TECHNISCHE UNIVERSITÄT MÜNCHEN

Fakultät für Informatik
Lehrstuhl für Echtzeitsysteme und Robotik

System Level Periodic Thermal Management for Hard Real-Time Systems

Long Cheng

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzende/-r: ……Prof. Dr. Uwe Baumgarten…….

Prüfende/-r der Dissertation:

1. Prof. Dr.-Ing. habil. Alois Knoll
2. Prof. Dr. Kai Huang, Sun Yat-Sen University, China

Die Dissertation wurde am 20.06.2017 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 15.11.2017 angenommen.

Abstract

As the VLSI technology is scaling to deep sub-micron domain, more and more transistors are integrated into microprocessors. As a consequence, the power density is rapidly increased, resulting in the rising temperature on microprocessors. High temperature poses serious challenges to designers of hard real-time systems since it severely hampers the reliability and performance of the system. Temperature has become an emerging issue of high importance for real-time systems. Therefore, developing thermal managements is a fundamental aspect in the design of real-time systems. The role of a real-time thermal management is twofold. On one hand, it should correctly and accurately model the timing characteristics and non-determinisms of real-time tasks so that one can tightly bound the demanded system resources. On the other hand, it must perform thermal optimization actions, e.g., reducing the peak temperature, minimizing thermal gradients, etc., under the aforementioned hard real-time constraints.

In this thesis, we focus on developing the system level dynamic thermal management technique, i.e., periodic thermal management, for real-time systems with single and multi-core architectures. To handle general event arrivals with non-determinisms, the theory of real-time calculus is adopted as the task model. The main contributions of this thesis can be listed as the following:

- An offline thermal management, termed as periodic thermal management, is presented for single core real-time systems.
- Periodic thermal management is extended to pipelined multi-core systems by reversely utilizing the pay-burst-only-once principle.
- An online adaptive periodic thermal management that can capture

the variations in event arrivals and executions is proposed.

- A thermal framework which can evaluate various thermal managements in a fast manner is presented.

Zusammenfassung

Aufgrund der Entwicklung von VLSI hin zu einer deep sub-micron Domäne, werden immer mehr Transistoren auf Mikroprozessoren integriert. Als Folge davon nimmt die Leistungsdichte immer mehr zu, was zu erhöhten Temperaturen dieser Prozessoren führt. Hohe Temperaturen stellen Entwickler von Echtzeitsystemen vor große Herausforderungen, da diese die Zuverlässigkeit und Leistung dieser Systeme beeinträchtigt. Temperatur entwickelt sich daher zunehmend zu einem Problem von hoher Bedeutung für Echtzeitsysteme. Aufgrund dessen ist die Entwicklung von Thermomanagement ein fundamentaler Aspekt beim Design von Echtzeitsystemen. Ein Thermomanagementsystem hat zwei Aufgaben. Zum einen soll es die Timing-Eigenschaften und den Nichtdeterminismus von Echtzeitaufgaben korrekt modellieren, sodass man möglichst gute Vorhersagen bezüglich der benötigten Ressourcen des Systems treffen kann. Zum anderen muss es thermale Optimierungsaktionen unter den zuvor genannten harten Echtzeitbeschränkungen durchführen, wie zum Beispiel die Reduzierung der Höchsttemperaturen, die Minimierung des Temperaturgradients, usw. Der Fokus dieser Arbeit liegt auf der Entwicklung einer auf Systemlevel dynamischen Thermomanagementmethode, d.h. einem periodischen Thermomanagementsystem für Echtzeitsysteme mit Ein- oder Mehrkernarchitekturen. Um eintreffende, nichtdeterministische Ereignisse handhaben zu können, wird auf die Theorie von Echtzeit-Differentialrechnung zurückgegriffen. Die Hauptanteile dieser Arbeit können wie folgt aufgelistet werden:

- ein offline Thermomanagementsystem, bezeichnet als periodisches Thermomanagementsystem wird für Einkern-Echtzeitsysteme vorgestellt.
- das periodische Thermomanagementsystem wird erweitert, um Mehrkernsysteme zu unterstützen, indem das "pay-burst-only-once"-

Prinzip angewandt wird.

- ein online anpassbares periodisches Thermomanagementsystem, welches die Variation von eintreffenden Ereignissen einfangen kann wird vorgeschlagen.
- ein Thermo-Framework, welches verschiedene Thermomanagementsysteme schnell evaluieren kann wird vorgestellt.

Acknowledgements

First of all, I would like to express my sincere gratitude to Prof. Dr. habil. Alois C. Knoll for offering the opportunity for studying in Technical University of Munich and constantly patiently supervising my research. Without his support, this thesis would have not been possible.

I would like to thank Prof. Dr. Kai Huang for being my coexaminer in this thesis and providing me valuable suggestions about my research in my Ph.D. life.

I would also like to thank: Assoc. Prof. Dr. Gang Chen, Dr. Guang Chen and Dr. Biao Hu for the fruitful research cooperation; Zhenshan Bing for the nice collaboration in the snake robot project; Mingchuan Zhou for the exciting cooperation in the research of thermal management; Xiebing Wang and Zhuangyi Jiang for their supports and proofreading my thesis; Dipl. Inf. Brian Jensen and Alexander Perzylo for their kind help in the beginning of my Ph.D. life. Furthermore, I would like to thank all my former and current colleagues of the whole Robotics and Embedded System chair for their company and support.

My sincere thanks also goes to my friends: Xiang Lu, Zhu Liu, Zhen Yao, Di Xu and Yao Xiao for all the times we had in the last four years.

Finally, my dearest thanks go to my family for their love and support throughout all these years of my Ph.D. study.

The work presented in this thesis was supported by the China Scholarship Council (grant number: 201306120019). This support is gratefully acknowledged.

To my wife, Shanshan.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

As predicted by the Moore's law, more and more transistors have been integrated in modern microprocessors. Hence the power density is rapidly increasing, which consequently raises the temperature of microprocessors. High temperature seriously hampers the reliability and performance of microprocessors. Real-time systems, in which tasks must finish before their deadlines, have additional requirements with respect to reliability and performance stability. Therefore, high temperature poses challenges to designers of real-time systems. This thesis presents a set of novel thermal management technologies for real-time systems. In particular, we focus on solutions for optimizing temperature under hard real-time constraints by adopting dynamic power management technology. Section 1.1 introduces the thermal issue of microprocessors. Section 1.2 surveys the state-of-art thermal management technologies. Section 1.3 draws the outline and summaries the contributions of this thesis.

1.1 The Emerging Thermal Issues

Temperature is a fundamental parameter associated with the performance and reliability of electronic equipments [77]. In the past several years, thermal-related issues have become especially important for microprocessor design [54]. In this section, we explain the causes behind the emerging thermal issues in three aspects: the increasing power density (Section 1.1.1), negative effects of high temperature (Section 1.1.2), and thermal management methods (Section 1.1.3).

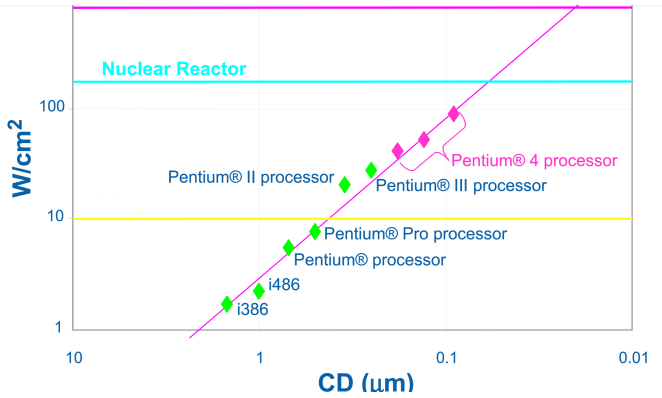


Figure 1.1: A plot of power density against critical dimensions [94]. The logarithmic vertical scale indicates exponential growth of power density.

