the

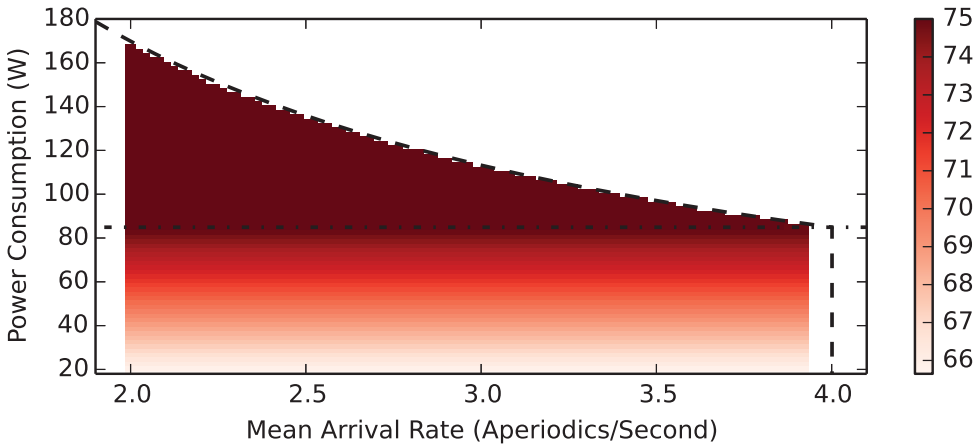


Figure 5.5: Maximum temperature, as a function of arrival rate and aperiodic task power consumption.

probabilities of selecting aperiodic task types, different average power levels of aperiodic tasks are simulated. Average power consumption is P_S, $1.5 \times P_S$ and $2 \times P_S$ for *Low*, *Medium* and *High* power levels respectively. In this experiment, 100,000 aperiodic tasks are scheduled for each simulation point. Figure 5.3 shows the measured value of average response time for different average power levels and arrival rates. The vertical lines in Figure 5.3 indicate the maximum arrival rate which can be supported for a given average power consumption, determined using Equation 5.5. Figure 5.3 also shows the analytical value of average response time which was determined using Equation 5.4. As shown in the figure, the response time of aperiodic tasks increases with an increase in arrival rate. Furthermore, response time also increases with an increase in average power consumption of aperiodic tasks. This is because aperiodic tasks with high power consumption are assigned larger deadlines. This increases the service time of aperiodic tasks; leading to an increase

in response time. For all simulations conducted, the measured value of average response time matches very closely to the theoretical value from Equation 5.4.

In the next experiment, we conduct extensive simulations with 100 different arrival rates (λ) and 90 aperiodic task power consumptions (P). Each distinct (λ, P) pair constitutes a single simulation point in this experiment. For each simulation point, 100,000 aperiodic tasks are simulated with all of them having same power consumption. Figure 5.4 shows a heatmap, with color indicating response time. As shown in the figure, response time increases with an increase in arrival rate. An increase in power consumption does not cause an increase in response time when power consumption is less than P_S. This is because $D_j = D_j^C$ when $P_j \leq P_S$; and power consumption has no effect on aperiodic task execution rate. When $P_j > P_S$, $D_j = D_j^T$. Therefore, we see an increase in response time with an increase in power consumption $> 84.9W$. Figure 5.5 shows the maximum processor temperature for all simulation points. As shown in the figure, maximum temperature increases with an increase in aperiodic task power consumption $<= P_S$. When aperiodic task power consumption is $> P_S$, the maximum unadjusted temperature remains constant at $75°$C; since temperature is regulated by executing aperiodic tasks at lower rates based on their power consumption. Figure 5.4 and 5.5 also show $U_A = \lambda \cdot \max(\frac{P \cdot 0.1}{P_S U_A^C}, \frac{0.1}{U_A^C}) = 1$ curve. Area under this curve identifies the region where the response times of aperiodic tasks are bounded (stable system).

Next, we compare the performance of D-T2BS with T2BS. The execution interval length for D-T2BS is 1ms. In this experiment we simulate 10,000 aperiodic tasks with power consumption equal to $P_S \times 1.5$ and $\lambda = 2$. Figure 5.6 shows the histogram of unadjusted temperature for T2BS and D-T2BS. As shown in the figure, T2BS and D-T2BS show similar temperature distribution.

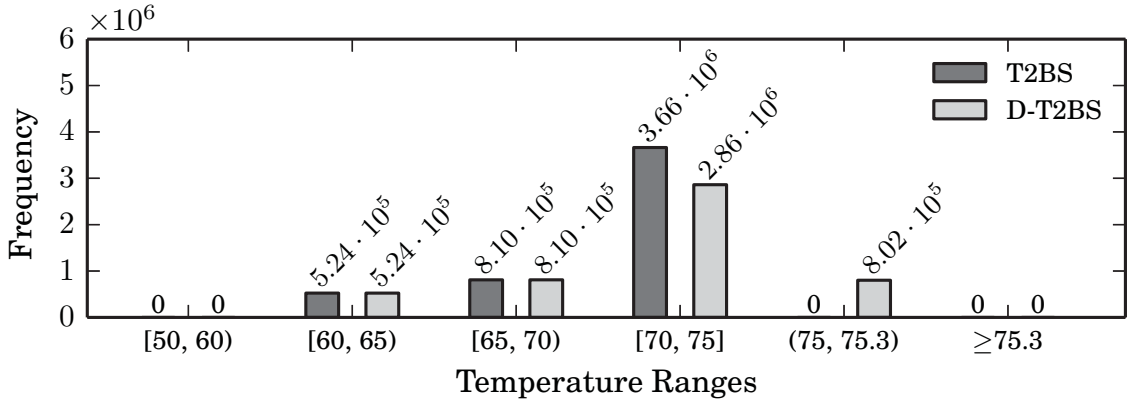


Figure 5.6: Histogram of temperature for T2BS and D-T2BS.

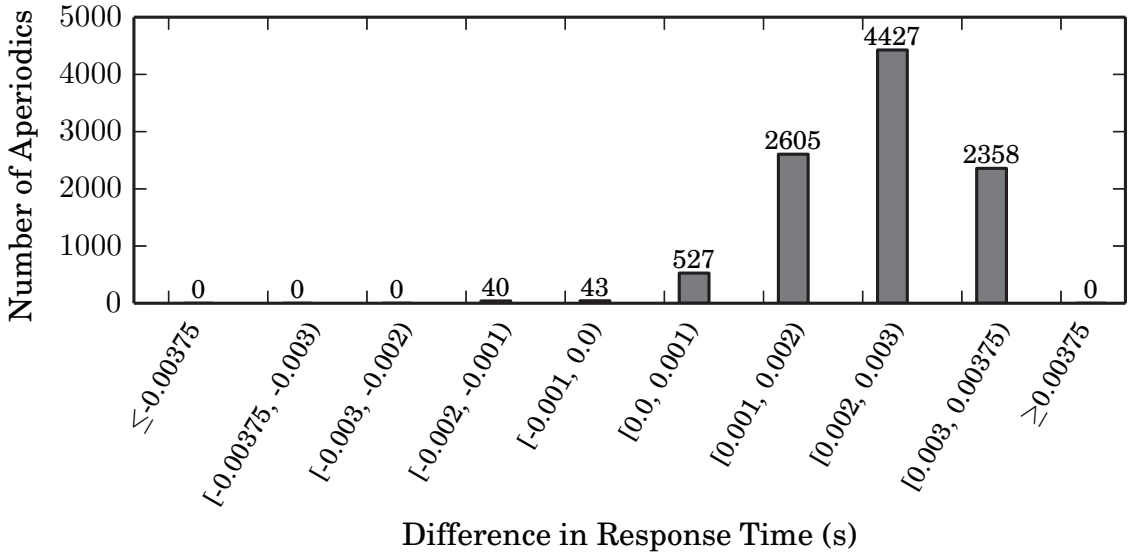


Figure 5.7: Histogram of response time difference between D-T2BS and T2BS.

T2BS never exceeds threshold temperature. D-T2BS, however, has some thermal violations with maximum unadjusted temperature equal to ($75.245°C$). This increase in temperature is due to discretization of the fluid T2BS schedule. In Figure 5.7, we plot the difference in response times of individual aperiodic tasks in T2BS and D-T2BS schedules. As stated in Section 5.3, the absolute difference in response time should never exceed $\frac{\text{Execution Interval}}{\min(U_A^C, \Delta U_A^T/(\zeta P_j))} = \frac{0.001}{\min(0.4, 0.267)} = 0.00375$.

As shown in Figure 5.7, no aperiodic task has absolute difference in response time ≥ 0.00375; empirically verifying the theoretical result.

In the next chapter, we introduce the tradeoff that exists between Computation Utilization and Thermal Utilization when the processing system is DVFS capable. Strategies to minimize thermal utilization by speed scaling are also presented in the next chapter .

Chapter 6

Thermal Utilization and Speed Scaling

The theoretical results presented in Chapters 3 and 4 were based on the assumption that speeds of all periodic tasks are fixed. Although different periodic tasks can have different speeds, all instances of a given periodic task τ execute at the same speed s_τ. This chapter first explores the tradeoff that exists between computation utilization and thermal utilization. Later part of this chapter explains how speeds can be assigned to different periodic tasks in order to minimize temperature. It is assumed that processor speed is upper and lower bounded by $s_{\max}$ and $s_{\min}$ respectively.

6.1 Thermal Utilization/Computation Utilization Tradeoff

For a single periodic task τ, the computation utilization and thermal utilization can be computed by the following expressions:

Computation Utilization$= \dfrac{C_\tau}{T_\tau s_\tau}$.

Thermal Utilization$= \dfrac{\zeta}{\Delta} \cdot \dfrac{P_\tau C_\tau}{T_\tau s_\tau} = \dfrac{\zeta}{\Delta} \cdot \dfrac{a_\tau C_\tau s_\tau^2}{T_\tau}$

The work covered in this chapter appears in [11] and is under review in [48]
Parameswaran Ramanathan, Kewal K. Saluja, "On thermal utilization of periodic task sets in uni-processor systems" RTCSA 2013
Rehan Ahmed, Parameswaran Ramanathan, Kewal K.Saluja "Necessary and Sufficient Conditions for Thermal Schedulability of Periodic Real-Time Tasks", Submitted to ACM TECS.

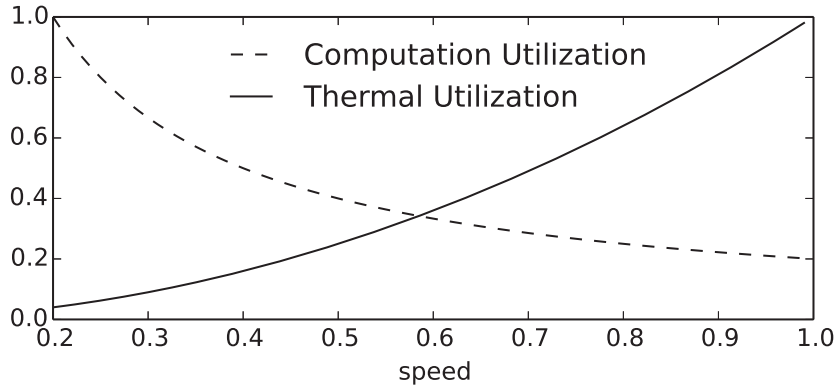


Figure 6.1: Variation in computation and thermal utilization of a periodic task with speed.

Based on these expressions, computation utilization of a periodic task is inversely proportional to the speed of execution; while thermal utilization is directly proportional to the square of execution speed. This tradeoff between computation and thermal utilization is illustrated in Figure 6.1. Based on this trend, it is possible to reduce thermal utilization; and consequently maximum temperature; by reducing the speed of execution. However, decrease in speed increases the computation utilization. Therefore, the avoid timing violations, we have to ensure that the computation utilization of the periodic taskset does not exceed 1.

The next immediate question is how can speed be assigned/changed such that thermal utilization/maximum processor temperature is minimized. We illustrate the effect of speed scaling on the thermal utilization of a periodic task by the following example:

Consider a periodic taskset with two periodic tasks. Tasks have following computation and thermal utilizations at speed 1:

Task1: Computation Utilization = 0.3, Thermal Utilization =0.3

Task2: Computation Utilization = 0.2, Thermal Utilization =0.9

Task 2 has a higher thermal utilization compared to task 1 due to higher activity (a_τ). Specifically, the activity of task 2 is 4.5 times larger than the activity of task 1.

Figure 6.2 illustrates how the computation and thermal utilization of each periodic taskset changes as speed is increased from 0.2 ($s_{\min}$) to 1 ($s_{\max}$). The objective is to minimize the sum of thermal utilizations of the two periodic tasks (dashed curves) while ensuring that the sum of computation utilizations of the two periodic tasks (solid curves) is ≤ 1 (To ensure timing feasibility). Figure 6.3 shows how the thermal utilization of periodic taskset changes when task 1 speed is varied. The speed of task 2 is set such that the computation utilization of the periodic taskset is 1. i.e. for a given task 1 speed (s_1), $\frac{C_{\text{task1}}}{T_{\text{task1}} s_1} + \frac{C_{\text{task2}}}{T_{\text{task2}} s_2} = 1$. For the periodic taskset in this example, the optimal task speed assignment makes the computation utilization of the periodic taskset 1 [11]. Therefore, the minimum point in Figure 6.3 is the minimum possible thermal utilization. At the optimal point, task 1 and task 2 are executed at speeds 0.630 and 0.381 respectively. This speed assignment increases the computation utilization to 1 and decreases the thermal utilization to 0.25 (compared to computation utilization of 0.5 and thermal utilization of 1.2 when both periodic tasks are executed at speed 1).

6.2 Thermal Utilization Minimization

Now, we will now go over speed assignment approaches for minimizing thermal utilization. According to Equation 3.13, the thermal utilization of a periodic taskset is given by the following

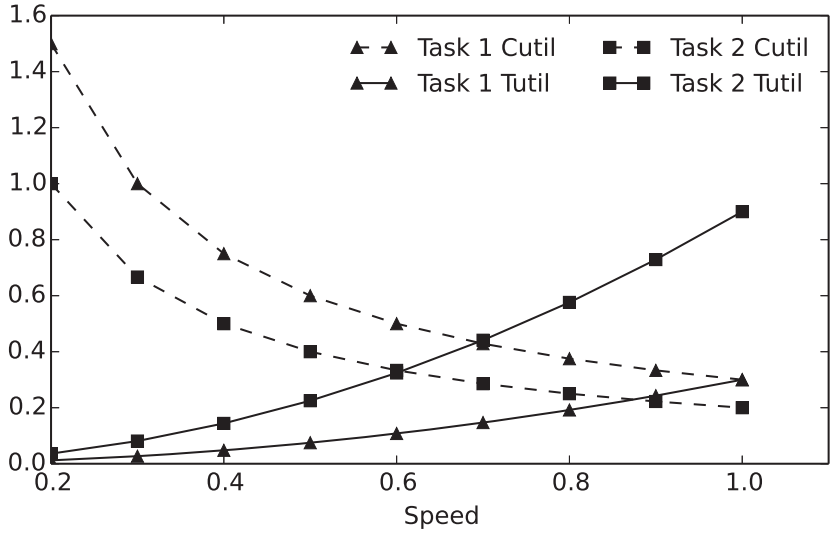


Figure 6.2: Variation in computation and thermal utilization of a periodic tasks with speed.