1.1.1 The Increasing Power Density

Most of the energy consumed by a microprocessor is ultimately dissipated in form of heat because of the resistive behaviour of the processor circuits. Temperature is a measurement of how much heat has been produced and thus directly determined by the power density, which denotes the power consumed per unit area of the chip. The transistors in microprocessors have continued to shrink in size since the very first microprocessor. This scaling has significant impacts on the temperature, which is illustrated below by the relationship between the scaling and power density.

Now, we study this relationship according to the Scaling Theory [35]. The length of the transistor is shrunk by every successive technology generation to a constant fraction of previous length. The fraction can be denoted by a scaling factor s and is typically about $1/\sqrt{2}$ [84]. One can conclude that the area of transistors scales proportional to s^2, i.e., about $1/2$. The power consumption of the transistors can be approximately given by formula CV^2f, where C is the intrinsic capacity, V denotes the supply voltage, and f is the clock frequency. If we consider the same microarchitecture, then the scaling of C is linear to s. Assuming the ideal scaling is applied to V and f, i.e., V scales down and f scale up linearly to s, we have the power dissipation is scaled down by factor s^2, indicating the power density keeps constant. However, in reality, it's impossible to continuously scale the supply voltage by a scalar. The reason is that for a clock frequency f, a minimal supply voltage which is approximately linear to f is required by the processor. This causes the supply voltage is not able to scale further. Therefore, for the past

several decades, the power density of microprocessors increases exponentially every generation [84]. A plot of CPU power density against critical dimensions is displayed in Fig. 1.1.

The exponentially growth of power density is the main driving force of the continuously increasing temperature of modern microprocessors. Now, the questions are (1) What is the influence of high temperature to microprocessors? (2) Do we really need to lower the increasing temperature? Next section discusses both questions.

1.1.2 The Influence of High Temperature

People have put significant efforts into removing the heat from the die surface of modern processors, i.e., developing sophisticated physical devices such as liquid cooling systems. The reason is that high temperature is undesirable for microprocessors due to its negative influence in several aspects such as reliability, stability and performance. Next, we list several microprocessors failure mechanisms that can be affected by temperature [57].

Electro-migration

Electro-migration is a failure mechanism referring to the transport of mass in metals caused by the gradual movement of the ions in a conductor due to the momentum transfer between conducting electrons and diffusing metal atoms (AI, Cu), leading to voids in the metal lines [13]. High temperature increases the mobility of carriers and thus accelerates the rate of Electro-migration, decreasing the Mean Time To Failure of microprocessors [4].

High Temperature Stress Migration

This failure mechanism is not caused by the current flow during electro-migration, but the high temperature induced stress which causes the Al metal lines to open up, resulting in open-circuit failure. This failure usually happens when the metal line width is about or less than 2-3 μm. Since there is a trend towards reduction in Al metallization width, this failure mechanism is non-negligible.

Mechanical stresses induced by differential thermal expansion of materials

Microprocessors are constructed from silicon, metal, plastic encapsulation and epoxy resin used in the construction of a plastic package. These materials have different thermal coefficients of expansion (TCE). The TCE describes how the size of an object changes with a change in temperature. When a microprocessor is subjected to wide-range thermal cycling or shocking, the mismatch in TCEs of different materials bounded together inside the processor leads to mechanical stresses, which could cause the passivation cracks in the device.

Iconic Effect

- Hot Carriers. The term hot carrier here refers to the additional electrons produced when electrons collide with the atoms in the crystal lattice. Because of their high kinetic energy, hot carriers can cause problems in memory devices and logic circuits leading to malfunctioning and failure [31]. This failure mechanism is especially enhanced by high temperature.
- Ionic Contamination. Ionic contaminants are typically flux residues or harmful materials that are picked up or left behind during the process. They contain molecules or atoms that are conductive when in solution which can disassociate into either positively or negatively charged species and increase the overall conductivity of the solution. Their mobility gets higher in the presence of high electric fields and at high temperatures and thus further degrades the reliability of the electronic components and increases the risk of corrosion [92].

In additional to above mechanisms, high temperature can also accelerate other several failure mechanisms such as solder joint failures, bond-wire fatigue, electrical overstress, and PCB stress [57]. For most of these failure mechanisms, the Mean Time To Failure (MTTF) can be empirically described using the well-known Arrhenius Equation given by:

$$MTTF = Ae^{\frac{E_a}{kT}} \tag{1.1}$$

where A is an empirical constant, T denotes the temperature, and E_a is the activation energy of the failure mechanism. Although this equation does not capture all features (thermal cycling, thermal shocking, etc.), it

is a useful expression for first-order estimation. From (1.1), the MTTF decreases exponentially with respect to the temperature, which indicates high temperature significantly reduces the reliability of microprocessors. For example, according to [77], the mission life of a microprocessor is about 2×10^5 hours (22.83 years) at temperature 38°C. However, it drops to 1×10^4 hours (1.14 years) when the temperature is increased to 93°C.

Transistors still consume power even when they are idle or not switching. This kind of power is termed as the leakage power or static power. It is directly influenced by the temperature and grows exponentially as the temperature increases. Moreover, since temperature strongly depends on the power dissipation, there is a circular dependency between them. In extreme cases, this can lead to a self-reinforcing positive feedback loop that cause thermal runaway. Thus, high temperature results in higher leakage power consumption.

High temperature can also affect the performance of a microprocessor. The time parameters, such as frequency, of components like transistors, clock, oscillators, etc., drift due to the effect of temperature [57]. Although the drift in parameters by itself may not lead to a failure, it can cause system malfunctions, instability, etc., which seriously hampers the performance of microprocessors.

In conclusion, high temperature has several negative effects on microprocessors. First, the Mean Time To Failure, i.e., the reliability, can be exponentially reduced by high temperature. Second, higher temperature leads to more leakage power consumption, which, in turn, raises the temperature and may cause thermal runaway in extreme cases. Last but not the least, the performance of the microprocessor like speed and stability can be hampered by high temperature. Therefore, temperature has become a first-class design constraint in microprocessor development akin to performance [84]. Proper thermal management methods are required to control the temperature varies in a certain range. Inadequate thermal control can lead to complete failure, as several recent products have shown [95, 99].

1.1.3 Thermal Management Methods

The traditional way to control temperature of microprocessors is using physical heat-removing systems, such as air cooling devices and liquid cooling systems. It's a significant challenge for mechanical engineers to design heat-removing systems for modern microprocessors with afford-

able cost since the temperature is ever rising while the cost increases exponentially with temperature. For high performance microprocessors, the costs of cooling solutions are rising at $1–3 or more per watt of dissipated power [14, 41], and could reach over 35% of electricity costs [90]. Apart from the disadvantage in cost, physical cooling systems may also require additional space and power to install and run itself, which limit the application in portable and hand-held devices. In other words, traditional physical cooling systems have below limitations.

- cooling package cost increases exponentially with respect to power dissipation.
- need additional space to install.
- may consume additional power to run devices such as fans.

To cope with aforementioned limitations of traditional thermal management methods, alternative technologies that reduce the temperature by putting microprocessors into lower power consumption states have been widely adopted. Such technologies can be generally termed as Dynamic Thermal Management (DTM) techniques [15]. Most DTM technologies can be implemented in system-level with basic hardware supports such as temperature sensors, hardware-timers, etc. DTM technologies can remarkably reduce the expense in terms of packing cost, space.

In summary, temperature has become the first-class design concern for microprocessors due to the ever-increasing temperature and its significant impacts on the reliability, performance and power consumption. The Dynamic Thermal Management technologies are promising approaches to control the temperature due to their advantages in cost, space, etc..

1.2 State of the Art Thermal Managements

In this section, we discuss the state of the art thermal managements for microprocessors with single and multi-core architectures. Firstly, we briefly overview the representative existing works. Then, we summary the special requirements that are not completely fulfilled for hard real-time systems by existing works.

1.2.1 Overview

In this section, we briefly review the state of art thermal managements for microprocessors with single and multi-core architectures. Note that

only a representative subset of related works is discussed due to their vast amount.

A thermal management is developed usually for one or more of the following objectives: (1) minimizing the peak temperature; (2) minimizing the thermal gradients on the microprocessor; (3) maintaining the temperature under certain threshold. To control temperature or thermal gradients, most thermal managements adopt task scheduling and power controlling techniques. Temperature can be influenced by the workload as different workloads utilize different processing components inside the microprocessor, which is the main motivation of thermal managements based on task scheduling. Temperature can also be reduced via power controlling mechanisms. Thermal managements based on power controlling mainly follow two main mechanisms, i.e., Dynamic Voltage Frequency Scaling (DVFS), and Dynamic Power Management (DPM). Now, we categorize existing thermal managements according to the temperature-control mechanism adopted by them.

Task scheduling Thermal-aware task scheduling techniques consider spatial and temporal correlations between cores or functional units through balancing the workloads. Thidapat et al. [21] address the problem of assigning and scheduling tasks on MPSOC (Multiprocessor System-on-Chip). They presented a mixed-integer linear programming (MILP) formulation of the problem and then gave an optimal solution as well as a flexible heuristic framework for the MILP formulation. Due to the thermal analysis difficulties, this approach examines only steady-state temperatures without considering the transient behavior. Cox et al. proposed a fast thermal-aware approach for streaming applications based on a 3D MPSoC model under the throughput constraints in [32]. This approach assumes periodic task model and also does not consider the transient temperature. A task scheduling policy that considers temporal correlations is presented in [108]. This work focuses on choosing the right task to execute while maintaining the temperature under given threshold. No real-time guarantee is provided in this work.

DVFS DVFS techniques adjust the supply voltage or clock frequency of a microprocessor and thus can control the dynamic power dissipation. Since dynamic power dominates the total power consumption of early microprocessors, DVFS has been widely studied by researchers. In [6], the authors address the speed scaling problem and proposed two algorithms, an online one and an offline one, to solve the optimization problem under temperature and deadline thresholds, respectively. The

relationship between leakage power dissipation and temperature, however, is not considered for the simplicity of analysis. In [111], two DVFS algorithms, a pseudo-polynomial one and a fully polynomial time approximation one, are presented to optimally improve the system performance for a set of periodic tasks under given temperature constraints. Jian-Jia Chen et al. proposed two algorithms in [25] to optimize the response time and temperature respectively. Chantem et al. [20] made an observation about maximizing the workload under thermal constraints. The authors demonstrated that while working with proactive scheduling, the scheduler which maximizes the workload under given peak temperature must be a periodic one [2]. Yong and et al. [39] presented a feedback thermal control framework named Real-Time Multicore Thermal Control which dynamically enforces both the desired temperature and the CPU utilization bounds for multicore real-time systems, through DVFS. All aforementioned researches assume simple task models such as periodic task model and cannot handle general event arrivals. More DVFS-based thermal managements can be found in [102, 104, 8, 70, 112].

DPM The leakage power dissipation can be reduced by adopting DPM techniques, which put microprocessors into deeper power saving states by decreasing or even cutting off the supply voltage of some portion of the microprocessor. DPM techniques can also be applied on peripheral devices such as memories, interconnects, etc. Kumar et al. [56] developed a thermally optimal stop-go scheduling called JUst Sufficient Throttling (JUST) to minimize peak temperature within given makespan constraints. This scheduling is designed only for static order tasks and is not applicable for non-deterministic tasks. A framework and mechanisms for thermal stress analysis in real-time systems are proposed in [44] to meet the challenge of determining the real-time guarantees in the presence of unpredictable dynamic environmental conditions. Buyoung [110] addressed the problem of avoiding thermal hotspot on a multi-core chip by employing a runtime thermal aware scheduler (TAS) using job-migration and power-gating techniques. Adopting thermal-aware periodic resources, Masud Ahmed et al. [2] proposed an offline algorithm which minimizes the peak temperature for sporadic tasks scheduled by earliest-deadline first (EDF) while guaranteeing all their deadlines. To simplify the complexity of timing analysis, aforementioned works all assumed simple task models, i.e., either periodic or sporadic task model.

1.2.2 Hard Real-Time System Requirements

In previous section, the state of the art thermal managements are briefly reviewed. While having made significant contributions to this field, most existing thermal managements have just partly solved the challenge of optimizing the temperature of hard real-time systems in system level. Compared to general-purpose systems, real-time systems have additional requirements with respect to timing correctness, reliability and stability. Thermal managements in real-time systems not only need to reduce the temperature , but also should tackle the additional requirements posed by real-time system characteristics. Specifically, the following requirements are not completely met in existing thermal managements.

- providing hard real-time guarantees. The tasks in hard real-time systems have deadline constraints. Every task should complete and produce result before its deadline. Many existing works fail to provide hard real-time guarantees or even do not consider deadline constraints [34, 108, 72, 64, 63, 3, 32, 111, 20, 79, 70, 104, 112, 69].
- handling non-deterministic event arrivals. In reality, event arrivals contain non-determinism such as jitter. Modelling such event arrivals by simple task models under hard real-time constraints may cause the problem of over-estimation and result in high temperature. Thus, thermal managements should be able to properly handle events arrivals with non-determinism. Existing works [38, 100, 32, 45, 44, 110, 39, 102, 2, 20, 103] adopt simple task models such as periodic, or sporadic models, and thus cannot meet this requirement.
- modelling temperature behaviours with high accuracy. To find the correct thermal management scheme, the temperature behaviours should be modelled with high accuracy. The temperature accuracy can be remarkably hampered by the bad-established thermal models and incorrect parameters. Thermal managements [64, 34, 63, 70, 6] do not consider the correlation between leakage power and temperature for simplicity. Moreover, the transient thermal behaviour is also ignored in [21, 32].
- identifying the exact peak temperature quickly. In order to efficiently explore the design space of multi-core architecture real-time systems for optimal thermal management, one should calculate the exact peak temperature quickly. Majority of existing

works [26, 36, 88, 67, 81, 66, 71] adopts thermal simulation toolboxes to find the peak temperature, which is computation costly and slow. There are also several works [100, 81] directly utilize the steady-state temperature as the peak temperature, which could be incorrect due to spatial and temporal thermal fluctuations.

In this thesis, we aim to tackle these challenges by adopting system-level Periodic Thermal Management for hard real-time systems. Periodic Thermal Management periodically switches microprocessor cores to sleep state to reduce the temperature. By fully utilizing such timing feature, we proposed a closed-form solution and two numerical calculating algorithms to quickly determine the peak temperature of single core and multi-core architectures, respectively. Thus, we fulfill the aforementioned last requirement. For the third requirement, based on the well-known Fourier equation and HotSpot model, we construct thermal models with high accuracy where heat flow between different thermal blocks, transient thermal behaviors and the leakage current dependency on temperature are all considered.

The Real-Time Calculus (RTC) theory is adopted in our work to model the event arrivals and system resources. The benefits of using RTC are twofold: first, the concepts of arrival curve is introduced as task model. The arrival curve is an abstract model and can model arbitrary event arrivals containing non-determinism. Second, with the existing results of service curve, constraints on the demanded system resources can be derived to provide hard real-time guarantees. Therefore, the aforementioned first two requirements can be met.

In conclusion, the Periodic Thermal Management presented in this thesis enables hard real-time system designers to quickly find the optimal system resource management scheme which minimizes the peak temperature under deadline constraints for event arrivals with non-determinism.

1.3 Thesis Outline and Contributions

This thesis focuses on how to optimize temperature for both single-core and multi-core architectures hard real-time systems. In particular, we aim to lower the peak temperature for general event arrivals under deadline constraints by adopting static and adaptive DPM techniques. In the following, we summarize the content and individual contributions of every following chapter of this thesis.