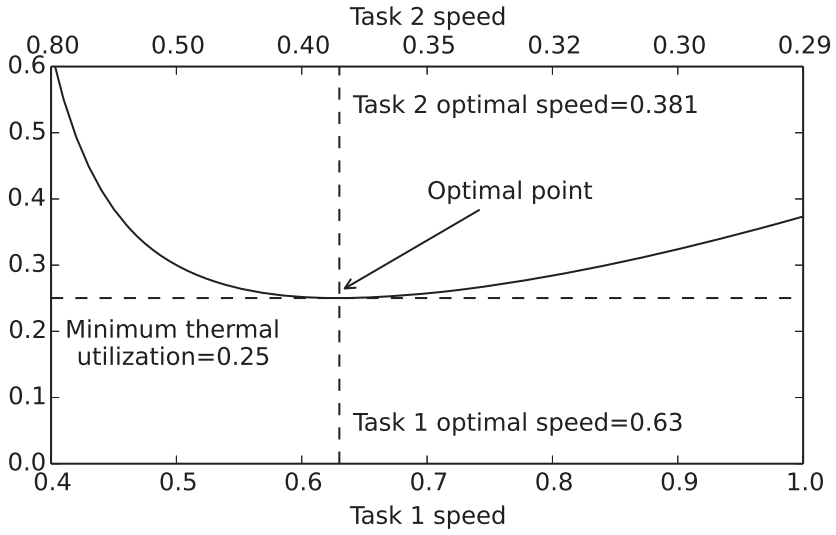


Figure 6.3: Variation in thermal utilization of periodic taskset wrt task speeds.

expression:

$$\Upsilon(\Gamma) = \frac{\zeta}{\Delta} \sum_{\tau \in \Gamma} \frac{P_\tau C_\tau}{T_\tau s_\tau} = \frac{\zeta}{\Delta} \sum_{\tau \in \Gamma} \frac{a_\tau C_\tau s_\tau^2}{T_\tau}.$$

Therefore, minimizing thermal utilization involves finding values of $s_\tau \forall \tau \in \Gamma$ such that the value of summation is minimized and computation utilization of the periodic taskset is ≤ 1. Based on the power consumption of periodic tasks, the energy consumption during one hyperperiod is given by the following equation:

$$\text{Energy Consumption} = L \cdot \sum_{\tau \in \Gamma} \frac{P_\tau C_\tau}{T_\tau s_\tau} = L \cdot \sum_{\tau \in \Gamma} \frac{a_\tau C_\tau s_\tau^2}{T_\tau}$$

Since the summations in both equations are same, minimizing thermal utilization also minimizes energy consumption and vice-versa. Therefore, we can use existing approaches in literature which minimize the energy consumption of periodic tasks by speed scaling. When, all periodic tasks have the same activity (a_τ), the solution is straightforward. Aydin et al.[49], prove that when all periodic tasks have same activity (a_τ), speed scaling all periodic tasks uniformly such that the computation utilization of the periodic taskset becomes 1, minimizes the energy consumption. For the more generalized task power model adopted in this work, where different periodic tasks can have different activity, Aydin et al. [50] give an order n^3 algorithm for minimizing energy; where n is the number of tasks. We now propose an order n^2 algorithm for minimizing thermal utilization which is shown to be optimal in majority of the cases. The reduced complexity is beneficial for run-time/dynamic energy minimization strategies.

For notational brevity, we define the following term:

Definition 6.1. *For* $Z \subseteq \Gamma$

$$G(Z) = \sum_{i \in Z} \frac{C_i}{T_i}(a_i)^{1/3}$$

Theorem 6.2. *If all instances of periodic task* i *are executed at speed:*

$$s_i^* = a_i^{-1/3} G(\Gamma) \qquad \forall i \in \Gamma \tag{6.1}$$

then thermal utilization of periodic taskset Γ *is minimized.*

Proof: It has already been established that, if we minimize energy of a periodic task set, its thermal utilization is also minimized. We know from [49], that energy is minimized if speed scaling is performed such that all tasks have same power consumption and the periodic task utilization is equal to 1, i.e.,

$\sum_{i \in \Gamma} \frac{C_i}{T_i s_i^*} = 1$ and $a_i s_i^{*3} = a_j s_j^{*3} = P_{\mathrm{T}} \qquad \forall i, j \in \Gamma$

P_{T} represents the power consumption of all tasks after optimal speed scaling. With this power consumption:

$s_i^* = \left(\frac{P_{\mathrm{T}}}{a_i}\right)^{1/3} \qquad \forall i \in \Gamma$ Therefore

$\sum_{i \in \Gamma} \frac{C_i}{T_i} \left(\frac{a_i}{P_{\mathrm{T}}}\right)^{1/3} = 1$

$P_{\mathrm{T}}^{1/3} = \sum_{i \in \Gamma} \frac{C_i}{T_i} {a_i}^{1/3} = G(\Gamma)$. The expression for s_i^* in Theorem 6.2 follows. ■

Unfortunately, it may not be possible to execute all tasks at speeds given by Theorem 6.2 because $S_i^* \notin [s_{\min}, s_{\max}]$ for some task i, where $s_{\min}$ and $s_{\max}$ are the processor speed constraints. We now present an algorithm to identify the speed solution which minimizes thermal utilization of the periodic task set Γ for the special case when there is no constraint on the minimum speed i.e $s_{\min} = 0$. Without loss of generality, $s_{\max}$ is assumed to be 1. Algorithm NoMinSpeed is iterative.

Algorithm 1 NoMinSpeed: Assigning speeds to periodic tasks when $s_{\min} = 0$ and $s_{\max} = 1$.

Input: Periodic task set Γ

Output: Optimal speed setting $s_i \ \ \forall i \in \Gamma$

1: Let $\Gamma_X = \emptyset, \Gamma_{\mathrm{UX}} = \Gamma$

2: $Z_X = \left(i : \left(a_i^{-1/3} G(\Gamma_{\mathrm{UX}}) \right) \Big/ \left(1 - \sum_{j \in \Gamma_X} \frac{C_j}{T_j s_j} \right) > 1, \ \ i \in \Gamma_{\mathrm{UX}} \right)$

3: **while** $\Gamma_{\mathrm{X}} \neq \emptyset$ **do**

4: $\quad s_i = 1 \ \forall i \in M_X$

5: $\quad \Gamma_x = \Gamma_x \cup Z_X$

6: $\quad \Gamma_{\mathrm{UX}} = \Gamma_{\mathrm{UX}} - Z_X$

7: $\quad Z_X = \left(i : \left(a_i^{-1/3} G(\Gamma_{\mathrm{UX}}) \right) \Big/ \left(1 - \sum_{j \in \Gamma_X} \frac{C_j}{T_j s_j} \right) > 1, \ \ i \in \Gamma_{\mathrm{UX}} \right)$

8: **end while**

9: $s_i = \left(a_i^{-1/3} G(\Gamma_{\mathrm{UX}}) \right) \Big/ \left(1 - \sum_{j \in \Gamma_X} \frac{C_j}{T_j s_j} \right) \qquad \forall i \in \Gamma_{\mathrm{UX}}$

Γ_X is a set of periodic tasks for which speed decision has been made. Initially, this set is empty. Conversely, Γ_{UX} is a set of periodic tasks for which speed decision has not been made. This set initially consists of all periodic tasks in Γ. In line 2, Z_X is the set of periodic tasks which violate the maximum speed constraint. These tasks are then assigned speed 1($s_{\max}$) in line 4. Z_X is then recomputed from the set of periodic tasks for which speeds have not been assigned in line 7. This process of computing and assigning speed 1 to all tasks in Z_X is repeated until the re-computation of Z_X in line 7 yields an empty set. After the termination of while loop, the remaining tasks are assigned speeds in line 9.

Theorem 6.3. *Algorithm NoMinSpeed is optimal in terms of thermal utilization minimization when* $s_{\min} = 0$

Proof: Given in Appendix A ■

Algorithm 2 SeCTUM: Assigning speeds to periodic tasks when $s_{\min} \neq 0$ and $s_{\max} = 1$.

Input: Periodic task set Γ
Output: Optimal speed setting $s_i \ \ \forall i \in \Gamma$

1: Let $\Gamma_X = \emptyset, \Gamma_{\mathrm{UX}} = \Gamma$
2: $Z_X = \left(i : \left(a_i^{-1/3} G(\Gamma_{\mathrm{UX}}) \right) \Big/ \left(1 - \sum_{j \in \Gamma_X} \frac{C_j}{T_j s_j} \right) > 1, \ \ i \in \Gamma_{\mathrm{UX}} \right)$
3: **while** $\Gamma_{\mathrm{X}} \neq \emptyset$ **do**
4: $\quad s_i = 1 \ \forall i \in Z_X$
5: $\quad \Gamma_x = \Gamma_x \cup Z_X$
6: $\quad \Gamma_{\mathrm{UX}} = \Gamma_{\mathrm{UX}} - Z_X$
7: $\quad Z_X = \left(i : \left(a_i^{-1/3} G(\Gamma_{\mathrm{UX}}) \right) \Big/ \left(1 - \sum_{j \in \Gamma_X} \frac{C_j}{T_j s_j} \right) > 1, \ \ i \in \Gamma_{\mathrm{UX}} \right)$
8: **end while**
9: $Z_N = \left(i : \left(a_i^{-1/3} G(\Gamma_{\mathrm{UX}}) \right) \Big/ \left(1 - \sum_{j \in \Gamma_X} \frac{C_j}{T_j s_j} \right) < s_{\min}, \ \ i \in \Gamma_{\mathrm{UX}} \right)$
10: **while** $\Gamma_{\mathrm{X}} \neq \emptyset$ **do**
11: $\quad s_i = s_{\min} \ \forall i \in Z_N$
12: $\quad \Gamma_x = \Gamma_x \cup Z_N$
13: $\quad \Gamma_{\mathrm{UX}} = \Gamma_{\mathrm{UX}} - Z_X$
14: $\quad Z_N = \left(i : \left(a_i^{-1/3} G(\Gamma_{\mathrm{UX}}) \right) \Big/ \left(1 - \sum_{j \in \Gamma_X} \frac{C_j}{T_j s_j} \right) < s_{\min}, \ \ i \in \Gamma_{\mathrm{UX}} \right)$
15: **end while**
16: $s_i = \left(a_i^{-1/3} G(\Gamma_{\mathrm{UX}}) \right) \Big/ \left(1 - \sum_{j \in \Gamma_X} \frac{C_j}{T_j s_j} \right) \qquad \forall i \in \Gamma_{\mathrm{UX}}$

If $s_{\min} \neq 0$, then it is possible that the speed assignment from algorithm NoMinSpeed is infeasible for some task i.e. for some task $s_i < s_{\min}$. Algorithm (SeCTUM)(Speed Constrained Thermal Utilization Minimization) strives to find the optimal speed assignment when $s_{\min} \neq 0$. Note that, without loss of generality, $s_{\max}$ can be assumed to be 1. Lines 1-8 of SeCTUM are exactly as lines 1-8 of algorithm NoMinSpeed. In line 9, Z_N is the set of tasks that violate the minimum speed constraint at the end of algorithm NoMinSpeed. These tasks are then assigned speed $s_{\min}$ and Z_N set is recomputed out of the set of periodic tasks for which speed assignments

have not been done. This process of computing and setting the speeds of tasks in it to is repeated in the second while loop (lines 10-15) until the computation of Z_N yields an empty set. After the termination of second while loop, the remaining periodic tasks are assigned speeds in line 16.

Theorem 6.4. *If $s_i^* \leq s_{\max}$ or $s_i^* \geq s_{\min} \ \ \forall i \in \Gamma$ where s_i^* is as defined in Equation 6.1 , then the speed solution given by SeCTUM will be optimal.*

Proof: See Appendix B. ■

If the conditions in Theorem 6.4 are not satisfied, the solution from Algorithm SeCTUM may be suboptimal. In these instances, the solution from Algorithm SeCTUM can be further improved using the following simple modification:

Algorithm 3 I-SeCTUM: Assigning speeds to periodic tasks when $s_{\min} \neq 0$ and $s_{\max} = 1$.

1: Let S_{SC1} be the speed solution given by SeCTUM
2: Let S_{SC2} be the speed solution given by an altered version of SeCTUM where the assignment of set Z_N and second while loop (lines 9-15) are executed before the assignment of set Z_X and first while loop (lines 3-8)
3: If speed solution S_{SC2} is computationally feasible, then pick speed solution $S \in \{S_{\text{SC1}}, S_{\text{SC2}}\}$which gives the minimum thermal utilization. Otherwise, $S = S_{\text{SC1}}$

6.3 Simulation Results

We now investigate the impact of speed scaling on thermal utilization. We also compare I-SeCTUM with optimal speed assignment strategy. Figure 6.4 illustrates the reduction in thermal utilizations of all periodic tasksets evaluated in Section 4.4, after all periodic tasks are optimally speed scaled such that the taskset thermal utilization is minimized.

In this figure, color represents:

$$\frac{\text{Thermal Utilization after speed scaling}}{\text{Thermal utilization at speed 1}}.$$

Therefore, a lighter color shows higher reduction in thermal utilization. For the results presented here, $s_{\min} = 0.2$ and $s_{\max} = 1$. As shown in the figure, when base thermal utilization of the periodic taskset is small, thermal utilization of the periodic taskset can be reduced substantially by executing periodic tasks at lower speeds (thermal utilization reduces quadratically with linear decrease in speed). However, as computation utilization of the periodic taskset increases, there is less margin to execute individual periodic tasks at lower speeds. Therefore, there is smaller reduction in thermal utilization at higher computation utilizations. Furthermore, the fractional reduction in thermal utilization is only a function of computation utilization; i.e. it is agnostic to the thermal utilization of the periodic taskset at speed 1.

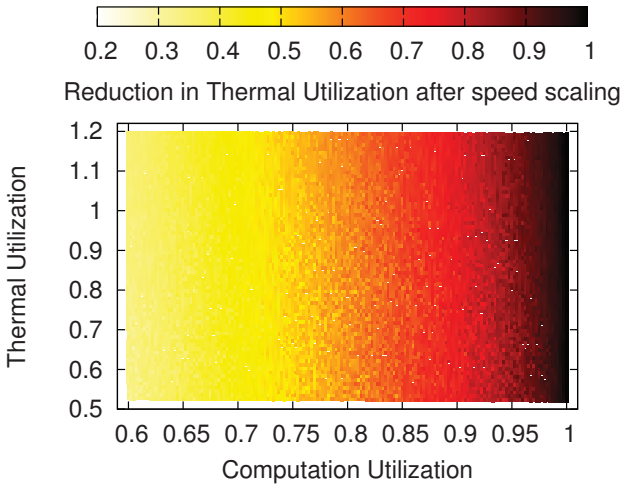


Figure 6.4: Fractional reduction in thermal utilization of periodic tasksets after speed scaling.

We now compare I-SeCTUM with an optimal speed assignment strategy for more than 10,000 periodic tasks with computation utilization varying between [0.65, 0.99] and thermal utilization varying between [0.3, 1.4]. For this analysis, $s_{\min} = 0.65$ and $s_{\max} = 1.0$. Following three schemes are evaluated:

a) Optimal: Results corresponding to Optimal speed assignment solution.

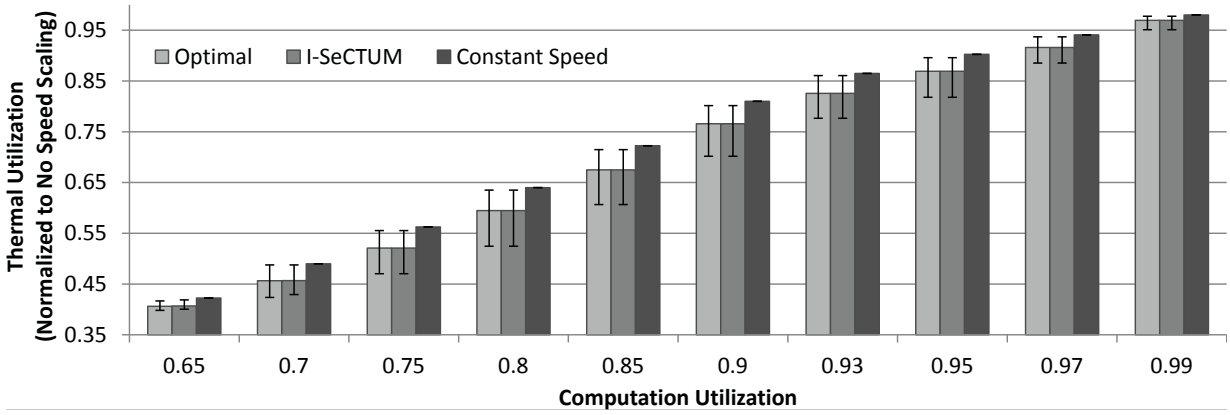


Figure 6.5: Thermal Utilization of Optimal, I-SeCTUM and Constant Speed schemes normalized to the thermal utilization without speed scaling.

b) I-SeCTUM: Results corresponding to speed assignment solutions obtained from heuristic I-SeCTUM.

c) Constant Speed: In this speed assignment approach, each periodic task is executed at the same constant speed such that the computation utilization of the periodic task set becomes 1; subject to the minimum speed constraint. This is similar to the approach presented in [49] for energy minimization.