1.3.1 Chapter 2: Single Core Thermal Management

In Chapter 2, we present the Periodic Thermal Management (PTM) for single-core real-time systems to optimize the peak temperature. The PTM is a static method and requires negligible run-time computation effort and is suitable for single-core processors having little computing power. The real-time calculus [96] interface is adopted to model general event arrivals and ensure the deadline constraints can be satisfied. A close-form solution of the peak temperature is given as a criterion of the optimal solution. We also present two algorithms which can compute the optimal PTM scheme in different levels of accuracy and speed. Specifically, the contributions of this chapter are:

- Based on the well-known Fourier's law thermal model, a closed-form solution of the peak temperature with respect to the periodic thermal management is developed.
- Two PTM algorithms that can derive periodic on/off schemes with a trade-off between accuracy and efficiency are developed. One offers precise solution by making thorough searches and the other is a fast approximation based on bounded-delay function.
- The effectiveness and efficiency of our algorithms are studied by comparison to two related work [2, 55] in the literature. Single-event streams and multi-event streams scheduled by Earliest Deadline First (EDF) are tested in the case studies.

1.3.2 Chapter 3: Pipelined System Thermal Management

In Chapter 3, we investigate how to apply Periodic Thermal Management on real-time multi-core systems. The processor handles the applications that can be divided into sub-tasks which are executed on the cores concurrently. By reversely using the Pay Burst Only Once principle, we can calculate the aggregate service demand bound instead of the individual bound for each stage to obtain feasible PTM schemes for the cores. In this way, we benefit from the advantages from two domains: On one hand, the burst in the event arrivals is accounted only once and thus leads to a lower peak temperature. On the other hand, the complexity of the problem is significantly reduced, which makes our approach scalable with respect to the number of cores. We also perform a comprehensive analysis on the peak temperature of multi-core processors under PTM, the results of which enable the fast computation of the peak temperature. In summary, the contributions of Chapter 3 are:

- Based on the well-known HotSpot model, a peak temperature representation for a multi-core processor under Periodic Thermal Management (PTM) is given, where the heat flow among cores and the leakage current dependency on temperature (LDT) are considered.
- To overcome the inefficiency produced by the strictly accurate method of calculating the peak temperature, two algorithms with different levels of accuracy and complexity are proposed to offer good approximations of the peak temperature.
- By reversely using the Pay Burst Only Once principle, the optimization problem is transformed into a set of sub-problems. We formulate the sub-problems and solve them by two fast heuristic algorithms corresponding to the two peak temperature methods.
- Based on two real life platforms: a homogeneous ARM multiprocessor and the Intel Single-chip Cloud Computer (SCC), we evaluate the effectiveness and efficiency of our approaches by comparing them with two brutally searching approaches, one with PBOO and one without PBOO.

1.3.3 Chapter 4: Adaptive Periodic Thermal Management

While Chapter 2 and Chapter 3 focus on the analysis of static PTM approaches which search the solution in design phase, in Chapter 4 we propose a novel dynamic thermal optimize method termed as Adaptive Periodic Thermal Management (APTM). Specifically, APTM is an offline and online combined approach. The offline learned thermal properties are adopted in online adaption to optimize the calculated solutions. Two thermal curves, i.e., the `warming curve` and the `cooling curve` are proposed to model the thermal properties of each stage in different scenarios. To effectively exploit the dynamic slacks in event arrivals, the Dynamic Counter technique is adopted to give history-aware event predictions. Moreover, the dynamic state information of the processor are also collected to reflect the real execution of jobs. The following contributions are contained in Chapter 4:

- We present a sufficient condition of guaranteeing deadline constraints of unfinished and future events for pipelined systems under APTM schemes. The condition can be easily utilized to derive APTM schemes that satisfy real-time constraints at adaption instants.

- Several lightweight algorithms are presented to compute APTM schemes in runtime efficiently according to the unique thermal properties of the stages. The obtained APTM schemes can effectively reduce the peak temperature under real-time constraints for the pipelined system with negligible online overheads.
- The effectiveness and efficiency of our proposed approach for reducing temperature are evaluated by comparing it with two existing approaches with two real-life hardware platforms.

1.3.4 Chapter 5: Multi-core Fast Thermal Prototyping Framework

In this chapter, we present a multi-core thermal framework named Multi-core Fast Thermal Prototyping (McFTP). McFTP is designed to be a general framework and can evaluate different thermal management policies on actual hardware platforms in an efficient and reliable manner. It is a re-configurable thermal framework running in the user-space and enables multi-core system designers to validate any resource distribution decision in design phase on the target architecture. McFTP can not only implement a thermal management policy at high-level of abstraction, but also execute real or user-defined task-set. The specific contributions can be summarized as:

- To allow the implementation of customized thermal management policies with minimal effort, an intermediate interface named Configuration Manipulation Interface (CMI) is defined to isolate thermal management policies from the low-level implementations.
- A set of commonly used temperature control mechanisms, including, DVFS, DPM, job scheduling and task migration, is implemented as a library which can be accessed via CMI.
- We implement McFTP on the top of Linux with the API defined in POSIX standard. Comprehensive experiments are conducted to investigate the effectiveness and efficiency of the implementation.

Chapter 2

Single Core Thermal Management

Single core processor is the traditional and classical architecture adopted in real-time systems. For example, the microcontroller architecture has been widely used in the filed of control-dominant field having real-time requirements. It's estimated that more than half of all CPUs sold worldwide are microcontrollers [61]. Compared to that in multi-core architecture, the worst-case execution time of a task in single-core processors is more predictable because there is no interference between cores, which can cause delay spikes as high as 600% in industry benchmarks [87]. This feature makes single core architecture suitable for hard real-time systems, which have additional requirements with respect to reliability, and real-time behaviour [91].

To meet these requirements, real-time system designers need to consider an important factor, the temperature of the processor, which plays a key role in determining the allowable execution speed [2], as aforementioned in Chapter 1. The traditional way to control temperature of the processor, using hardware cooling devices, suffers the cost, energy and space disadvantages. The alternative technologies termed as Dynamic Thermal Management (DTM) have been widely adopted. In Chapter 1, we show that DTM techniques follow two main mechanisms, i.e., Dynamic Voltage Frequency Scaling (DVFS) and Dynamic Power Management (DPM). The DPM technologies are demonstrated to be more effective to optimize the temperature on modern processors due to leakage power dominates the total power consumption of 32 nm or more advanced processors.

The main issue of using DPM technologies to control the temperature is when and how long one should turn the processor to the sleep state [11]. It's obvious that dynamically switching the processor into 'sleep' mode according to the event arrivals and their relative deadlines is an effective way to minimize the peak temperature. However, single-core processors adopted in real-time systems usually has little computation ability. Dynamical switching methods can be hardly implemented in this scenario. Further, the additional computation in online manner also incurs power overhead, which, in turn, elevates the temperature. Therefore, an interesting research topic is designing a DPM technique for single core hard real-time system which can:

1. guarantee all events complete within their deadlines.
2. minimize the peak temperature of the processor
3. introduce little running overhead in terms of time and energy.
4. be easily implemented with basic hardware features.

2.1 Overview

In this chapter, we propose the periodic thermal management (PTM), which holds the aforementioned properties, to optimize the peak temperature for general events arrivals while the deadlines are guaranteed.

The single core processor has two power dissipation modes, 'active' and 'sleep' mode, with different power consumptions. The peak temperature is controlled by periodically switching the processor to 'sleep' mode according to the event stream model and thermal properties of the processor. To meet the deadline constraints, real-time calculus [96] interface is employed to model the non-deterministic event arrivals and service provided by the processor in the time interval domain. Combining event timing model and the relative deadline, a service bound is derived to determine PTM schemes that can provide hard real-time guarantee. The applied PTM scheme is calculated in offline manner and thus requires negligible run-time computation effort, which makes our approach suitable to real-time systems having little computation resource. A closed-form solution of the peak temperature with respect to the periodic thermal management is developed as a criterion of the optimal PTM scheme.

It's worth noting that how long should the processor stay in 'sleep' and 'active' mode, i.e., the switching frequency, needs careful consideration.

On the one hand, the length of 'sleep' time interval should be long enough such that fewer switching operation is performed and thus less switching overhead is incurred. On the other hand, due to real-time constraints, longer 'sleep' interval leads to longer 'active' interval, which cause higher temperature peaks at the end and thus higher temperature. To resolve these concerns, two PTM algorithms that can derive periodic on/off schemes with a trade-off between accuracy and efficiency are developed. One offers precise solution by making thorough searches and the other is a fast approximation based on bounded-delay function.