Figure 6.5 plots the resulting thermal utilization after speed scaling for the three algorithms, for different groups of task sets categorized based on their computation utilization. For each group (corresponding to a particular computation utilization), 1000 periodic task sets are simulated. All results are normalized to the thermal utilization without speed scaling, and the average of normalized thermal utilizations for each group is plotted in Figure 6.5. Three key observations can be made from this figure:

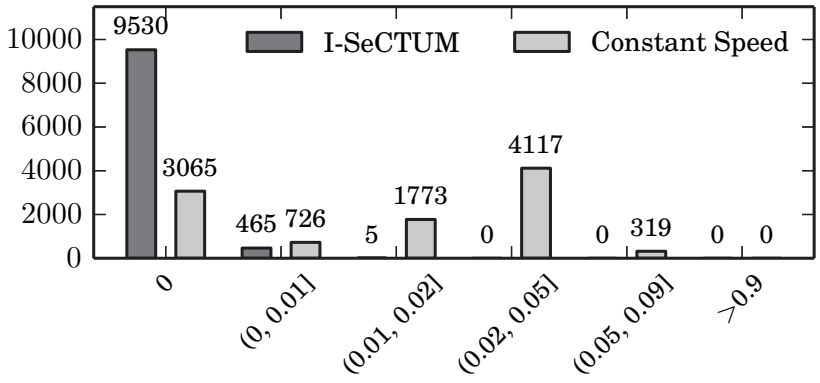


Figure 6.6: Difference in thermal utilizations of I-SeCTUM and Constant Speed schemes from optimal speed assignment.

1. When computation utilization is small, speed scaling can significantly reduce thermal utilization. For instance, when computation utilization is 0.65, the thermal utilization after speed scaling is less than 45% of the thermal utilization without speed scaling. This result is expected because the processor speed can be more easily reduced, without violating timing constraints, when computation utilization is low.

2. Algorithm I-SeCTUM is better than constant speed scheme. In other words, unlike [49], it is better to run each task at an individual selected speed rather than a single speed for all tasks. This qualitative difference in result arises because Algorithm I-SeCTUM is more general in the sense that it does not assume that all tasks consume the same power when executed. Instead, I-SeCTUM accounts for differences in activity (hence power consumption) between the tasks.

3. Algorithm I-SeCTUM is almost as effective as optimal scheme irrespective of the computation utilization. This is highly desirable because the runtime complexity of I-SeCTUM is less than the optimal scheme. In fact, I-SeCTUM can be used at runtime to reclaim processing times unused by other task instances. This is a significant factor because real-time tasks are often scheduled based on worst case execution times which are usually much larger than the actual execution times. The effectiveness of I-SeCTUM as compared to Optimal is further demonstrated in Figure 6.6, where a histogram of thermal utilization difference between I-SeCTUMs and Optimals solutions is plotted. Observe that in more than 95% of the case, I-SeCTUMs solution exactly matches the optimal solution. Almost all of the remaining suboptimal solutions have difference in thermal utilization of less than 0.01. Constant speed, on the other hand, is suboptimal in considerably more cases.

In the next chapter we extend the concepts of Thermal Impact/Utilization to a multi-core processing environment. We also present a thermally optimal partitioned scheduling solution for implicit deadline periodic tasks and corresponding simulation results. Evaluation of the proposed multi-core solution on a hardware testbed is also presented.

Chapter 7

Thermal-Aware Multi-Core Scheduling

In this chapter we extend the concepts of thermal impact/utilization presented in Chapter 3 to a multi-core platform and propose optimal scheduling strategies. However, before going into details of the proposed concepts, it is important to first define the processor and execution models which have been used in literature:

A multi-core platform may be heterogeneous, homogeneous or uniform:

Heterogeneous: The cores are different. Therefore, the computation time of a task is dependant on the core it is executing on.

Homogeneous: Each core is identical and executing at the same speed. Therefore, the computation time of a task is not a function of the core it is executing on.

Uniform: The cores are identical. However, they are operating at different speeds. The computation time of all tasks is inversely proportional to the speed of the core.

The scheduling schemes can be broadly characterized as *Global* and *Partitioned* approaches.

The work covered in this Chapter appeared in [42] and its extension [48] which is currently under review
Rehan Ahmed, Parameswaran Ramanathan, Kewal K. Saluja, "Necessary and Sufficient Conditions for Thermal Schedulability of Periodic Real-Time Tasks", ECRTS 2014.
Rehan Ahmed, Parameswaran Ramanathan, Kewal K.Saluja "Necessary and Sufficient Conditions for Thermal Schedulability of Periodic Real-Time Tasks", Submitted to ACM TECS.

Global: Tasks are allowed to migrate from one core to another.

Partitioned: Tasks are assigned to cores and are not allowed to migrate across cores.

In this chapter, we propose a thermally optimal partitioned scheduling solution for implicit deadline periodic tasks. The proposed concepts are initially evaluated on a homogeneous multi-core platform. However, in Section 7.6, an extension for heterogeneous multi-core platform is discussed. We also present the implementation and evaluation of the proposed concepts on a hardware testbed.

7.1 Computational Feasibility

In this section, we succinctly cover the research that has been done for scheduling implicit deadline periodic tasks on multi-cores without any consideration of thermal constraints. Dhall and Liu [51] studied the global scheduling problem. They showed that if global EDF is used to schedule all periodic tasks of a homogeneous m core system, then the worst-case schedulable utilization is one. This occurs when we have m tasks of computation time 2ϵ and period 1 (light tasks), and one task with computation time 1 and period $1 + \epsilon$ (heavy task). Due to earlier deadline the m light tasks, they will be executed concurrently on m cores and finish at time 2ϵ. The heavy task will start executing at time 2ϵ and will therefore miss its deadline at time $1 + \epsilon$. As $\epsilon \rightarrow 0$, the maximum schedulable utilization approaches 1. Notice that we had the same schedulable utilization on a uni-core executing periodic tasks using EDF. Therefore global EDF is not able of leverage the additional computation power of a multi-core processor. This phenomena has since been called *Dhall Effect*. To alleviate this issue, researchers have proposed several priority based algorithms.

The most notable of these algorithms is EDF-US[ς] [52]. EDF-US[ς] assigns priority to tasks in the following manner:

1. Assign highest priority to all tasks τ with computation utilization $\frac{C_\tau}{T_\tau s_\tau} > \varsigma$.

2. For periodic tasks τ with computation utilization $\leq \varsigma$, assign priority based on its deadline.

Using this priority assignment strategy, if $\varsigma = \frac{m}{2m-1}$, the schedulable utilization of m core processor is increased from 1 to $\frac{m^2}{2m-1}$. Setting $\varsigma = 1/2$, we can achieve maximum schedule utilization of $\frac{m+1}{2}$. P-fair [44] is another approach for scheduling periodic tasks on multi-cores. In this strategy, individual periodic tasks are broken down into smaller high frequency tasks and scheduled across the multi-core chip. With this strategy, an ideal schedulable utilization of m is achieved at the cost of higher number of task preemptions/migrations.

Various partitioned and hybrid (mix of partitioned and global) approaches have also been proposed in literature[51][53][54][55]. The advantage of partitioned scheduling is that, after partitioning, the problem reduces to uni-core scheduling problem. Therefore, after partitioning, wealth of research on uni-core scheduling can be applied. However, task partitioning problem is similar to bin packing problem and is known to be NP-Hard [56]. Furthermore, the maximum worst-case schedulable utilization for implicit deadline periodic tasks with partitioned scheduling is $\frac{m+1}{2}$ [57]. This is because regardless of the partitioning strategy used, $m + 1$ tasks with computation time $1 + \epsilon$ and deadline 2 can not be scheduled on an m-core processor.

7.2 Necessary Condition For Thermal Schedulability on Multi-Core

To extend the concept of Thermal Impact presented in Chapter 3, we first redefine Unit Thermal Impact (ζ). In a multi-core environment, the power consumption on a given core j results in temperature increase of all cores $i \in M$. Suppose that a single instance of unit power executes for one second on core j. Also suppose that the initial reference temperature of all cores is 0. Let $\theta(t)_{i,j}$ be the temperature of core i at time t as a result of this execution. Then, we redefine Unit Thermal Impact for multi-core ($\boldsymbol{\zeta}$) as an $m \times m$ matrix where $m = |M|$ and the element $\zeta_{i,j}$ of this matrix is such that:

$$\zeta_{i,j} = \int_0^\infty \theta(t)_{i,j} dt \tag{7.1}$$

Due to the linearity of the system, if a single instance of a task with power P executes for C seconds on the core j, then its *Total Thermal Impact* will be a vector equal to $PC \cdot \boldsymbol{\zeta} \boldsymbol{e}_j$ where $\boldsymbol{e}_j$ is the j^{th} unit vector. Similarly, due to linearity of the system, it follows from superposition principle that total thermal impact of executing more than one instance is the sum of the total thermal impacts of each instance.

If we repeatedly execute a periodic schedule Π, the temperature of all cores at the start of the hyperperiod will converge to a vector $\boldsymbol{\Theta}_{\boldsymbol{\Pi}}$ such that the processor temperature at the end of hyperperiod approaches $\boldsymbol{\Theta}_{\boldsymbol{\Pi}}$. This scenario where the temperature at the start of hyperperiod approaches temperature at the end of hyperperiod, as before, is called *thermal steady state*.

Theorem 7.1. *Consider a periodic taskset Γ, with assignment Λ and schedule $\Pi(t)$. Also suppose that $\boldsymbol{\Theta}(0) = \boldsymbol{\Theta}_{\boldsymbol{\Pi}}$ and assignment Λ is such all instances of periodic task τ executes on core j in*

schedule $\Pi(t)$*, then*

$$\int_0^L \mathbf{\Theta}(t)dt = L \cdot \sum_{\tau \in \Gamma} \frac{P_\tau C_\tau}{T_\tau s_\tau} \boldsymbol{\zeta} \boldsymbol{e}_j \tag{7.2}$$

Proof: This theorem is an extension of Theorem 3.5 and follows from the definition of $\boldsymbol{\zeta}$ matrix and the linearity of RC thermal model. We call the value on integral in Equation 7.2 the Hyperperiod Thermal Impact of periodic taskset Γ with assignment Λ (**HTI**(Γ, Λ)). Note that (**HTI**(Γ, Λ)) is a vector with number of elements equal to the number of cores. ■

Theorem 7.2. *Consider a periodic taskset* Γ *with schedule* Π *starting at an initial temperature* $\mathbf{\Theta}(0) = \mathbf{\Theta}_\Pi$*. Let* $\boldsymbol{\phi}(\Pi)_i$ *be the corresponding maximum temperature of core* i *during the periodic schedule, i.e.,* $\phi(\Pi)_i = \max_{0 \le t \le L}(\Theta(t)_i)$*. Then,*

$$\phi(\Pi)_i \geq \frac{\textit{HTI}(\Gamma, \Lambda)_i}{L}. \tag{7.3}$$

Proof: This theorem specifies a lower bound on the maximum temperature of each core $i \in M$ for periodic taskset Γ and assignment Λ. This theorem is a direct extension of Theorem 3.6 in Chapter 3. ■

Based on this lower bound on temperature, we also define thermal utilization of core $i \in M$ for a given periodic taskset Γ and assignment Λ:

$$\Upsilon(\Gamma, \Lambda)_i = \frac{\text{HTI}(\Gamma, \Lambda)_i}{L\Delta_i} \tag{7.4}$$

Theorem 7.3. *If thermal utilization of any core* $i > 1$*, then no scheduling algorithm can meet thermal constraint for* i*.*

Proof: This theorem follows from the lower bound on temperature given in Theorem 7.2 and the definition of $\Upsilon(\Gamma, \Lambda)_i$. ■

7.2.1 Taskset Thermal Utilization

Since the integral of processor temperature is not constant for a periodic taskset and speed assignment (it also depends on assignment Λ), *Thermal Utilization* of a periodic taskset in multi-core processing environment can not be defined as it was done in Chapter 3 for uni-core. For multi-core, the thermal utilization of periodic taskset Γ is defined as:

$$\Upsilon(\Gamma) = \inf_{\Pi} \left\{ \max_{i \in M} \left(\frac{\phi(\Pi)_i}{\Delta_i} \right) \right\} \tag{7.5}$$

where $\phi(\Pi)_i$ is the maximum temperature of core i when periodic schedule $\Pi(t)$ is executing on the multi-core system and Δ_i is the threshold temperature for core i. i.e. $\Upsilon(\Gamma)$ is the lower bound on $\max_{i \in M} \left(\frac{\phi(\Pi)_i}{\Delta_i} \right)$ for any schedule ($\Pi(t)$) of periodic taskset Γ. Based on this definition, thermal utilization of ≤ 1 is a necessary condition for thermal feasibility of a periodic taskset.

We find the value of $\Upsilon(\Gamma)$ for the following two cases: 1) Fixed task speeds 2) Variable task speeds. In the first case, $\Upsilon(\Gamma)$ is computed while keeping speeds of all periodic tasks in Γ fixed. In the later case, this fixed speed assumption is relaxed. To evaluate the value of $\Upsilon(\Gamma)$, in both cases, we solve optimization formulations. We first relax the deadline constraints of periodic taskset Γ by transforming it to taskset $\widehat{\Gamma}$. For each task $\tau \in \Gamma$, there is a corresponding task $\hat{\tau} \in \widehat{\Gamma}$ such that: $P_{\hat{\tau}} = P_\tau$, $C_{\hat{\tau}} = \frac{C_\tau L}{T_\tau}$, $T_{\hat{\tau}} = L$. Note that the amount of computations done for a given periodic task in Γ in one hyperperiod are the same as the computations done for the corresponding task in $\widehat{\Gamma}$. However, $\widehat{\Gamma}$ does not have any task deadlines within the hyperperiod. We also relax the *No Migration* constraint on the execution of periodic task instances.

Let $x_{\hat{\tau},j}$ be a positive variable which is the amount execution done for periodic task $\hat{\tau} \in \widehat{\Gamma}$ on core $j \in M$ in periodic schedule $\widehat{\Pi(t)}$. Based on the values of $x_{\hat{\tau},j}$, the integral of temperature of core i for one hyperperiod at thermal steady state is given by:

$$\int_0^\infty \Theta(t)_i dt = \sum_{\hat{\tau} \in \widehat{\Gamma}} \sum_{j \in M} x_{\hat{\tau},j} a_{\hat{\tau}} s_{\hat{\tau}}^3 \zeta_{i,j} \qquad \forall i \in M \tag{7.6}$$

where $\mathbf{\Theta}(0) \approx \mathbf{\Theta}(L)$. Then based on Theorem 7.2, lower bound on the value of $\frac{\phi(\widehat{\Pi})_i}{\Delta_i}$ is given by

$$\frac{\phi(\widehat{\Pi})_{i,\mathrm{LB}}}{\Delta_i} = \frac{1}{L \cdot \Delta_i} \sum_{\hat{\tau} \in \widehat{\Gamma}} \sum_{j \in M} x_{\hat{\tau},j} a_{\hat{\tau}} s_{\hat{\tau}}^3 \zeta_{i,j} \qquad \forall i \in M \tag{7.7}$$

For the fixed speed case, we assume that the speed of each periodic task $\hat{\tau} \in \widehat{\Gamma}$ is s_τ where s_τ is the fixed speed of its corresponding task in Γ. We find the optimal values of $x_{\hat{\tau},i}$ variables, such that $\Upsilon_{\max} = \max\limits_{i \in M} \left(\dfrac{\phi(\widehat{\Pi})_{i,\mathrm{LB}}}{\Delta_i} \right)$ is minimized with the following constraints:

$$\sum_{\hat{\tau} \in \Gamma} x_{\hat{\tau},i} \leq L \qquad \forall i \in M \tag{7.8}$$

$$\sum_{i \in M} x_{\hat{\tau},i} = \frac{C_{\hat{\tau}}}{s_{\hat{\tau}}} \qquad \forall \hat{\tau} \in \widehat{\Gamma} \tag{7.9}$$

Constraint in Equation 7.8 ensures that no core is assigned more computation that it is able to complete within the hyperperiod. Constraint in Equation 7.9 ensures that all tasks in the periodic taskset $\widehat{\Gamma}$ complete within the hyperperiod. Since $\widehat{\Gamma}$ is a relaxation of periodic taskset Γ with all deadline constraints removed, optimal value of $\Upsilon_{\max}$ is the lower bound of $\inf\limits_{\Pi} \left\{ \max\limits_{i \in M} \left(\dfrac{\phi(\Pi)_i}{\Delta_i} \right) \right\}$, where $\Pi(t)$ is a schedule of periodic taskset Γ and the speed of all periodic tasks $\tau \in \Gamma$ are fixed at s_τ. Therefore: $\Upsilon(\Gamma) = \Upsilon_{\max}$ in the optimal solution of the LP problem presented in this section.

The lower bound on maximum temperature of periodic taskset Γ can be computed as $\Upsilon(\Gamma) \cdot \Delta_i$ where $i = \underset{i \in M}{\arg\max} \left(i : \frac{\phi(\widehat{\Pi})_{i,\mathrm{LB}}}{\Delta_i} \right)$.

For the variable speed case, $s_{\widehat{\tau}}$ is also considered a decision variable in the optimization formulation. The objective and constraints remain the same. However, considering $s_{\widehat{\tau}}$ a decision variable makes the optimization problem non linear; increasing its complexity.

7.3 Thermal Aware Multi-Core Scheduling

In this section, we present our proposed scheduling and assignment approach. We propose the following three solutions which vary in the way speed assignment is done for periodic tasks:

1. TrUMPS: (**T**he**r**mal **U**tilization **M**inimization with **P**artitioned **S**cheduling) assumes that the speeds of all periodic tasks are fixed.

2. TrUMPS^2: (**T**he**r**mal **U**tilization **M**inimization with **P**artitioned **S**cheduling and Speed Assignment) assumes that the each periodic task can execute at a different speed and speeds can be changed independently.

3. TrUMPS-C: (**T**he**r**mal **U**tilization **M**inimization with **P**artitioned **S**cheduling and **C**onstant Core Speed) assumes that the speed of a core can not change. However, different cores can execute at different speeds.

All of these schemes have the same basic structure. They first solve an optimization problem to find the value of task assignment (task and speed assignment for TrUMPS^2 and TrUMPS-C) such that the maximum thermal utilization across all cores is minimized. After the assignment stage,

GPS/WF2Q is used on each individual core to minimize its maximum temperature. GPS ensures that the maximum value of temperature of each core is equal to its thermal utilization times its threshold temperature i.e.($\Upsilon(\Gamma,\Lambda)_i \cdot \Delta_i$). We now explain each of these schemes in detail.

7.3.1 TrUMPS

TrUMPS assumes that the speed of each periodic task is fixed/predetermined. Therefore, in this solution, we first find the task to core assignments such that maximum value of core Thermal Utilizations ($\Upsilon(\Gamma,\Lambda)_i$) across all cores is minimized. The objective of this optimization problem is: Minimize $\Upsilon_{\max} = \max_{m \in M}(\Upsilon(\Gamma,\Lambda)_m)$ subject to:

$$\sum_{\tau \in \Gamma} \frac{C_\tau}{T_\tau s_\tau} \boldsymbol{e}_j \leq \mathbf{1} \quad \text{where task } \tau \text{ is assigned to core } j \text{ in } \Lambda \tag{7.10}$$

Where $\Upsilon(\Gamma,\Lambda)_m$ is defined in Equation 7.4. This optimization formulation finds the optimal value of task to core assignments (Λ) such that the maximum value of thermal utilization across all cores is minimized. Constraint given in Equation 7.10 ensures that the computation utilization of each individual core is ≤ 1. Once we find the optimal task-to-core assignments, GPS/WF2Q is used on individual cores to minimize their temperature. Since we minimize the maximum value of core thermal utilization in the assignment stage of the proposed solution, we conjecture that TrUMPS with GPS yields the minimum value of maximum unadjusted temperature for a given periodic taskset when execution of tasks is constrained by *No Migration* model and task speeds are fixed.

7.3.2 TrUMPS^2

TrUMPS^2 assumes that the speed of each periodic task can be changed independently. However, all executions of a given periodic task are done at the same speed. In this solution, we first

find the task to core assignments and speed to task assignments such that maximum value of core Thermal Utilizations ($\Upsilon(\Gamma, \Lambda)_i$) across all cores is minimized. The optimization formulation for TrUMPS^2 is similar to the formulation for TrUMPS given in Section 7.3.1 except that the speed of each periodic task s_τ is also a decision variable. The introduction of task speed as a decision variable introduces following two additional constraints:

$$s_{\min} \leq s_\tau \leq s_{\max} \qquad \forall \tau \in \Gamma$$

The optimization problem also becomes non-linear by considering speed a decision variable. However, the complexity of the optimization problem is not large considering that it needs to be solved a-priori. In this solution, since we minimize the maximum value of core thermal utilization in the assignment stage, we conjecture that TrUMPS^2 scheme yields the minimum value of maximum unadjusted temperature for a given periodic taskset when execution of tasks is constrained by *No Migration* model. However, TRUMPS^2 has a high speed change overhead, since system frequency/voltage can potentially change after each preemption. For systems where the overhead for changing frequency/voltage is high, we propose the next solution (TrUMPS-C)

7.3.3 TrUMPS-C

TrUMPS-C jointly finds the task to core assignment and the speed at which each core performs execution; such that maximum value of core thermal utilization across all cores is minimized. Once the speed of a core is assigned, it does not change temporally. Therefore there is no speed change overhead during scheduling. We use s_m to represent speed of a core $m \in M$. With core speeds

fixed, the thermal utilization of each core is given by the following equation:

$$\Upsilon(\Gamma, \Lambda, S)_i = \frac{1}{\Delta_i} \left(L \cdot \sum_{\tau \in \Gamma} \frac{P_\tau C_\tau}{T_\tau s_k} \boldsymbol{\zeta} \boldsymbol{e}_k \right)_i \tag{7.11}$$

where Λ is the task to core assignment and S represents the core speeds in the multi-core system. Each task τ executes on core k with speed s_k. In TrUMPS-C solution, we minimize:

$$\Upsilon_{\max} = \max_{i \in M} (\Upsilon(\Gamma, \Lambda, S)_i) \tag{7.12}$$

with the following constraints:

$$\sum_{\tau \in \Gamma} \frac{C_\tau}{T_\tau s_j} \boldsymbol{e}_j \leq 1 \qquad \text{where task } \tau \text{ is assigned to core } j \text{ in } \Lambda \tag{7.13}$$

$$s_{\min} \leq s_j \leq s_{\max} \qquad \forall j \in M \tag{7.14}$$

These constraints ensure that the computation utilization of each core ≤ 1 and the speed of each core is within the speed bounds, respectively. Like TrUMPS^2, TrUMPS-C is a Mixed Integer Non-Linear Programming problem. However, complexity of the optimization problem is not large considering that it needs to be solved a-priori. Once task to core assignments and core speeds are determined GPS/WF2Q is used to minimize temperature of each individual core. Since the speeds of all cores are fixed, there is no speed change overhead during scheduling; as opposed to TrUMPS^2 in which speed may change after each preemption. In the following section, we compare the performance of TrUMPS-C with TrUMPS^2 to illustrate that TrUMPS-C performs close to TrUMPS^2 with substantially lower scheduling overhead. We also compare TrUMPS-C with schemes that decouple speed and task assignments. This comparison is done to quantify the benefit of solving speed and task assignment problem jointly.

7.4 Results

In this section, we conduct simulations for thermal feasibility of several periodic tasksets on a 3-core system. In the first part of the results section, it is assumed that the speeds of all periodic tasks are fixed at 1. With this assumption, we compare the maximum temperatures given by the proposed TrUMPS assignment/scheduling strategy against the lower bound on maximum temperature for fixed task speeds; which is computed as explained in Section 7.2.1. Unadjusted temperature values are given in all the results. We then compare the joint speed and task assignment strategies (TrUMPS^2 and TrUMPS-C). TrUMPS^2 is compared against the lower bound on maximum temperature for variable task speeds (computed as explained at the end of Section 7.2.1). This is done to illustrate that for majority of the periodic tasksets, TrUMPS^2 reaches close to the lower bound on maximum temperature; even though lower bound is computed with task deadline and migration constraints/restrictions relaxed. TrUMPS-C is compared against TrUMPS^2 to show that it performs close to TrUMPS^2 with reduction in runtime overhead; due to no temporal speed transitions. Trumps-C is also compared against three static speed assignment schemes to quantify the benefit of solving speed/task assignment problem jointly. Details of the static schemes will be presented later in the results section.

7.4.1 Simulation Setup

We conducted simulations using HOTSPOT [39] to evaluate $\boldsymbol{\zeta}$ for our 3-core floorplan. For our simulations:

$$\boldsymbol{\zeta} = \begin{pmatrix} 0.72225 & 0.156 & 0.156 \\ 0.156 & 0.55375 & 0.16525 \\ 0.156 & 0.16525 & 0.55375 \end{pmatrix}$$

Unadjusted threshold temperature ($\widehat{\Delta}$) is assumed to be 75°C and $\boldsymbol{\Psi}_{\text{idle,ss}}$ is assumed to be 40°C for all cores. We use UUniFast Discard algorithm [58] to generate more than 10,000 periodic tasksets with computation utilizations varying from 1.5-3.0. The power consumption of the periodic tasks in a periodic taskset is uniformly distributed between 10W and 125W. Thermal utilizations of periodic tasksets vary between 0.5-1.1. All of these values are given at a speed of 1. Note that some periodic tasksets may not be computationally feasible even though their computation utilization is < 3 [59]. This is because there may be no possible task to core assignments which result in the computation utilization of each core being ≤ 1. In this section, only tasksets which are computationally feasible are included in the results.

7.4.2 Simulation Results - Fixed speed

Figure 7.1 shows the computation and thermal utilizations of all periodic tasksets. It shows the periodic tasksets which did not have any thermal violations using GPS, as gray dots. The periodic tasksets which have thermal violations are shown as black dots. Contrary to the results in the uni-core section, having thermal utilization ≤ 1 is not sufficient for thermal feasibility. This is because the periodic tasks considered in this section are deadline constrained and their execution

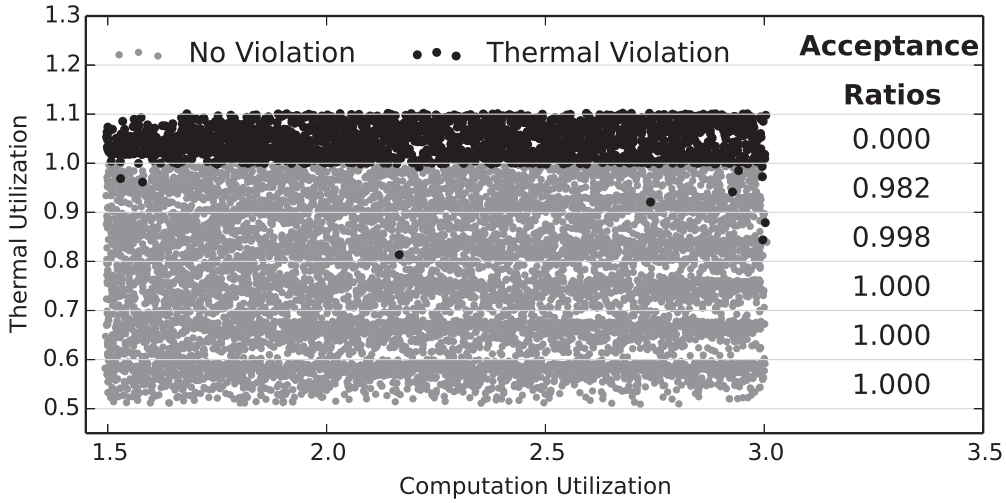


Figure 7.1: Computation and thermal utilization of periodic tasksets simulated; and their thermal feasibility using TrUMPS.

is constrained by *No Migration* Model. Both of these restrictions were removed when lower bound on temperature and thermal utilization was computed/defined in Section 7.2.1. Therefore, as Figure 7.1 shows, there are some periodic tasksets which violate thermal constraints when scheduled using TrUMPS; even though their thermal utilizations are < 1. Furthermore, Figure 7.1 shows that all periodic tasksets with thermal utilization > 1 violated thermal constraints. This is because having thermal utilization ≤ 1 is a necessary condition for thermal feasibility of periodic taskset.

Figure 7.2 shows the difference in maximum temperature given TrUMPS from the lower bound on temperature explained in Section 7.2.1. As shown in figure, more than 80% of the periodic tasksets have the difference of less than 0.1°C from the lower bound on temperature. Furthermore, there are very few periodic tasksets with difference greater than 4°C (7 periodic tasksets out of 10206), and no periodic tasksets with difference > 10. A large difference in temperature for some periodic tasksets is observed because in these tasksets, there is one periodic task with high thermal

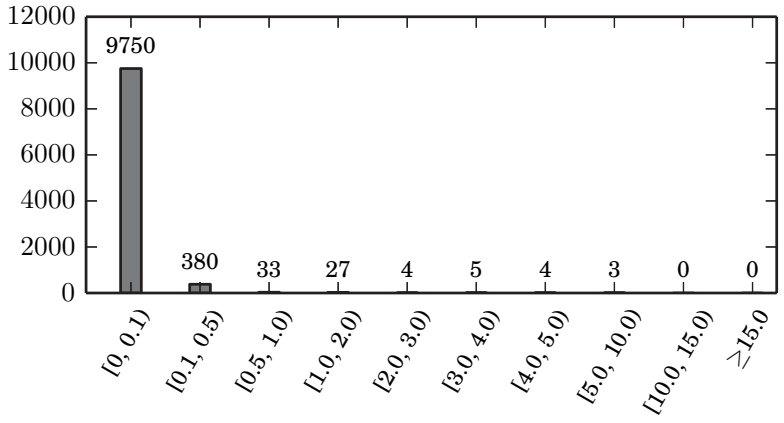


Figure 7.2: Difference in maximum temperature given by TrUMPS and the lower bound on temperature.

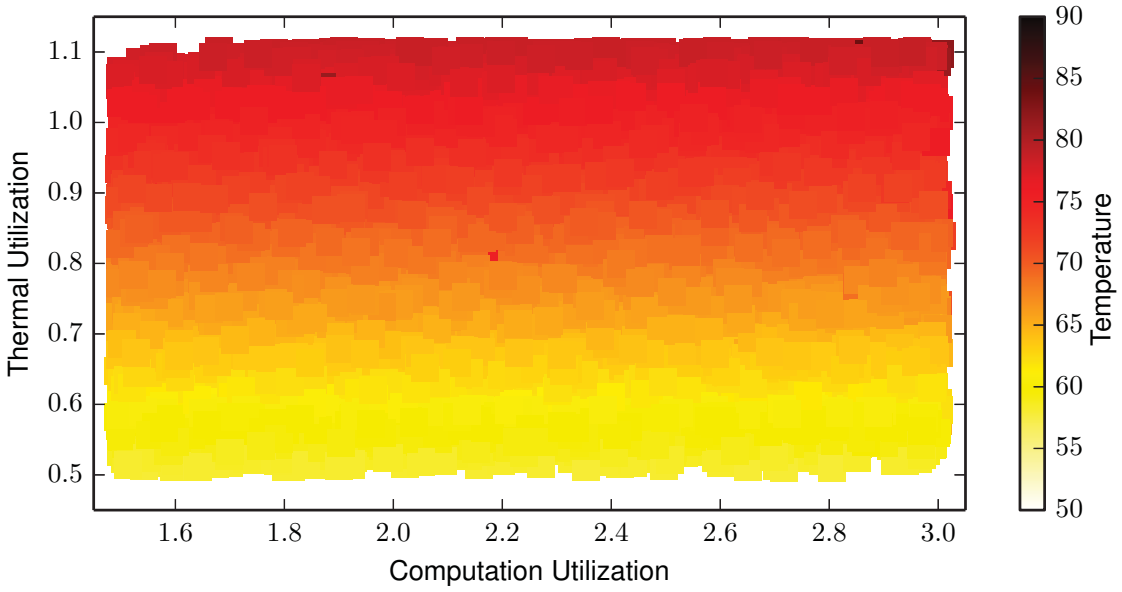


Figure 7.3: Maximum temperature given by the TrUMPS for all tasksets.