The rest of this chapter is organized as follows. The related work is introduced in the next section. Section 2.4 presents system models, including hardware model, power model and thermal model, and the problem definition. Section 2.5 derives the closed-form solutions of the peak temperature. The real-time analysis is presented in Section 2.6. Section 2.7 presents our PTM algorithms. Several cases are studied in Section 2.7.3 and Section 2.8 concludes this chapter.

2.2 Related Work

The thermal behaviour of a processor is directly influenced by the power consumption. Thus researchers in previous work on thermal-aware scheduling have followed two main approaches: DVFS and DPM, which have already been widely exploited in power-aware scheduling. In this section, we overview previous work for thermal-aware scheduling that based on DVFS and DPM.

Sushu Zhang et al. [111] proposed two DVFS approaches: a pseudo-polynomial optimal algorithm and a fully polynomial time approximation one. These two approaches can optimally and approximately improve the system performance for a set of periodic tasks under thermal constraints, respectively. Jian-Jia Chen et al. [25] presented two approaches to schedule periodic real-time tasks under DVFS while the response time and temperature constraints are satisfied respectively. Chantem et al. [20] made an observation about maximizing the workload under thermal constraints. The authors demonstrated that while working with proactive scheduling, the scheduler which maximizes the workload under given peak temperature must be a periodic one [2]. According to this observation, a speed schedule was proposed to maximize the workload based on DVFS with discrete speeds and transition overhead under given temperature constraints. S. Wang et al. [102] presented a re-

active speed control algorithm for tasks that have the same period to minimize temperature and performed several schedulability tests. The aforementioned work, however, based on either a simplified workload model, such as periodic tasks, or the processor feature of keeping the 'ideal' speed, which may not be found in recent top-of-the-line microprocessors [2]. The periodic thermal management (PTM) proposed in this chapter can handle general event arrival patterns by adopting real-time calculus [96]. Moreover, lower power state, which is a basic power management feature, can be conveniently utilized to implement PTM.

There are also several researches that utilize DPM to minimize the peak temperature under deadline constraints. Kumar et al. [56] developed a thermally optimal stop-go scheduling called JUst Sufficient Throttling (JUST) to minimize peak temperature within given makespan constraints. This scheduling is designed only for static order tasks and is not applicable for non-deterministic tasks. To address the challenge of determining the real-time guarantees in the presence of unpredictable dynamic environmental conditions, Hettiarachchi and et al. [44] proposed a framework and mechanisms for thermal stress analysis in real-time systems. Adopting thermal-aware periodic resources, Masud Ahmed et al. [2] proposed an offline algorithm which minimizes the peak temperature for sporadic tasks scheduled by earliest-deadline first (EDF) while guaranteeing all their deadlines can be met. The workload models of the aforementioned work are also simplified and lead to pessimistic results, that is, higher peak temperature since they cannot exhibit non-determinism like jitter or burst arrivals of the workload. These shortcomings can also be overcome in PTM since it work with general event arrival patterns, as mentioned above. In [55], a Cool Shaper is studied to minimize the peak temperature by delaying the execution of workload for general events arrivals. It is an online/offline-combined approach, where the parameters of the shaper are offline computed and the workload is runtime orchestrated with the pre-computed shaper. Besides the online monitoring overhead which can result in a higher temperature, determining the parameters of the shaper according to the system specification also requires considerable calculation effort. In this chapter, a closed form of the peak temperature is derived such that our PTM can easily obtain the peak temperature offline instead of simulating the online evolution of the temperature, which saves great quantity of calculation.

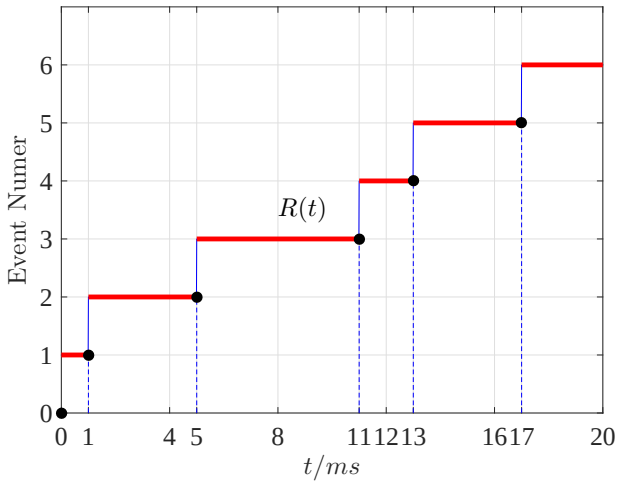


Figure 2.1: An example of the cumulative function $R(t)$.

2.3 Introduction to Real-Time Calculus

This section presents the basic concepts and results of the Real-Time Calculus framework, i.e., the arrival curve, the service curve, and the deadline bound. We also elaborate how to use these results to analyze the timing properties of a system.

2.3.1 Models for Event Stream

Basically, the event streams to a system can be specified by means of the cumulative function $R(t)$, which indicates the number of events that arrive the system in time interval $[0, t]$. The function $R(t)$ is always a wide-sense increasing function. Moreover, It is a discontinuous function since it has a smallest granularity, that is, one event. By convention, we take $R(0) = 0$ in the whole scope of this dissertation unless otherwise specified. An example of $R(t)$ is displayed in Fig. 2.1.

Note that the function $R(t)$ specifies a concrete event stream. To analyze timing properties of the system, an abstract model which provides guarantees to the event streams is required. This is done by using the concept of arrival curve [60], which is defined below.

Definition 2.1 (Arrival Curve) *For an event stream R and a 2-tuple wide-sense increasing functions $\alpha(\Delta) = [\alpha^u(\Delta), \alpha^l(\Delta)]$ defined for $\Delta >= 0$, we say R has $\alpha^u(\Delta)$ and $\alpha^l(\Delta)$ as upper arrival curve and lower arrival curve,*

respectively, if and only if for all $s \geq t$:

$$\alpha^l(s-t) \leq R(s) - R(t) \leq \alpha^u(s-t) \tag{2.1}$$

with $\alpha^u(0) = \alpha^l(0) = 0$.

It's worth noting that the condition must hold for any time interval with length $\Delta = s - t$.

As Def. 2.1 indicates, arrival curves $\alpha^u(\Delta)$ and $\alpha^l(\Delta)$ actually upper and lower bound the number of events arriving in any time interval with length Δ. For instance, consider the example trace in Fig. 2.1, we can derive its upper arrival curve $\alpha^u(\Delta)$ satisfies $\alpha^u(1) \geq 1$ since there is one event arrival in time interval $[0, 1]ms$, if we set the time unit as millisecond. Similarly, we have $\alpha^l(6) = 0$ since no event arrives in time interval $[5, 11]ms$.

Arrival curves substantially generalize classical event timing models such as periodic, sporadic, periodic with jitter or other event models including non-determinism timing behavior. Thus, they are well suited to representing the complex event streams in hard real-time systems. For example, a periodic event stream can be abstracted by a set of step function where $\alpha^u(\Delta) = \lfloor \frac{\Delta}{p} \rfloor + 1$ and $\alpha^l(\Delta) = \lfloor \frac{\Delta}{p} \rfloor$. A sporadic event stream can also be modeled by $\alpha^u(\Delta) = \lfloor \frac{\Delta}{p} \rfloor + 1$, $\alpha^l(\Delta) = \lfloor \frac{\Delta}{p'} \rfloor$, where p and p' are the minimal and maximal inter arrival distance of the event stream, respectively. Moreover, for an event stream which can be specified by a period p, jitter j and minimal inter arrival distance d, the upper arrival curve is $\alpha^u(\Delta) = \min\{\lceil \frac{\Delta+j}{p} \rceil, \lceil \frac{\Delta}{d} \rceil\}$. Fig. 2.2 demonstrates the arrival curves of different event timing models.