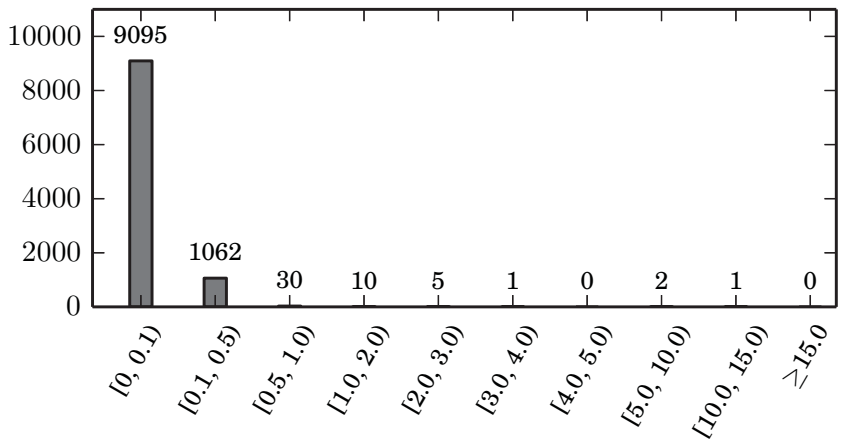


Figure 7.4: Distribution of temperature differences between TrUMPS^2 temperature lower bound.

utilization. The execution and thermal load of the *hot* task cannot be distributed across cores because of partitioned nature of our solution. For these types of periodic tasks (where there is one task with very high thermal utilization), adopting global scheduling strategies is expected to bring the maximum temperature closer to the lower bound. Figure 7.3 illustrates how the maximum temperature given by TrUMPS varies with the thermal and computation utilizations of periodic tasksets. As shown in the figure, the maximum temperature is not dependent on computation utilization. However, it rises linearly as thermal utilization is increased; which is the expected result.

7.4.3 Simulation Results - Variable Speed

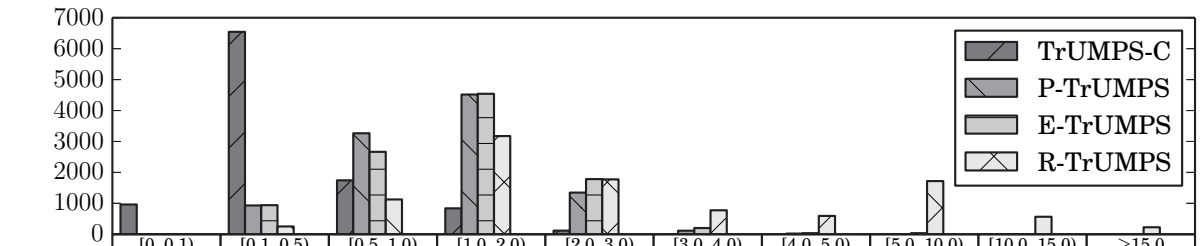
This section covers the results of multi-core scheduling when task speeds are not fixed. The first comparison in this section is the performance of TrUMPS^2 against the lower bound on processor

temperature with variable task speeds. Since TrUMPS^2 is a thermally optimal partitioned scheduling scheme, This comparison quantifies the loss in performance due to the partitioned nature of our scheduling scheme. Figure 7.4 plots the histogram of differences in maximum temperature between TrUMPS^2 and lower bound. The results show that we approach very close to the lower bound on temperature (difference of less than 0.1°C) for a high percentage of periodic tasksets (9095 out of 10206 tasksets). Furthermore, there are very few periodic tasksets which have high difference in maximum temperature from the lower bound. However, we did encounter one periodic taskset which showed a difference in maximum temperature of 10.57°C. The reason behind large difference was one periodic task with high computation and thermal utilization (computation utilization of 0.708 and thermal utilization of 0.574 at speed 1). The execution of this task could not be distributed across multiple cores due to the no-migration task execution model. Therefore, the core on which this task executed had a high maximum temperature.

Next, we compare TrUMPS-C against TrUMPS^2. We also provide analysis for the following three core speed assignment strategies:

1. **Proportional**: Speeds are assigned to cores inversely proportional to their self heating ζ coefficients. For the results presented in this section, since $\zeta_{1,1}$ is large compared to $\zeta_{2,2}$ and $\zeta_{3,3}$, core 1 is assigned a lower speed compared to core 2 and core 3 with the ratio 0.75:1.0:1.0. The sum of the speeds of the three cores is kept equal to the computation utilization of the periodic taskset.

2. **Equal**: All cores are assigned the same speed. The sum of the speeds of the three cores is kept equal to the computation utilization of the periodic taskset.



	[0, 0.1)	[0.1, 0.5)	[0.5, 1.0)	[1.0, 2.0)	[2.0, 3.0)	[3.0, 4.0)	[4.0, 5.0)	[5.0, 10.0)	[10.0, 15.0)	≥15.0
TrUMPS-C	962	6544	1743	838	118	1	0	0	0	0
P-TrUMPS	2	931	3265	4517	1347	115	22	6	1	0
E-TrUMPS	6	940	2665	4541	1783	201	35	35	0	0
R-TrUMPS	2	252	1125	3174	1774	775	593	1719	566	226

Figure 7.5: Distribution of temperature differences between core speed assignment strategies and TrUMPS^2.

3. **Random**: Speeds are assigned randomly to each core with the condition that the sum of the speeds of the three cores is equal to the computation utilization of periodic taskset.

Once speeds are assigned to cores, TrUMPS is used to assign tasks to different cores. Simulation results for *Proportional*, *Equal* and *Random* speed assignment schemes is added to quantify the benefit of solving the speed/task assignment problem jointly (As TrUMPS-C). Figure 7.5 shows the histogram of difference in maximum temperature, of all four core speed assignment schemes, compared to TrUMPS^2. As shown in the figure TrUMPS-C shows best performance with high number of periodic tasksets (90.6$\%$) having difference in maximum temperature of less than 1°C. Furthermore, there are no periodic tasksets which had a difference in maximum temperature of greater than 4°C. *Proportional* and *Equal* speed assignment strategies have lower performance compared to TrUMPS-C, with high number of periodic tasksets having difference in maximum temperature between 0.5 and 3 degrees. *Random* speed assignment scheme has the worst performance with substantial number of periodic tasksets having high difference in maximum temperature. Another

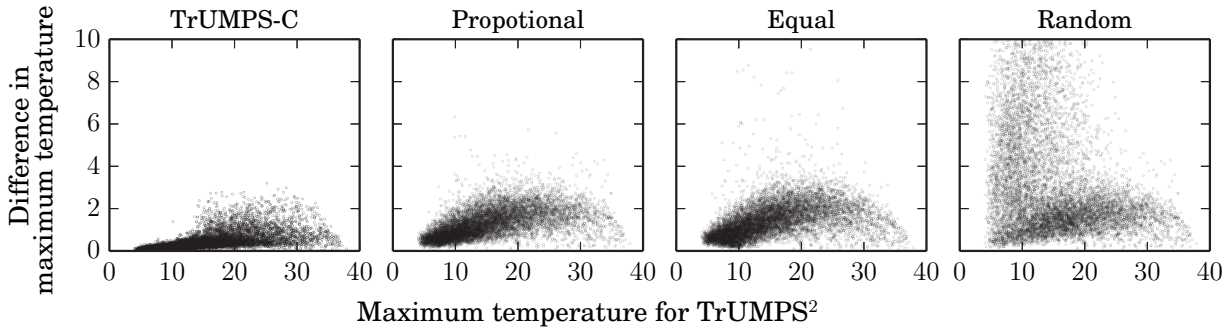


Figure 7.6: Difference in maximum temperature between core speed assignment strategies and TrUMPS^2.

visualization of these results is given in Figure 7.6. In this figure, we plot the difference in maximum temperature for each periodic taskset for the core speed assignment schemes from TrUMPS^2. TrUMPS-C has a temperature difference of less than 1°C for nearly all periodic tasksets. However, Proportional and Equal speed assignment schemes have substantial number of periodic tasksets with difference in maximum temperature greater than 2°C. Random speed assignment strategy has substantial number of periodic tasksets with high difference in maximum temperature.

7.5 Hardware Implementation/Analysis

We also implemented the proposed solution on a hardware testbed primarily to get rudimentary estimates on the thermal gains which are possible using the task scheduling scheme proposed in this work. We limit our evaluations to TrUMPS scheme without any speed scaling. Our testbed constitutes Linux with Preempt-RT patch running on Intel Core i7-3632QM processor. Preempt-RT patch adds real-time capability by making the kernel a preemptible process. It allows userspace applications to have higher priority than the kernel, allowing for deterministic execution times.

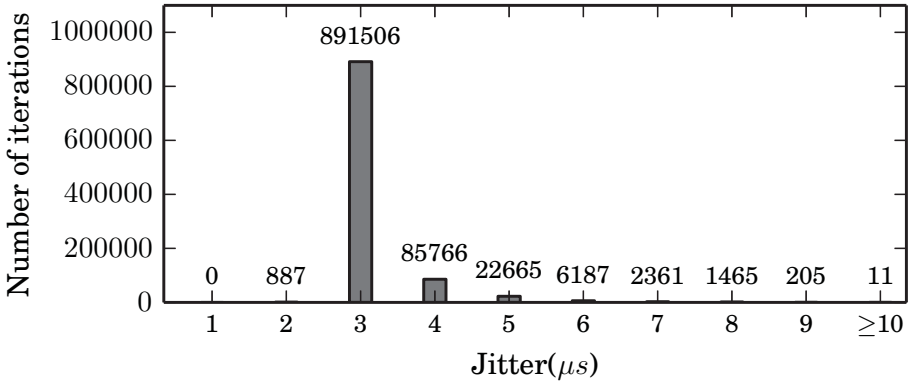


Figure 7.7: Distribution of jitter in sleep/wake time of high priority threads.

To determine real-time performance of our testbed, we ran the cyclictest application for 20 high priority real-time threads. This tool acquires time jitter by measuring accuracy of sleep and wake operations of highly prioritized real-time threads. A lower time jitter correlates to high real-time performance. Figure 7.7 shows the time jitter for more than 10 million sleep/wake iterations. As the figure shows, the value of jitter for most threads was about $3\mu s$. Furthermore, the maximum value of jitter we observed was $12\mu s$. We contend that this is reasonable real-time performance for our testbed implementation. Our testbed is composed of a pseudo-scheduler which is the highest priority task in the system. The scheduler spawns high priority real-time threads and controls their execution by sending Kill signals to their process IDs. The pseudo scheduler also has a thread which periodically monitors and stores processor temperature. On our quad core system, one of the cores is reserved for temperature monitoring thread. The remaining three cores run the real-time schedule. Since the computed schedule is static, there is no run-time overhead for determining the schedule.

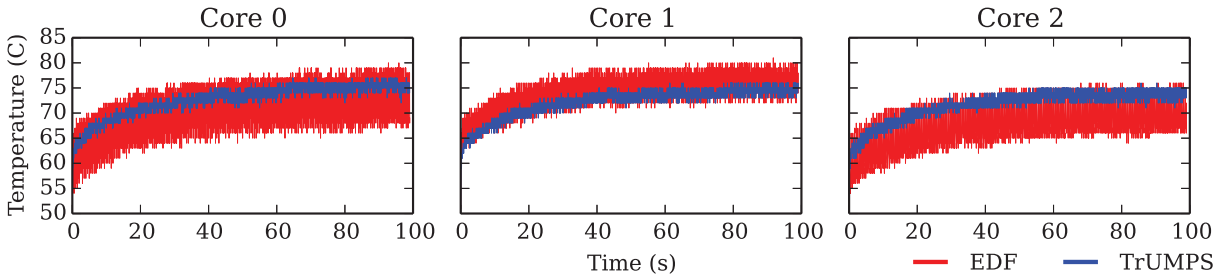


Figure 7.8: Core temperatures when EDF and TrUMPS are used to schedule periodic tasks.

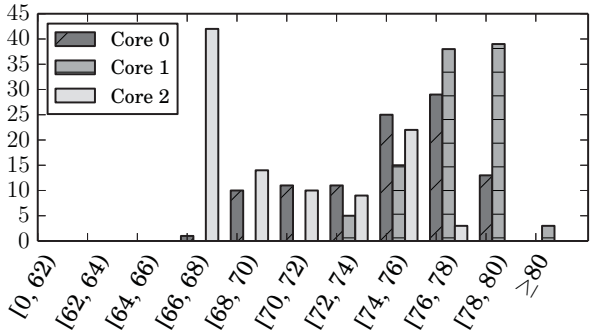


Figure 7.9: Histogram of temperature in steady state hyperperiod of EDF schedule.

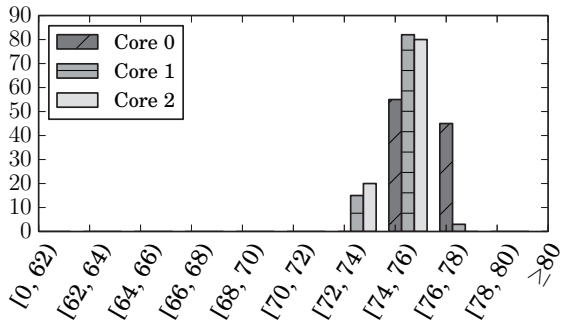


Figure 7.10: Histogram of temperature in steady state hyperperiod of TrUMPS schedule.

On our testbed, we schedule a periodic taskset composed of 10 periodic tasks. The periods of periodic tasks vary between $20ms$ to $1sec$ while their worst case execution time varies between $8ms$ and $65ms$. In this experiment, the computation time of all tasks is the worst-case computation time. The real-time tasks perform arithmetic operations (Add, Quick Sort, FFT). In our analysis, we compare TrUMPS with WF2Q (scheduling interval of 1ms) against partitioned-EDF. The temperature of each core at the start of experiment is $51°$C and temperature is monitored for a duration of 100s. Figure 7.8 shows the temperature of the three cores when EDF is used. As can

be seen, EDF causes the temperature of each core to vary significantly. The maximum temperature in the case of EDF across all cores is 81°C. When TrUMPS is used, there are less variations in processor temperature. Furthermore, the maximum temperature across all cores is now 77°C (A reduction of 4 degrees from EDF). Figure 7.9 shows the histogram of temperature readings from all three scheduling cores in the last hyperperiod when EDF is used for scheduling (hyperperiod where we have reached thermal steady state). The histogram results show significant variations in processor temperatures. For instance, the temperature of core 0 varies between 67°C and 79°C. Also the histogram in figure 7.9 shows that the thermal load across the cores is not balanced; since core 0 has majority of temperature readings below 70°C while core 1 has majority of temperature readings higher than 75°C. TrUMPS minimizes the processor temperature by distributing thermal load evenly and scheduling tasks using WF2Q so that temporal temperature variations are reduced. This is illustrated in temperature histogram for TrUMPS in figure 7.10. The temperature of all cores remains between 72°C and 76°C. The temperature across all cores is also more evenly equalized; resulting in maximum temperature reduction of 4°C. Figure 7.11 shows the maximum temperature across all cores. As shown in the figure, the maximum temperature for WF2Q scheme remains consistently below the maximum temperature for EDF.

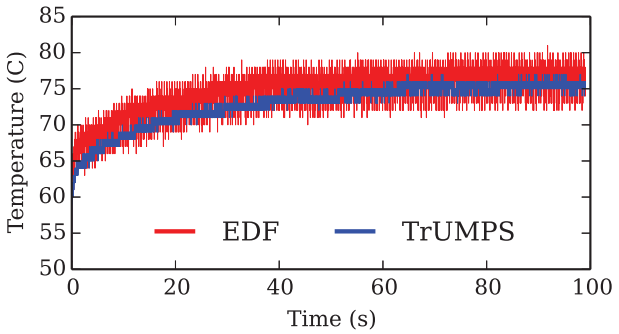


Figure 7.11: Maximum temperature across all cores for EDF and TrUMPS scheduling schemes.

7.6 Computation Heterogeneity

A given multi-core platform may be computationally hydrogenous. This implies that a periodic task may exhibit different worst-case execution time depending on the core it is executing on. Computation heterogeneity may be caused by a non-uniform processing platform. Although computational heterogeneity has not been explicitly evaluated in this section, it can easily be incorporated in the proposed solution.

This is done by considering computation time a core dependant parameter. To this end, we define $C_{i,j}$ as the worst-case computation time of task i on core j. Similarly, we may also consider power consumption a core dependant parameter ($P_{i,j}$) is the dynamic power consumption of task i on core j. With this generalization of worst-case computation time and power consumption, we can compute hyperperiod thermal impact for a given periodic taskset Γ and task to core assignment Λ:

$$\mathbf{HTI}(\Gamma,\Lambda) = \int_0^L \mathbf{\Theta}(t)dt \quad \text{initial condition } (\mathbf{\Theta}(0) = \mathbf{\Theta_\Pi}) = L \cdot \sum_{\tau\in\Gamma} \frac{P_{\tau,k}C_{\tau,k}}{T_\tau s_\tau}\boldsymbol{\zeta}\boldsymbol{e}_k \tag{7.15}$$

Where all instances of task τ are executed on core j and $\boldsymbol{e_j}$ is the j^{th} unit vector. The expression for thermal utilization is the same ($\Upsilon(\Gamma,\Lambda)_i = \frac{\text{HTI}_{(\Gamma,\Lambda)_i}}{L\Delta_i}$). For incorporating computation heterogeneity in TrUMPS, as before, we minimize thermal utilization across all core subject to the constraint:

$$\sum_{\tau\in\Gamma} \frac{C_{\tau,j}}{T_\tau s_\tau}\boldsymbol{e}_j \leq \mathbf{1} \quad \text{where task } \tau \text{ is assigned to core } j \text{ in } \Lambda \tag{7.16}$$

Similarly, the TrUMPS-C and TrUMPS^2 can also be extended to incorporate computational heterogeneity.