We consider not only single event streams but also multi-event streams. For multi-event scenarios, N event streams are supposed in the input source, where $N \geq 2$. We order the event streams $S_1, S_2, \cdots, S_N$ according to their relative deadlines, where D_i, the relative deadline of event stream S_i, is smaller than that of S_j when $i < j$. Thus, the input event model of our processor can be depicted by the tuple $\mathbf{EM}(N) = (\alpha(\Delta)_1, c_1, D_1, \cdots, \alpha(\Delta)_N, c_N, D_N)$, where $\alpha(\Delta)_i$ denotes the arrival curve tuple of event stream S_i.

2.3.2 Service Model

The general model arrival curve abstract the cumulative function $R(t)$ for the worst-case and best-case event arrivals. Similarly, the service

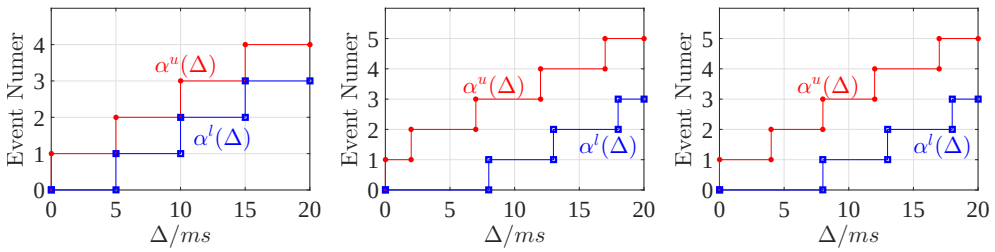


Figure 2.2: Example arrival curves for (a) periodic event streams with period $5ms$, (b) event streams with period $5ms$ and jitter $j = 3ms$, (c) event streams with period $5ms$, jitter $j = 3ms$ and minimal inter-arrival distance $d = 4ms$.

providing ability of the system can also be described by a cumulative function $C(t)$ and then modeled by the service curve. The function $C(t)$ is defined as the amount of total time slots provided by the system to handle workloads in time interval $[0, t]$. It's also a wide-sense increasing and discontinuous function. In the same way, the service curve is defined as:

Definition 2.2 (Service Curve) *For a system C and a 2-tuple wide-sense increasing functions $\beta(\Delta) = [\beta^u(\Delta), \beta^l(\Delta)]$ defined for $\Delta >= 0$, we say C has $\beta^u(\Delta)$ and $\beta^l(\Delta)$ as upper service curve and lower service curve, respectively, if and only if for all $s \geq t$:*

$$\beta^l(s-t) \leq C(s) - C(t) \leq \beta^u(s-t) \tag{2.2}$$

with $\beta^u(0) = \beta^l(0) = 0$.

Service curve is also an abstract model and can generalize traditional resource models such as Time Division Multiple Access (TDMA) and periodic model [89]. For example, consider a bus with bandwidth B that implements TDMA model, then a slot can be represented by service curves: $\beta^l(\Delta) = B \cdot \min\{\lceil \Delta/l \rceil, \Delta - \lfloor \Delta/l \rfloor (l - s_i)\}$ and $\beta^u(\Delta) = B \cdot \max\{\lceil \Delta/l \rceil, \Delta - \lfloor \Delta/l \rfloor (l - s_i)\}$, where s_i is the length of the slot and l denotes the TDMA cycle length.

Note that the arrival curves $\alpha(\Delta)$ is event-based and specifies the upper and lower bounds of the number of input events in any time interval Δ, while the service curve $\beta(\Delta)$ is time-based and specifies the upper and lower bounds of the amount of available execution time in any time interval Δ. Thus, operations involving both of them cannot be

performed directly. The event-based arrival curve is transformed to the time-based arrival curve $\bar{\alpha}(\Delta)$ for correct operation results. Suppose that the worst-case execution time of one event in arrival stream is c, then the arrival curve transformation can be performed as $\bar{\alpha}^u(\Delta) = c \times \alpha^u(\Delta)$ and $\bar{\alpha}^l(\Delta) = c \times \alpha^l(\Delta)$ [50].

For brevity, in the following of this chapter, the time-based arrival is also termed as arrival curve, denoted by $\bar{\alpha}(\Delta)$.

2.3.3 Basic Results

In this section we discuss the main basic real-time calculus result presented in [60] which is useful to analyze how to guarantee deadline constraints for hard real-time systems.

Theorem 2.3 (Delay Bound) *Consider an event stream, constrained by upper arrival curve $\bar{\alpha}^u(\Delta)$, is processed by a system that offers a lower service curve $\beta^l(\Delta)$. Then the maximal possible delay $d(t)$ experienced by any event arriving at time t satisfies the following condition if the events arriving before it are handled before it.*

$$d(t) \leq h(\bar{\alpha}^u, \beta^l) \tag{2.3}$$

where $h(\alpha, \beta)$ denotes the supremum of horizontal deviations between α and β and is defined as:

$$h(\alpha, \beta) = \sup\{\delta(s) : \delta(s) = \inf\{\tau \geq 0 : \alpha(s) \leq \beta(s+\tau)\}\} \tag{2.4}$$

The conclusion of Thm. 2.3 is intuitive. It indicates the delay experienced by any event is upper bounded by the supremum of horizontal deviations between upper arrival curve and lower service curve. An example is shown in Fig. 2.3. The figure also graphically demonstrates the condition of meeting deadline constraints for a hard real-time system, which is given below.

Theorem 2.4 (Deadline Condition) *Given an event stream with relative deadline D which is constrained by upper arrival curve $\bar{\alpha}^u(\Delta)$, a system can guarantee the delay of any event is no larger than D if its lower service curve meets following condition.*

$$\beta^l(\Delta) \geq \bar{\alpha}^u(\Delta - D) \tag{2.5}$$

Proof Thm. 2.4 is actually a reverse representation of Thm. 2.3. We prove it by contradiction. Suppose the delay of one or more event is larger than D while (2.5) holds. From Thm. 2.3, it's clear that $h(\bar{\alpha}^u, \beta^l) >$

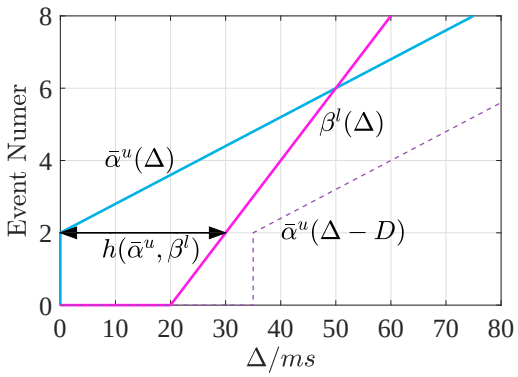


Figure 2.3: The delay bound and deadline condition for an event stream with relative deadline D, constrained by $\bar{\alpha}^u(\Delta)$, when it is served by a system offering $\beta^l(\Delta)$.

D holds, that is, there exists at least one $\delta(s) > D$. Since $\delta(s)$ is the infimum of τ that satisfies $\bar{\alpha}(s) \leq \beta(s+\tau)$, one can derive that $\bar{\alpha}(s) > \beta(s+D)$ for all $s > 0$, which contradicts the condition (2.5). □

2.4 System Model and Problem Statement

2.4.1 Hardware Model

A single core processor that has two power dissipation modes, i.e., 'active' and 'sleep' mode, is adopted in this chapter. The processor must be in 'active' mode with a fixed speed to process coming event streams and can be turned to 'sleep' mode with a lower power consumption when there is no event to handle.