In the next chapter, we present a framework that enables solutions for other task/execution models and constraints. We also present simulation results for non-partitioned scheduling of implicit deadline periodic tasks on a multi-core platform.

Chapter 8

Generalized Multi-Core Scheduling Framework

Thus far, in the domain of multi-core, we have only considered partitioned solutions for implicit deadline periodic tasks. However, there are real-time applications characterized by other execution and task models. For instance, periodic tasks may have constrained deadlines i.e. deadlines $<$ period; or they may have additional constraints such as resource or precedence. Under resource constraints, tasks cannot execute concurrently on the multi-core because they require a shared resource for execution. Due to precedence constraints a task cannot execute until certain other task instances have completed their execution. In its current form, the periodic task scheduling strategies presented in this work do not account for these additional constraints.

In this section, we propose a framework for thermal constrained scheduling of periodic tasks with these additional execution constraints on a multi-core platform. The general idea is to divide the hyperperiod into intervals and then solve an optimization formulation to distribute thermal load across cores and across intervals for minimization of maximum temperature. The schemes presented in this section are heuristics with bounded deviation from optimal solutions.

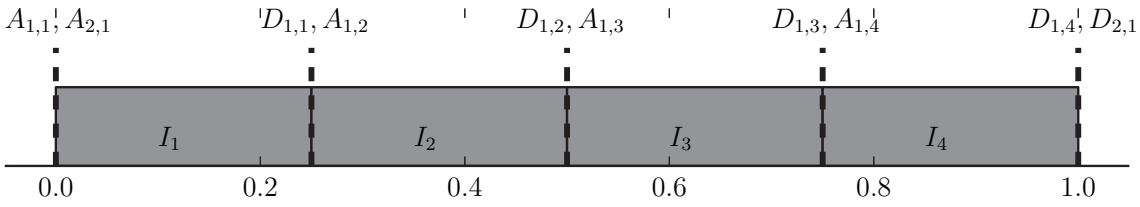


Figure 8.1: Interval assignment for the example periodic taskset. Minimum number of intervals.

8.1 Non-Partitioned Scheduling of Periodic Tasks

To introduce the solution strategy, we consider non-partitioned scheduling of periodic tasks. In the proposed solution, hyperperiod is divided into *intervals*. An interval i is characterized by an end time E_i and a start time which is equal to the end time of the interval before (Intervals are contiguous). The set of all intervals is called $I = (I_1, I_2 \cdots I_n)$. By definition $E_0 = 0$. Interval end times are chosen such that there are no task arrivals or deadlines within (E_{i-1}, E_i). and intervals cover the entire hyperperiod such that:

$$\sum_{i \in I} E_i - E_{i-1} = L$$

All task arrivals and deadlines occur at interval boundaries. As an example, consider a periodic taskset composed of two implicit deadline periodic tasks:

Task 1 $(C_1 = 0.1, T_1 = 0.25)$

Task 2 $(C_2 = 0.3, T_2 = 1.00)$

For this periodic taskset, arrivals and deadline for one hyperperiod are shown in Figure 8.1. In the figure $A_{i,j}$ denotes arrival of the j^{th} instance of task i while $D_{i,j}$ denotes deadline of the j^{th} instance of task i. Based on task arrivals and deadlines, hyperperiod can be divided into intervals

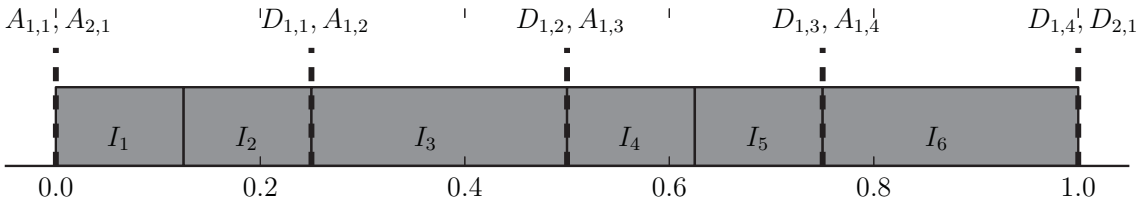


Figure 8.2: Intervals assignment for the example periodic taskset. Arbitrary interval assignment.

as shown in Figure 8.1. Note that Figure 8.1 shows the case where hyperperiod is divided into minimum number of intervals and each interval is as large as possible. This is not necessary and we may choose intervals that are smaller as shown in Figure 8.2. Smaller intervals give more scheduling flexibility and enable solutions to approach closer to the lower bound on temperature. However, the complexity for finding solution increases linearly as the number of intervals increase.

Now we present the optimization formulation used for non-partitioned scheduling of implicit deadline periodic tasks. Let I be a set of all intervals in the hyperperiod and let M be the set of all cores in the multi-core platform. For a given task i, we define variable $x_{i,j,k}$ as the units of execution performed for task i in interval j on core k. Based on this definition and the notations defined in Chapter 7, the Total Thermal Impact of all executions done in interval j is given by:

$$\int_{E_{j-1}}^{E_j} \boldsymbol{\eta}(t) \;\; dt \;+\; \int_{0}^{\infty} \mathbf{cool}(\boldsymbol{\eta}(E_j), t) \;\; dt = \sum_{\substack{i \in \Gamma \\ k \in M}} x_{i,j,k} P_i \boldsymbol{\zeta} \boldsymbol{e_k} \tag{8.1}$$

We call this value *Interval Thermal Impact* where $\mathbf{ITI}(\Gamma, X)^j$ is the vector denoting total thermal impact of interval j for execution assignments X and periodic taskset Γ. In the current formulation, task speeds are assumed to be fixed. The thermal utilization of interval j on core k is likewise

defined as:

$$\Upsilon(\Gamma, X)_k^j = \frac{\text{ITI}(\Gamma, X)_k^j}{(E_j - E_{j-1})\Delta_k} \tag{8.2}$$

In order to minimize processor temperature, we solve the following optimization formulation:

Minimize $\left(\max_{\substack{j \in X \\ k \in M}} \left(\Upsilon(\Gamma, X)_k^j \right) \right)$

Subject to:

$$\alpha_{i,j,k} = \begin{cases} 1, & \text{if } x_{i,j,k} > 1 \\ 0, & \text{otherwise} \end{cases} \tag{8.3}$$

$$\sum_{\substack{i \in \Gamma \\ k \in M}} \alpha_{i,j,k} \leq 1 \qquad \forall j \in I \tag{8.4}$$

$$\sum_{i \in \Gamma} x_{i,j,k} \leq E_j - E_{j-1} \qquad \forall j \in I, \forall k \in M \tag{8.5}$$

$$\sum_{\substack{j \leq l \\ k \in M}} x_{i,j,k} = \frac{E_l}{T_i} \cdot \frac{C_i}{s_i} \qquad \forall i \in \Gamma, \forall \{l : \frac{E_l}{T_i} \in \mathbb{Z}\} \tag{8.6}$$

Constraint in Equation 8.4 ensures that tasks do not migrate across cores within an interval. Constraint in Equation 8.5 ensures that an interval is not assigned more execution than its capacity. Constraint in Equation 8.6 ensures that each periodic task instance finishes before its deadline.

Lemma 8.1. *If constraints in Equations* 8.*4,* 8.*5 and* 8.*6 are met and each task* i *is executed in interval* j *on core* k *at execution rate equal to* $\frac{x_{i,j,k}}{E_j - E_{j-1}}$*, then there will be no deadline violations*

Proof: Constraint in Equation 8.6 ensures that each task is assigned execution equal to its computation time within each period. This ensures that if each task receives execution equal to its execution assignment in each interval, all task deadlines will be met. With an execution rate equal to $\frac{x_{i,j,k}}{E_j - E_{j-1}}$, task i receives execution equal to $x_{i,j,k}$ in interval j on core k; proving the lemma. ■

Theorem 8.2. *For a periodic taskset* Γ *and interval execution assignment* X*, if constraints in Equation* 8.*4,* 8.*5 and* 8.*6 are satisfied and GPS is used to schedule tasks within each interval as in Lemma 8.1, then:*

- *All task deadlines will be met.*
- *Maximum processor temperature* $\Phi(\Gamma, X) \leq \left(\max_{\substack{j \in X \\ k \in M}} \left(\Upsilon(\Gamma, X)_k^j \cdot \Delta_k \right) \right)$.

Proof: By Lemma 8.1, all task deadlines will be met. To prove the maximum temperature bound, we consider the following two cases:

(a) $\Upsilon(\Gamma, X)_k^j = \max_{\substack{j \in I \\ k \in M}} \left(\Upsilon(\Gamma, X)_k^j \right)$

(b) $\Upsilon(\Gamma, X)_k^j \leq \max_{\substack{j \in I \\ k \in M}} \left(\Upsilon(\Gamma, X)_k^j \right)$

In case (a), the thermal utilization of all intervals on all cores is equal:

$\boldsymbol{\Upsilon}(\Gamma, X) = \max_{\substack{j \in X \\ k \in M}} \left(\Upsilon(\Gamma, X)_k^j \right) \cdot \mathbf{1}$

This results in constant power consumption throughout the hyperperiod equal to $\boldsymbol{\zeta}^{-1}(\boldsymbol{\Upsilon}(\Gamma, X) \circ \boldsymbol{\Delta})$. This constant power constant power consumption results in constant temperature throughout the hyperperiod equal to $\boldsymbol{\Upsilon}(\Gamma, X) \circ \boldsymbol{\Delta}$. Since the thermal utilization across all cores is equal, all cores will have the same temperature equal to $\max_{\substack{j \in X \\ k \in M}} \left(\Upsilon(\Gamma, X)_k^j \cdot \Delta_k \right)$.

Since $\Upsilon(\Gamma, X)_k^j$ in case (b) is $\leq \Upsilon(\Gamma, X)_k^j$ in case (a), $\forall j \in I,\ \forall k \in M$, the maximum temperature in case (b) is no more than the maximum temperature in case (a); proving the theorem. ■

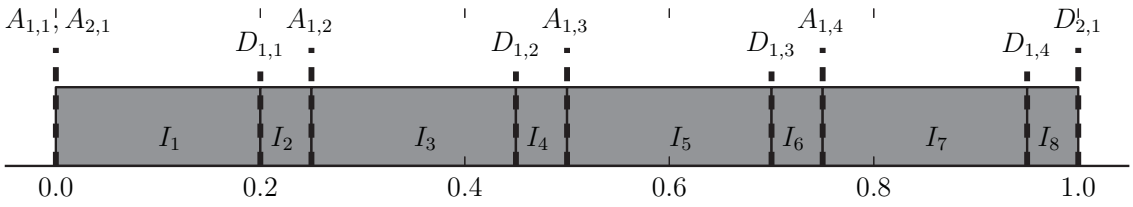


Figure 8.3: Interval assignment for the constrained deadline periodic taskset. Minimum number of intervals.

8.2 Adaptations of Scheduling Framework

In this section, we extend the optimization formulation to find solutions for constrained deadline periodic tasks, tasks with resource conflicts and tasks with precedence constraints.

8.2.1 Constrained deadline periodic tasks

Constrained deadline periodic tasks are defined as periodic tasks where task deadline is less than or equal to period. If a constrained deadline periodic task i is scheduled in accordance with optimization formulation covered in Section 8.1, it may finish right before the arrival of next instance. Therefore, if $D_i < T_i$, deadline violations will occur. In this subsection, we introduce an additional constraint which has to be considered in addition to constraints given in Equation 8.4, 8.5 and 8.6 to correctly schedule constrained deadline periodic tasks.

$$\sum_{\substack{j \leq l \\ k \in M}} x_{i,j,k} = \frac{E_l - T_i + D_i}{T_i} \cdot \frac{C_i}{s_i} \qquad \forall i \in \Gamma, \forall \{l : \frac{E_l - T_i + D_i}{T_i} \in \mathbb{Z}\} \tag{8.7}$$

Constraint in Equation 8.7 ensures that each periodic task finishes on or before its deadline. Note that, intervals have to be constructed such that there are no periods, arrivals or deadlines within an

interval. As an example, we consider the following constrained deadline periodic taskset:

Task 1 $(C_1 = 0.1, T_1 = 0.25, D_i = 0.20, a_1 = 80)$

Task 2 $(C_2 = 0.3, T_2 = 1.00, D_i = 1.00, a_2 = 120)$

Note that this periodic taskset is similar to the one we considered earlier in Section 8.1 with the exception that Task 1 now has a constrained relative deadline of 0.2. The interval assignment for this periodic taskset with the minimum number of intervals in one hyperperiod is shown in Figure 8.3. Note that because of Task 1's constrained deadline, the number of intervals has increased from 4 in Figure 8.1 to 8 in Figure 8.3

8.2.2 Incorporation of Resource Constraints

If two or more tasks share a common resource, they cannot be executed concurrently. Such resource constraints can be incorporated in the scheduling framework by imposing additional constraint:

$$\text{resc}_{\text{i,j}} = \begin{cases} 1, & \text{if task } i \text{ and } j \text{ have a resource conflict} \\ 0, & \text{otherwise} \end{cases}$$

$$\sum_{k \in M} \alpha_{i,j,k} + \sum_{k \in M} \alpha_{l,j,k} \cdot \text{resc}_{i,l} \leq 1 \qquad \forall i, l \in \Gamma \;\; \forall j \in I \tag{8.8}$$

Constraint in Equation 8.8 ensures that tasks with resource conflicts are not assigned execution in the same interval.

8.2.3 Incorporation of Precedence Constraints

If periodic tasks have data dependencies, we have to introduce precedence constraints in our scheduling framework. It is assumed here that tasks which have data dependencies have the same

period. However, they can have different deadlines (constrained deadline periodic tasks), which are accounted for constraint given in Equation 8.7. To incorporate precedence constraints, we introduce the following variables:

$$\text{pre}_{i,j} = \begin{cases} 1, & \text{if task } j \text{ has to precede task } i \\ 0, & \text{otherwise} \end{cases}$$

$$\text{b\_fin}_{i,j} = \begin{cases} 0, & \text{if current instance of task } i \text{ finished before interval } j \\ 1, & \text{otherwise} \end{cases}$$

Binary variable $\text{b\_fin}_{i,j}$ can be set using the following constraints:

$$\left\lceil \frac{E_{j-1}}{T_i} \right\rceil \cdot \frac{C_i}{s_i} - \sum_{\substack{1 \le l < j \\ k \in M}} x_{i,l,k} \le L \cdot \text{b\_fin}_{i,j} \qquad \forall i \in \Gamma,\ \forall j \in (I \backslash I_1) \qquad \text{(8.9a)}$$

$$\left\lceil \frac{E_{j-1}}{T_i} \right\rceil \cdot \frac{C_i}{s_i} - \sum_{\substack{1 \le l < j \\ k \in M}} x_{i,l,k} \ge \text{b\_fin}_{i,j} \qquad \forall i \in \Gamma,\ \forall j \in (I \backslash I_1) \qquad \text{(8.9b)}$$

$$\text{b\_fin}_{i,1} = 1 \qquad \forall i \in \Gamma \qquad \text{(8.9c)}$$

Constraint in Equation 8.9a forces $\text{b\_fin}_{i,j}$ to be 1 if current instance of task i has not completed execution before interval j. Constraint in Equation 8.9b forces $\text{b\_fin}_{i,j}$ to be 0 if current instance of task i has completed execution. Constraint in Equation 8.9c sets the b_fin variable for the first interval. We now introduce a binary variable $\text{block}_{i,j,k}$ such that:

$$\text{block}_{i,j,k} = \begin{cases} 1, & \text{if task } i \text{ is blocked by task } j \text{ in interval } k \text{ due to precedence constraint} \\ 0, & \text{otherwise} \end{cases}$$

$\text{block}_{i,j,k}$ variable is set using the following constraints:

$$\text{block}_{i,j,k} \leq \text{b\_fin}_{j,k} \qquad \forall i,j \in \Gamma,\ \forall k \in I \tag{8.10a}$$

$$\text{block}_{i,j,k} \leq \text{pre}_{i,j} \qquad \forall i,j \in \Gamma,\ \forall k \in I \tag{8.10b}$$

$$\text{block}_{i,j,k} \geq \text{b\_fin}_{j,k} + \text{pre}_{i,j} - 1 \qquad \forall i,j \in \Gamma,\ \forall k \in I \tag{8.10c}$$

Finally, we can use the following constraint to ensure that precedence constraints of the periodic taskset are met:

$$\sum_{k \in M} x_{i,j,k} \leq \frac{C_i}{s_i} \cdot (1 - \text{block}_{i,l,j}) \qquad \forall i,l \in \Gamma,\ \forall j \in I \tag{8.11}$$

Equation 8.11 ensures that if a periodic task i is blocked by any other periodic task in interval j, its execution assignment during interval j is 0.