We consider the time and power overheads during model-switching. Let t^{off} and t^{on} denote the time units required to switch the processor from 'active' mode to 'sleep' mode and back, respectively. During mode switching, the power dissipation equals that in 'active' mode but the processor does not tackle any coming event. The time and power overheads during mode switching have nontrivial impacts on the resource providing capability and thermal evolution of the processor. For example, suppose the processor is switched to 'active' mode first and then t^{on} time units later it is turned to 'sleep' mode and stays at this mode for t^{off} time units. As shown in Fig. 2.4, in this $(t^{on} + t^{off})$ units time interval, the length of the overall time slots in which the processor can handle coming events is $t^{on} - t^{swon}$, which is less than t^{on}. In other words, each

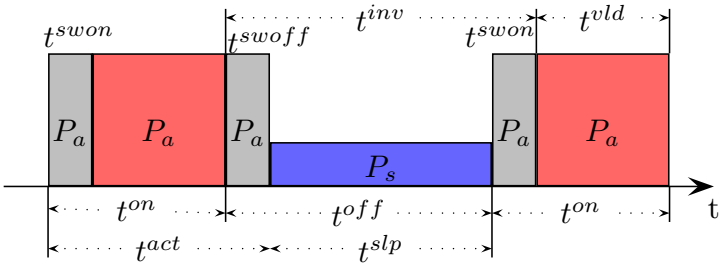


Figure 2.4: Hardware model of a single-core processor. The power consumptions in 'active' and 'sleep' modes are considered to be constant and are denoted as P_a and P_s, respectively.

mode-switching from 'sleep' to 'active' makes the *valid* serving time interval t^{swon} shorter. Similarly, in this $(t^{on} + t^{off})$ units time interval, the time interval during which the processor consumes power equals that in 'sleep' mode is $t^{off} - t^{swoff}$. Again, each mode-switching from 'active' to 'sleep' incurs an energy overhead and makes the *sleep* power consumption time interval t^{swoff} shorter. In conclusion, the mode-switching overhead leads to a higher temperature and a weaker resource providing capability. The quantitative impacts will be investigated later. Moreover, as shown in Fig. 2.4, to cover the mode-switching overhead, the time lengths for which the processor is switched to 'active' and 'sleep' mode must be larger than t^{swon} and t^{swoff}, respectively:

$$t^{off} > t^{swoff} \tag{2.6}$$

$$t^{on} > t^{swon} \tag{2.7}$$

2.4.2 Power Model

We consider the total power dissipation at time t, denoted by $P(t)$, is composed of two parts: (1) the dynamic power P^d due to dynamic current and (2) the leakage power P^l due to leakage current [43, 81].

Dynamic power P^d is consumed when the transistors inside a processor are active, i.e., switching between different states. It can be calculated by the following equation.

$$P^d \propto a \cdot {V_{dd}}^2 f \tag{2.8}$$

where a is a constant coefficient mainly depending on the wire length, V_{dd} is the supply voltage, and f is the clock frequency. From this equation, one can conclude that the dynamic power is primarily determined

by V_{dd} and f. Therefore, we consider P^d keeps constant in each power mode, i.e., P_a and P_s, in the 'active' and 'sleep' mode, respectively.

The leakage power mainly comes from the leakage current of the transistors which is influenced by the temperature and the clock frequency. The dependency relationship between the leakage power and the temperature can be closely approximated by a linear function of the processor temperature, which has been widely adopted [42, 97, 43, 68, 86]:

$$P^l(t) = \begin{cases} \varphi \cdot T(t) + v_a & \text{if in active mode} \\ \varphi \cdot T(t) + v_s & \text{if in sleep mode} \end{cases} \tag{2.9}$$

where w, v_a and v_s are constant coefficients, $T(t)$ is the temperature of the processor at time t.

In summary, the total power consumption can be represented as:

$$P(t) = \begin{cases} \varphi \cdot T(t) + \theta_a & \text{if in active mode} \\ \varphi \cdot T(t) + \theta_s & \text{if in sleep mode} \end{cases} \tag{2.10}$$

where $\theta_a = v_a + P_a$ and $\theta_s = v_s + P_s$.

2.4.3 Thermal Model

In this section, we introduce the thermal model of the processor, which is based on the well-known Fourier law of heating [80], which can be described by the following equation:

$$C\frac{dT}{dt} = P(t) - G(T - T_{amb}) \tag{2.11}$$

where T, C, and G denote the temperature, thermal capacitance, and thermal conductance of the processor, respectively. T_{amb} indicates the ambient temperature. In addition, the absolute temperature (Kelvin, K) is set as the unit of all temperature variables.

From (2.10) we have $P(t) = \varphi T(t) + \theta$ when the processor stays in one power mode. Rewriting (2.11), we have

$$\frac{dT}{dt} = -mT(t) + n \tag{2.12}$$

where $m = \frac{G-\varphi}{C}, n = \frac{\theta + GT_{amb}}{C}$. Since m and n are constants, a closed-form solution of the temperature yields:

$$T(t) = T^{\infty} + (T_{init} - T^{\infty}) \cdot e^{-m \cdot t} \tag{2.13}$$

where T_{init} indicates the initiate temperature, and T^{∞} is the steady-state temperature of currently power mode, which can be obtained by solving $\frac{dT}{dt} = 0$.

$$T^{\infty} = \frac{n}{m} \tag{2.14}$$

Then, combining (2.10) and (2.14), the coefficient for (2.13) are given as [80, 55]:

$$\begin{aligned} m_a = \frac{G - \varphi_a}{C}\,,\ m_s = \frac{G - \varphi_s}{C} \\ T_a^{\infty} = \frac{\theta_a + GT_{amb}}{G - \varphi_a}\,,\ T_s^{\infty} = \frac{\theta_s + GT_{amb}}{G - \varphi_s} \end{aligned} \tag{2.15}$$

In addition, we also regulate the thermal model by these following circumstances.

- $m_a > 0$ and $m_s > 0$.
- The steady-state temperature in 'active' mode is non-smaller than the one in 'sleep' mode, that is, $T_a^{\infty} \geq T_s^{\infty}$.
- The initial temperature $T_{init} = T_{amb} \leq T_s^{\infty}$.

Finally, the thermal mode of the processor in this chapter is characterized by the tuple $\mathbf{TM} = (T_a^{\infty}, m_a, T_s^{\infty}, m_s)$.

2.4.4 Problem Statement

Dynamically switching the processor into 'sleep' mode according to the event arrivals is an effective way to minimize the peak temperature. However, this needs vast calculating efforts, which hampers the efficiency. Periodic thermal management (PTM), a trade-off between effect and efficiency, is adopted in this chapter to minimize the peak temperature by periodically putting the system into 'active' and 'sleep' modes. In each period, the processor stays at 'active' mode and 'sleep' mode for t^{on} and t^{off} time units, respectively. In addition, $t^{p} = t^{on} + t^{off}$ denotes the length of the period.

We illustrate our approach with an example in which three thermal management policies are adopted: (a) a work conserving (WC) execution that with no DTM policy, which means that the processor stays at 'active' mode to process events if there is (at least) one event in the ready queue, (b) an online DPM policy called Cool Shaper (CS) which dynamically

Item	value
period	$200ms$
jitter	$50ms$
minimal inter-arrival distance	$1ms$
execution time	$110ms$
relative deadline	$320ms$
event arriving times	$(0, 150, 350, 550)ms$

Table 2.1: The concrete event trace adopted in the example.

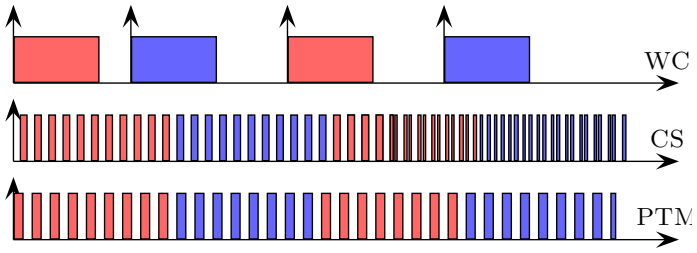


Figure 2.5: Execution of jobs in policy WC, DT and PTM.

transits the processor into 'sleep' mode according to the event arrivals, and (c) periodic thermal management (PTM). The thermal and hardware parameters are described in Tab. 2.2. A concrete trace of events is adopted in this example. The parameters specifying the concrete trace are list in Tab. 2.1.