In similar fashion, we can add additional constraints in this scheduling framework based on the execution requirements/constraints of the real-time application. Note that, all optimization formulations/constraints presented thus far are mixed integer linear programs. In the next section, we will extend the framework to be DVFS capable. This adaptation, however, makes the optimization model non-linear; significantly increasing its complexity.

8.2.4 Per-Interval Speed Solutions

In the previous formulations, each task was assigned a different fixed speed. Unless all tasks are assigned the same speed, the speed change overhead may be significant; since the speed can potentially change after each preemption. In this subsection, we present a formulation which assigns speeds to *intervals* such that thermal utilization and temperature is minimized. In this solution, speed change overhead is bounded since speed transition occurs only at interval boundaries. For

this formulation, we introduce the variable $s_{i,j}$ which denotes the speed of core j during interval i. It is assumed that the speed does not change throughout the interval. With interval speed assignments S, Interval Thermal Impact is redefined as:

$$\mathrm{ITI}(\Gamma, X, S)^k = \sum_{\substack{i \in \Gamma \\ k \in M}} x_{i,j,k} a_i s_{j,k}^3 \boldsymbol{\zeta e_k} \tag{8.12}$$

The expression for Interval Thermal Utilization is the same. It is however, a function of speed assignment such that:

$$\Upsilon(\Gamma, X, S)_k^j = \frac{\mathrm{ITI}(\Gamma, X, S)_k^j}{(E_j - E_{j-1})\Delta_k} \tag{8.13}$$

The constraints in Equations 8.6, 8.7, 8.9a, 8.9b and 8.11 are changed to account for the interval speeds. In these equations, $\frac{C_i}{s_i}$ is replaced with C_i and $x_{i,j,k}$ is replaced with $x_{i,j,k} \cdot s_{j,k}$. These expressions represent computation time of task i at unit speed and the amount of computation done for task i during interval j on core k at unit speed respectively. With this alteration in objective function and constraints, the optimization formulation will find the interval execution assignments (X) and interval speed assignments (S) such that task execution constraints are met and temperature is minimized. As with non-DVFS solutions, GPS is used to schedule tasks within each interval.

8.3 Results

In this section, we evaluate the benefit of global scheduling strategy in Section 8.1 for implicit deadline periodic tasks. For the results presented here, we assume zero preemption and task migration overhead. This is to evaluate whether we can reach the lower bound on temperature presented in Section 7.2.1. For evaluating the potential of global scheduling we relax constraint in

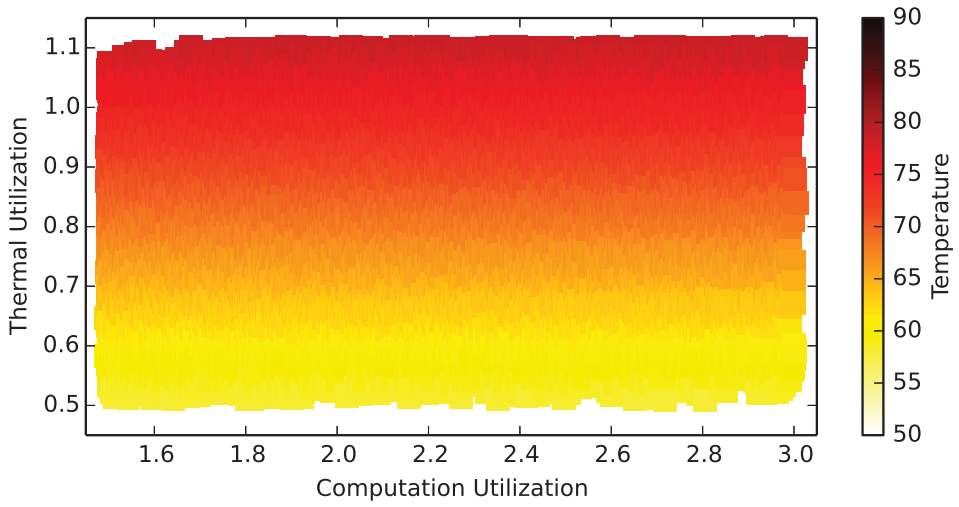


Figure 8.4: Maximum temperature given by global scheduling scheme.

Equation 8.4 and replace it with the following constraint:

$$\sum_{k \in M} x_{i,j,k} \leq E_j - E_{j-1} \qquad \forall i \in \Gamma, j \in I \tag{8.14}$$

Removal of constraint in Equation 8.4 allows tasks to migrate across cores within an interval. However, inclusion of constraint in Equation 8.14 prevents a task from executing more than the interval length (avoiding concurrent execution). Figure 8.4 shows the simulation results for all periodic tasksets simulated in Section 7.4 for this *Global Scheduling Formulation*. In Figure 8.4, color represents the maximum processor temperature across all cores and intervals for all periodic tasksets. We achieve the temperature lower bound defined in Section 7.2.1 in all of the simulated cases. The difference from temperature lower bound is illustrated in Figure 8.5 which plots the histogram of difference in maximum temperature given by global scheduling scheme from the temperature lower bound. The results of TrUMPS are reproduced here for providing reference. As

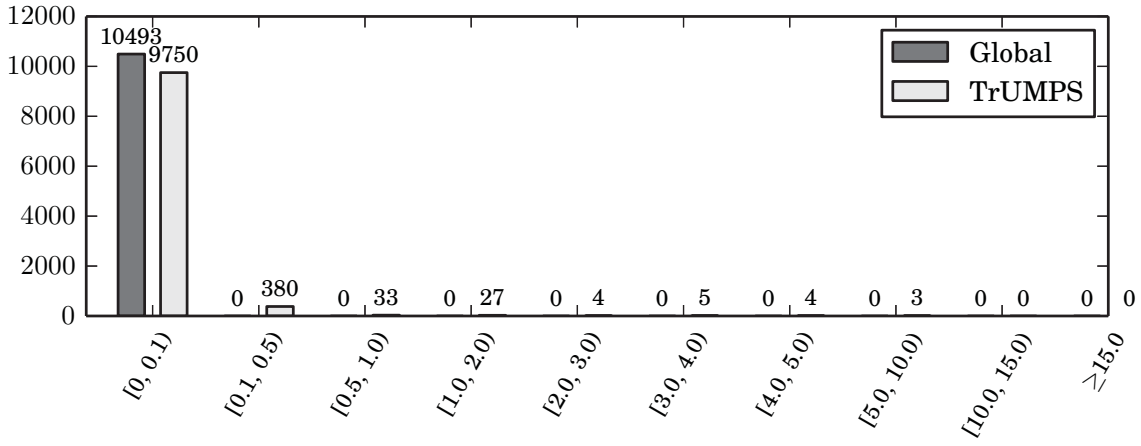


Figure 8.5: Difference in maximum temperature given by Global, TrUMPS and the lower bound on temperature.

shown in figure 8.5, the global scheduling strategy presented in this section achieves lower bound on temperature in all of the simulated cases.

This empirical evaluation leads to the conjecture that the higher maximum temperature for the TrUMPS scheme, compared to the lower bound on maximum temperature, is entirely due to the partitioned nature of TrUMPS solution. Therefore, a thermal utilization of ≤ 1 is both a necessary and sufficient condition for thermal feasibility of implicit deadline periodic tasks on multi-core. However, the sufficiency of this condition remains to be proven.

Chapter 9

Conclusion and Future Work

In this dissertation, we present optimal thermal constrained scheduling solutions for real-time tasks. The task models covered are periodic and aperiodic. The system models covered are uni-core and homogeneous/heterogeneous multi-core. We present several important theoretical results pertinent to the thermal feasibility of these tasks. Several extensions of this work are possible. We will conclude this dissertation by going over these directions for future research. Future research directions are ordered chronologically from short-term to long-term goals. Short-term goals are augmentations to the research covered in this dissertation while long-term goals are new research directions.

9.1 Dynamic reclaiming strategies

The worst-case estimates of computation time and power consumption are usually pessimistic. The average case may be lower than the worst-case; and the additional margin is allowed to guarantee that the system does not fail under worst-case conditions. Therefore, it is possible to exploit run-time behaviour of tasks to reduce temperature and energy consumption further. Work of this

nature has been conducted for energy minimization [49]. However, there is potential for constructing dynamic reclaiming strategies which minimize temperature as well as energy consumption by exploiting run-time behaviour of tasks.

9.2 Dynamic Priority Exchange

Dynamic Priority Exchange (DPM) schemes are a means of reducing response time of aperiodic tasks. They function by executing periodic tasks at higher rate when aperiodic tasks are not present in the system. When an aperiodic task arrives, the priorities of periodic and aperiodic tasks are *exchanged* and aperiodic task executes at a higher rate. This reduces the response time for aperiodic task. T2BS scheduling strategy presented in Chapter 5 can be extended by incorporating DPM. However, the priorities will have to be exchanged such that there are no thermal violations.

9.3 Improved Thermal Models

The thermal model assumed in this work considers a core to be a single thermal element. Accuracy of the thermal model can be improved if the modelling is done at finer granularity (e.g. by consideration each functional unit within a core as a separate thermal element). This extension will reduce temperature estimation error and enable improved applicability of the proposed concepts in real-systems. Furthermore, there are some factors that make a real thermal system non-linear. These include: (a) exponential dependence of leakage power on temperature, (b) different fan speeds at different temperatures, (c) dependence of thermal resistance and capacitance on system temperature. In this work, worst-case for all of these factors is assumed to ensure that no thermal

violations occur. Therefore, the schedulability bounds in this work can be improved if variation in these factors is incorporated in the thermal model.

9.4 Sufficient Conditions for WF2Q

We prove in Chapters 4 and 7 that a fluid GPS model of execution is optimal in-terms of minimizing maximum temperature. Since a real system is non-fluid, WF2Q is used to emulate GPS on a real system. For better applicability of the proposed concepts, computational and thermal overheads for WF2Q need to be studied. WF2Q computation overhead, comes from the preemption/context-switch overhead on a given system. High number of preemptions will increase the worst-case computation time of tasks. Similarly, the temperature of WF2Q fluctuates around the ideal GPS temperature. Therefore, the maximum temperature reached by WF2Q is higher compared to the maximum GPS temperature. It is valuable to bound these overheads so that sufficient feasibility conditions for WF2Q can be proposed.

9.5 Non-Preemptive Execution Model

The focus of this dissertation is on preemptive scheduling strategies. However, some critical applications are non-preemptive, either because of high preemption overhead or because of the critical nature of the tasks being performed by the application. We have done work in the non-preemtive domain. Specifically, [24] covers a scheme for non-preemptively scheduling aperiodic tasks along with a fixed periodic schedule, in a thermally constrained multi-core system.

The scheduling strategies proposed in this research rely on the preemptable nature of tasks and, therefore, cannot be applied when the application is non-preemptable. However, the necessary conditions for thermal feasibility presented in Chapter 3 and 7 apply to both preemptive and non-preemptive domains. Therefore, this research may be valuable in devising thermal-aware scheduling solutions for non-preemptive tasks. However, any solution in this domain will likely be a heuristic. This is because non-preemptive scheduling of implicit deadline periodic tasks on uni-core system is NP-Hard [60].

9.6 Sporadic Task Scheduling

Sporadic tasks are characterized by worst-case computation time, deadline and minimum-inter-arrival time. Minimum inter-arrival time is the minimum time interval between arrivals of two instances of a given sporadic tasks. Therefore, like periodic tasks, sporadic tasks also form an infinite sequence. However,unlike periodic tasks, their arrival pattern is non-deterministic. Significant work has been on on computation feasibility of sporadic tasks [61]. However, their thermal feasibility is a subject on ongoing research. The proposed concepts of Thermal Impact/Utilization and using GPS/WF2Q as scheduling strategies isolate/abstract the related constraints of computational feasibility and thermal feasibility. Therefore, using the framework presented in this dissertation for sporadic task scheduling may yield strong theoretical results pertinent to their thermal feasibility.

9.7 Mobile Computing

Mobile processing platforms have very strict thermal constraints since user will be inconvenienced if his/her cell phone is prone to overheating. Furthermore, mobile processing platforms

have very restrictive cooling solutions since fans and other active cooling solutions cannot be employed. Due to these factors, reducing temperature by means of scheduling has significant potential in mobile processing domain. Furthermore, several applications in this domain have soft real-time constraints. Examples of these applications include gaming, video conferencing and multi-media. Therefore, several of the schemes/concepts proposed in this dissertation have high applicability in this domain and may lead to improved user experience and battery life. Partitioned multi-core scheduling strategies and ability to model and exploit computational heterogeneity are especially useful in this domain.

9.8 Data-Center Computing

The concepts proposed in this dissertation may also be applied to large scale data-centres for reducing temperature and cooling costs. Data-centres are large structures that may house thousands of computing servers. These structures generate a high amount of heat and, therefore, need expensive cooling solutions to keep temperature within safe operating range. Although the thermal model for a data-center is different compared to the processor thermal model used in this work, the proposed concepts may be extended to incorporate the data-center thermal model. The schemes proposed in this dissertation may then be used to distribute thermal load across the data-center such that spacial variation in maximum temperature is minimized. This has potential of reducing maximum temperature across data-center; leading to lower cooling costs and making the data-center more *Green*.

DISCARD THIS PAGE

Appendix A: Proof of Theorem 6.3

Proposition A.1. *Let $\Gamma_X \subset \Gamma$ be a set of all periodic tasks for which the speed decision has been made with speeds equal to $s_{i,X}$ $\forall i \in \Gamma_x$. Then, the energy consumption of the remaining periodic tasks in set $\Gamma_m athrm UX = \Gamma - \Gamma_X$ is minimized if can be executed at speed*

$$s_i = \begin{cases} s_{i,x} & \forall i \in \Gamma_X \\ a_i^{-1/3} G(\Gamma_{\mathrm{UX}}) \Big/ \left(1 - \sum_{j \in \Gamma_X} \frac{C_j}{T_j s_j}\right) & \forall i \in \Gamma_{\mathrm{UX}} \end{cases} \tag{A.1}$$

Proof: Given that the speed decision for all periodic tasks in Γ_X has been made and their speed cannot be changed, the computation utilization of all periodic tasks in Γ_X is equal to $\sum_{j \in \Gamma_X} \frac{C_j}{T_j s_j}$. Therefore, the maximum computation utilization of periodic tasks in Γ_{UX} is $\left(1 - \sum_{j \in \Gamma_X} \frac{C_j}{T_j s_j}\right)$. Therefore, by Theorem 6.2, the energy consumption of periodic tasks in Γ_{UX} is minimized if speed assignments are done as in Equation A.1. ■

However, if we consider speed constraints, we may not be able to execute all periodic tasks at their optimal target speed. To solve the problem with speed constraints, we first enforce the maximum speed constraint without enforcing the minimum speed constraint.

Proposition A.2. *Let $Z_X = (i : s_i^* \geq s_{\max})$ where s_i^* is as defined in Equation 6.1. Then, for the special case where $s_{\min} = 0$, the thermal utilization of the periodic taskset is minimized only if: $s_i = s_{\max}$ $\forall i \in Z_X$*

Proof: Suppose that we start solving the optimization problem by setting the speed of a single periodic task $\tau \in Z_X$ to $s_{\max}$. Then by proposition A.1, we know that the optimal solution for

remaining periodic tasks would try to do speed assignment such that:

$$\bar{s_i} = a_i^{-1/3} G(\Gamma \backslash \tau) \Big/ \left(1 - \frac{C_\tau}{T_\tau s_{\max}}\right) \qquad \forall i \in (\Gamma \backslash \tau)$$

Since periodic task τ Since periodic task was executed at a speed $s_\tau = s_{\max} \leq s_\tau^*$, $\bar{s_i} \geq s_i^* \; \forall i \in (\Gamma \backslash \tau)$. Therefore, if we now compute set $\widehat{Z_X} = (i : s_i \geq \bar{} s_{\max})$, we know that $\widehat{Z_X} \supseteq Z_X$. Therefore, the order in which the speeds of periodic tasks in M_X are assigned does not influence the solution. Furthermore,due to the convexity of the objective function $\frac{\zeta}{\Delta} \cdot \frac{a_\tau C_\tau s_\tau^2}{T_\tau}$ in terms of speed s, if $s_\tau^* \geq s_{\max}$ for a given periodic task τ, the optimal speed assignment would set the speed of periodic task τ to $s_{\max}$. ■

In algorithm 1, we first set the speeds of tasks in M_X to $s_{\max} = 1$. After this assignment, M_X is recomputed tasks in M_X are assigned speed $s_{\max}$. This process is repeated until the recomputation of M_X yields an empty set. By Proposition A.2, the optimal solution would assign all tasks in M_X speed of $s_{\max} = 1$ for all iterations of the while loop. After the completion of while loop, the remaining tasks are assigned optimal speed based on Proposition A.1. Therefore, the solution given by Algorithm 1 is optimal if $s\min = 0$.

Appendix B: Proof of Theorem 6.4

Proposition B.1. *Let $Z_X = (i : s_i^* \geq s_{\max})$ and $Z_N = (i : s_i^* \leq s_{\min})$ where s_i^* is as defined in Equation 6.1. Then, for the special case where $M_X = \emptyset$, the thermal utilization of the periodic taskset is minimized only if: $s_i = s_{\min}\ \forall i \in Z_N$*

Proof: Suppose that we start solving the optimization problem by setting the speed of a single periodic task $\tau \in Z_N$ to $s_{\min}$. Then by proposition A.1, we know that the optimal solution for remaining periodic tasks would try to do speed assignment such that:

$$\bar{s_i} = a_i^{-1/3} G(\Gamma \backslash \tau) \Bigg/ \left(1 - \frac{C_\tau}{T_\tau s_{\min}}\right) \qquad \forall i \in (\Gamma \backslash \tau)$$

Since periodic task τ was executed at a speed $s_\tau = s_{\min} \geq s_\tau^*$, $\bar{s_i} \leq s_i^*\ \forall i \in (\Gamma \backslash \tau)$. Therefore, if we now compute set $\widehat{Z_N} = (i : \bar{s_i} \leq s_{\min})$, we know that $\widehat{Z_N} \supseteq Z_N$. Therefore, the order in which the speeds of periodic tasks in M_N are assigned does not influence the solution. Furthermore,due to the convexity of the objective function $\frac{\zeta}{\Delta} \cdot \frac{a_\tau C_\tau s_\tau^2}{T_\tau}$ in terms of speed s, if $s_\tau^* \leq s_{\min}$ for a given periodic task τ, the optimal speed assignment would set the speed of periodic task τ to $s_{\min}$. ■

Proposition B.2. *Let $Z_X = (i : s_i^* \geq s_{\max})$ and $Z_N = (i : s_i^* \leq s_{\min})$ where s_i^* is as defined in Equation 6.1. Then, for the special case where $M_N = \emptyset$, the thermal utilization of the periodic taskset is minimized only if: $s_i = s_{\max}\ \forall i \in Z_X$*

Proof: Similar to Proposition B.1. ■

Theorem 6.4 follows directly from Proposition B.1 and Proposition B.2. Furthermore, by Proposition B.1, $s_{\min} \leq s_i \leq s_{\max}\ \forall i \in \Gamma_{\mathrm{UX}}$ in line 16 of SeCTUM. Therefore, Proposition A.1 may be used to assign speeds to all tasks in Γ_{UX}.

LIST OF REFERENCES

[1] C. Liu and J. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM*, vol. 20, no. 1, pp. 46–61, 1973.

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

[3] "This car runs on code." http://spectrum.ieee.org/green-tech/advanced-cars/this-car-runs-on-code.

[4] "Internationl technology roadmap for semiconductors," 2012.

[5] R. Vishwanath, V. Wakharkar, A. Watwe, and V. Lebonheur, "Thermal performance challenges from silicon to systems," *Intel Technology Journal*, vol. 4, no. 3, 2000.

[6] A. Vassighi and M. Sachdev, "Thermal runaway in integrated circuits," *IEEE Transactions on Device and Materials Reliability*, vol. 6, no. 2, pp. 300–305, 2006.

[7] J.-J. Chen, S. Wang, and L. Thiele, "Proactive speed scheduling for real-time tasks under thermal constraints," in *Proceedings of the Real-Time and Embedded Technology and Applications Symposium*, pp. 141–150, April 2009.

[8] Y. Fu, N. Kottenstette, Y. Chen, C. Lu, X. Koutsoukos, and H. Wang, "Feedback thermal control for real-time systems," in *Proceedings of the Real-Time and Embedded Technology and Applications Symposium*, pp. 111–120, IEEE, Apr. 2010.

[9] G. Quan and Y. Zhang, "Leakage Aware Feasibility Analysis for Temperature-Constrained Hard Real-Time Periodic Tasks," in *Proceedings of the Euromicro Conference on Real-time Systems*, pp. 207–216, IEEE Computer Society, July 2009.

[10] S. Wang and R. Bettati, "Reactive speed control in temperature-constrained real-time systems," in *Proceedings of the Euromicro Conference on Real-time Systems*, pp. 160–170, July 2006.

[11] R. Ahmed, P. Ramanathan, and K. Saluja, "On thermal utilization of periodic task sets in uniprocessor systems," in *Proceedings of International Conference on Embedded and Real-Time Computing Systems and Applications*, IEEE, 2013.

[12] S. Wang, Bettati, and Riccardo, "Delay analysis in temperature-constrained hard real-time systems with general task arrivals," in *Proceedings of the Real-time Systems Symposium*, pp. 323–334, Dec. 2006.

[13] P. Kumar and L. Thiele, "Timing analysis on a processor with temperature-controlled speed scaling," in *Proceedings of the Real-Time and Embedded Technology and Applications Symposium*, pp. 77–86, IEEE, Apr. 2012.

[14] Y. Ahn and R. Bettati, "Transient overclocking for aperiodic task execution in hard real-time systems," in *Proceedings of the Euromicro Conference on Real-time Systems*, (Los Alamitos, CA, USA), pp. 102–111, IEEE Computer Society, July 2008.

[15] P. Hettiarachchi, N. Fisher, M. Ahmed, L. Y. Wang, S. Wang, and W. Shi, "The design and analysis of thermal-resilient hard-real-time systems," in *Proceedings of the Real-Time and Embedded Technology and Applications Symposium*, Apr. 2012.

[16] P. Kumar and L. Thiele, "End-to-end delay minimization in thermally constrained distributed systems," in *Proceedings of the Euromicro Conference on Real-time Systems*, pp. 81–91, IEEE, July 2011.

[17] L. Schor, H. Yang, I. Bacivarov, and L. Thiele, "Worst-case temperature analysis for different resource models," *IET Circuits, Devices & Systems*, vol. 6, no. 5, pp. 297–307, 2012.

[18] L. Schor, I. Bacivarov, H. Yang, and L. Thiele, "Worst-case temperature guarantees for real-time applications on multi-core systems," in *Proceedings of the Real-Time and Embedded Technology and Applications Symposium*, pp. 87–96, IEEE, Apr. 2012.

[19] J. Chen, C. Hung, and T. Kuo, "On the minimization fo the instantaneous temperature for periodic real-time tasks," in *Proceedings of the Real-Time and Embedded Technology and Applications Symposium*, pp. 236–248, 2007.

[20] B. Yun, K. Shin, and S. Wang, "Predicting thermal behavior for temperature management in time-critical multicore systems," in *Proceedings of the Real-Time and Embedded Technology and Applications Symposium*, pp. 185–194, Apr. 2013.

[21] L. Schor, H. Yang, I. Bacivarov, and L. Thiele, "Thermal-aware task assignment for real-time applications on multi-core systems," in *Formal Methods for Components and Objects*, pp. 294–313, Springer, 2013.

[22] N. Fisher, J.-J. Chen, S. Wang, and L. Thiele, "Thermal-aware global real-time scheduling on multicore systems," in *Proceedings of the Real-Time and Embedded Technology and Applications Symposium*, pp. 131–140, April 2009.

[23] P. Hettiarachchi, N. Fisher, and L. Wang, "Achieving thermal-resiliency for multicore hard-real-time systems," in *Proceedings of the Euromicro Conference on Real-time Systems*, pp. 37–46, July 2013.

[24] R. Ahmed, P. Ramanathan, K. K. Saluja, and C. Yao, "Scheduling aperiodic tasks in next generation embedded real-time systems," *Proceedings of International Conference on VLSI Design*, vol. 0, pp. 25–30, 2013.

[25] R. I. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems," *ACM Computing Surveys*, vol. 43, pp. 35:1–35:44, Oct. 2011.

[26] A. Pareh and R. Gallagher, "A generalized processor sharing approach to flow control in integrated services network the single node case," *ACM/IEEE Transactions on Networking*, vol. 1, pp. 344–357, June 1993.

[27] M. Spuri and G. Buttazzo, "Efficient aperiodic service under earliest deadline scheduling," in *Proceedings of the Real-time Systems Symposium*, pp. 2–11, Dec 1994.

[28] G. Quan, Y. Zhang, W. Wiles, and P. Pei, "Guaranteed scheduling for repetitive hard real-time tasks under the maximal temperature constraint," in *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis*, pp. 267–272, ACM, 2008.

[29] V. Chaturvedi, H. Huang, and G. Quan, "Leakage aware scheduling on maximum temperature minimization for periodic hard real-time systems," in *Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on*, pp. 1802–1809, IEEE, 2010.

[30] T. Chantem, X. S. Hu, and R. P. Dick, "Temperature-aware scheduling and assignment for hard real-time applications on mpsocs," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, pp. 1884 –1897, October 2011.

[31] C.-M. Hung, J.-J. Chen, and T.-W. Kuo, "Energy-efficient real-time task scheduling for a dvs system with a non-dvs processing element," in *Proceedings of the Real-time Systems Symposium*, (Los Alamitos, CA, USA), pp. 303–312, IEEE Computer Society, Dec. 2006.

[32] H. Liu, Z. Shao, M. Wang, and P. Chen, "Overhead-aware system-level joint energy and performance optimization for streaming applications on multiprocessor systems-on-chip," in *Proceedings of the Euromicro Conference on Real-time Systems*, pp. 92–101, July 2008.

[33] L. Thiele, S. Chakraborty, and M. Naedele, "Real-time calculus for scheduling hard real-time systems," in *Proceedings of International Symposium on Circuits and Systems*, vol. 4, pp. 101–104, IEEE, 2000.

[34] E. Kursun, C. yong Cher, A. Buyuktosunoglu, and P. Bose, "Investigating the effects of task scheduling on thermal behavior," in *In Third Workshop on Temperature-Aware Computer Systems (TACS06*, 2006.

[35] Y.-K. Cheng, P. Raha, C.-C. Teng, E. Rosenbaum, and S.-M. Kang, "ILLIADS-T: An electrothermal timing simulator for temperature-sensitive reliability diagnosis of CMOS VLSI chips," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 17, no. 8, pp. 668–681, 1998.

[36] D. Chen, E. Li, E. Rosenbaum, and S. Kang, "Interconnect thermal modeling for accurate simulation of circuit timing and reliability," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, no. 2, pp. 197–205, 2000.

[37] T. Wang and C. Chen, "3-D thermal-ADI: A linear-time chip level transient thermal simulator," *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, vol. 21, no. 12, pp. 1434–1445, 2002.

[38] Y. Zhan and S. Sapatnekar, "High-efficiency Green function-based thermal simulation algorithms," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 9, pp. 1661–1675, 2007.

[39] W. Huang, S. Ghosh, S. Velusamy, K. Sankaranarayanan, K. Skadron, and M. Stan, "HotSpot: A compact thermal modeling methodology for early-stage VLSI design," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 14, no. 5, p. 501, 2006.

[40] R. Ahmed, P. Ramanathan, and K. K. Saluja, "Temperature minimization using power redistribution in embedded systems," in *Proceedings of International Conference on VLSI Design*, Jan. 2014.

[41] Y. Liu, R. Dick, L. Shang, and H. Yang, "Accurate temperature-dependent integrated circuit leakage power estimation is easy," in *Design, Automation Test in Europe*, pp. 1–6, 2007.

[42] R. Ahmed, P. Ramanathan, and K. K. Saluja, "Necessary and sufcient conditions for thermal schedulability of periodic real-time tasks," in *Proceedings of the Euromicro Conference on Real-time Systems*, July 2014.

[43] J. C. R. Bennett and H. Zhang, "Wf2q: Worst-case fair weighted fair queueing," in *Proceedings of INFOCOM*, vol. 1, pp. 120–128, 1996.

[44] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel, "Proportionate progress: A notion of fairness in resource allocation," *Algorithmica*, vol. 15, pp. 600–625, 1996.

[45] E. Bini and G. C. Buttazzo, "Measuring the performance of schedulability tests," *Real-Time Systems*, vol. 30, no. 1-2, pp. 129–154, 2005.

[46] R. Ahmed, , A. Bansal, B. Kakunoori, P. Ramanathan, and K. K. Saluja, "Thermal extension of the total bandwidth server," in *Proceedings of International Conference on VLSI Design*, 2015.

[47] K. S. Trivedi, *Probability and Statistics with Reliability, Queuing and Computer Science Applications*. John Wiley and Sons Ltd., 2nd edition ed., 2002.

[48] R. Ahmed, P. Ramanathan, and K. K. Saluja, "Necessary and sufficient conditions for thermal schedulability of periodic real-time tasks," Submitted to ACM TECS.

[49] H. Aydin, R. Melhem, D. Mosse, and P. Mejia-Alvarez, "Dynamic and aggressive scheduling techniques for power-aware real-time systems," in *Proceedings of the Real-time Systems Symposium*, pp. 95–105, Dec 2001.

[50] H. Aydin, V. Devadas, and D. Zhu, "System-level energy management for periodic real-time tasks," in *Proceedings of the Real-time Systems Symposium*, (Washington, DC, USA), pp. 313–322, Dec. 2006.

[51] S. K. Dhall and C. L. Liu, "On a real-time scheduling problem," *Operations Research*, vol. 26, no. 1, pp. 127–140, 1978.

[52] A. Srinivasan and S. Baruah, "Deadline-based scheduling of periodic task systems on multiprocessors," *Information Processing Letters*, vol. 84, no. 2, pp. 93–98, 2002.

[53] Y. Oh and S. H. Son, "Allocating fixed-priority periodic tasks on multiprocessor systems," *Real-Time Systems*, vol. 9, no. 3, pp. 207–239, 1995.

[54] T. Rothvoss, "On the computational complexity of periodic scheduling," 2009.

[55] A. Burchard, J. Liebeherr, Y. Oh, and S. H. Son, "New strategies for assigning real-time tasks to multiprocessor systems," *Computers, IEEE Transactions on*, vol. 44, no. 12, pp. 1429–1442, 1995.

[56] D. S. Johnson and M. R. Garey, "Computers and intractability: A guide to the theory of np-completeness," *Freeman&Co, San Francisco*, p. 32, 1979.

[57] B. Andersson, S. Baruah, and J. Jonsson, "Static-priority scheduling on multiprocessors," in *Proceedings of the Real-time Systems Symposium*, pp. 193–202, IEEE, 2001.

[58] P. Emberson, R. Stafford, and R. I. Davis, "Techniques for the synthesis of multiprocessor tasksets," in *1st International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, pp. 6–11, 2010.

[59] R. I. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems," *ACM Comput. Surv.*, vol. 43, Oct. 2011.

[60] K. Jeffay, D. Stanat, and C. Martel, "On non-preemptive scheduling of periodic and sporadic tasks," in *Proceedings of the Real-time Systems Symposium*, pp. 129–139, Dec. 2002.

[61] S. Baruah and G. Lipari, "A multiprocessor implementation of the Total Bandwidth Server," in *Proceedings of the International Conference on Parallel and Distributed Systems*, 2004.