Fig. 2.5 and Fig. 2.6 show the execution of events and the temperature evolution for the three policies, respectively. As shown in Fig. 2.6, the peak temperature in policy PTM is slimly higher than the one in policy CS and they are both about 9 K less than the one in policy WC. This indicates that PTM policy can achieve close results to CS policy in terms of peak temperature and they are both effective compared to WC policy. From Fig. 2.5, we find that PTM can be seen as an approximate policy of CS, this interprets why the peak temperature of PTM is slimly higher. Despite of this, PTM requires less resources for computation with acceptable results and is very convenient to implement.

This chapter considers the temperature varying in a time interval L, where $L >> t$ and L/t is an integer. Due to the model-switching overhead, t_{on} and t_{off} cannot be directly utilized into thermal mode and service curve. Before giving the revised solutions, we first define some notations. From Fig. 2.4, t_{act} and t_{slp} denote the time interval that the

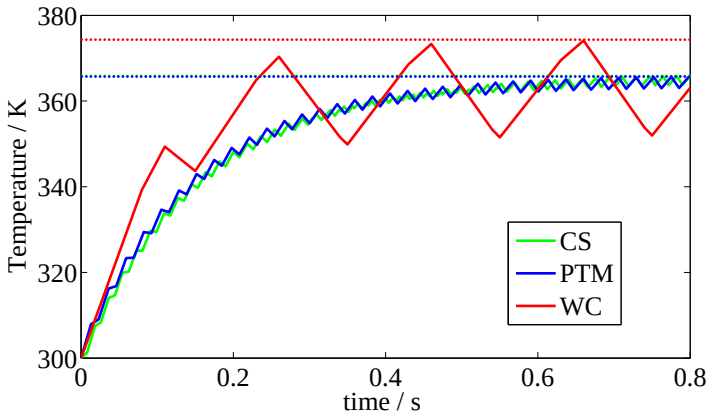


Figure 2.6: Temperature evolution in policy WC, DT and PTM.

processor consumes power P_a and P_s in one period, respectively. Analogously, t_{vld} denotes the time interval that the processor can tackle coming events in one period and t_{inv} represents the rest. Based on hardware model, we formulate them as:

$$t_{act} = t_{on} + t^{swoff}, \; t_{slp} = t_{off} - t^{swoff} \tag{2.16}$$

$$t_{vld} = t_{on} - t^{swon}, \; t_{inv} = t_{off} + t^{swon} \tag{2.17}$$

With these definitions, one can use t_{act} and t_{slp} to derive the peak temperature and t_{vld} and t_{inv} to calculate the service curve of the processor; meanwhile, the time and power overhead of mode-switching are considered.

Now we define our problem as follows:
Given a system characterized by the power model and the thermal model ***TM*** *described in the preceding pages, task streams that are modeled by* $\boldsymbol{EM}(N)$*, our goal is to derive a periodic thermal management depicted by* t^{on} *and* t^{off} *such that the peak temperature is minimized while all the events complete within their deadlines.*

2.5 Peak Temperature Analysis

In this section, we derive the formula of the peak temperature in PTM such that our algorithm can utilize it as a criterion of the optimal pair of $< t^{on}, t^{off} >$.

Since PTM periodically transits the system between two power modes, the values of the parameters in the temperature model (2.13) change pe-

riodically, which causes the general solution of the transient temperature T very complicated. Therefore, instead of utilizing the general solution, we derive the formula of the peak temperature based on some basic lemmas, which are obtained from close observations of the temperature evolution and are presented in the following.

Lemma 2.5 *With a periodic thermal management* PTM *(t^{on}, t^{off}), the temperature of the processor ceaselessly rises in the opening few periods and then rises in t^{act} and descends in t^{slp} in every following period.*

Proof As mentioned before, the very initial temperature $T_{init} = T_{amb} \leq T_s^\infty \leq T_a^\infty$. Based on (2.12), inequality $\frac{dT}{dt} \geq 0$ holds in the beginning several periods when $T(t) \leq T_s^\infty$. Therefore, temperature $T(t)$ continuously rises and then reaches T_s^∞. It's worth noting that $T(t)$ will never surpass T_a^∞ unless the initial temperature is higher than T_a^∞, as T_a^∞ is the steady-state temperature of the 'active' mode. Since $T(t)$ has already passed T_s^∞, it also will never drop back below T_s^∞ until the processor being completely shut down. Therefore, one can summarize the temperature $T(t)$ under PTM will keep changing between T_s^∞ and T_a^∞ once T passes T_s^∞, that is:

$$T_s^\infty \leq T(t) \leq T_a^\infty. \tag{2.18}$$

Now, assume the processor switches to 'active' mode at time $\bar{t}$ in one PTM period. Note that the T_{init} in (2.13) is actually $T(\bar{t})$. Then, we have:

$$\frac{dT}{dt} = -m_a(T(\bar{t}) - T_a^\infty)e^{-m_a \cdot t} > 0 \tag{2.19}$$

Furthermore, One can easily derive $\frac{dT}{dt} < 0$ following the similar derivation. □

Based on Lem. 2.5, in the jth period, the temperature T reaches its local maximum T_j at the end of the time interval t^{act}. Therefore, we can define the peak temperature of the processor.

Definition 2.6 (Peak Temperature) *For a single-core processor under* PTM, *the peak temperature $T^\star$ in a time interval L can be defined as the maximum of all the T_j:*

$$T^\star = max(T_1, \cdots, T_{\frac{L}{t}}). \tag{2.20}$$

As shown in Fig. 2.7, the local maximum increases in the beginning and then stays at a stable value in the rest time. This reveals that the peak temperature can be obtained based on the difference between two consecutive local maximums, which is depicted in the following lemma.

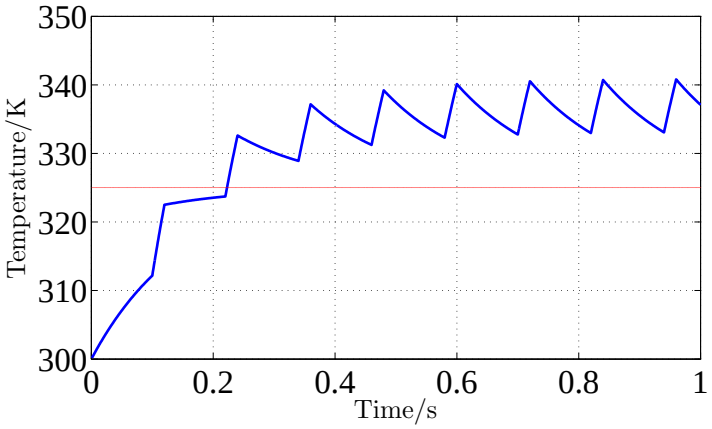


Figure 2.7: Example of temperature varying with PTM($t^{on} = 0.02s, t^{off} = 0.1s$) while the model-switching overhead is not considered. The thermal and hardware parameters are described in Tab. 2.2.

Lemma 2.7 *Denoting the local maximal temperature in the jth period as T_j, the temperature difference between two consecutive local maximums, $T_{j+1} - T_j$, can be formulated as:*

$$\begin{aligned} T_{j+1} - T_j =& (1 - e^{-m_a t_{act}}) T_a^{\infty} + e^{-m_a t_{act}} (1 - e^{-m_s t_{slp}}) T_s^{\infty} \\ & - [1 - e^{-m_a t_{act} - m_s t_{slp}}] T_j \end{aligned} \tag{2.21}$$

where t^{act} and t^{slp} are from (2.16).

Proof With T_j, t^{slp} and (2.13), we first derive t_j^{slp}, which is the temperature at the end of time interval t^{slp} in the jth period. From (2.13), one can get $t_j^{slp} = T_s + (T_j - T_s) e^{-m_s t^{slp}}$. Then, combining t^{act}, T_a and t_j^{slp}, (2.13) generates the following equation:

$$\begin{aligned} T_{j+1} =& (1 - e^{-m_a t_{act}}) T_a^{\infty} + e^{-m_a t_{act}} (1 - e^{-m_s t_{slp}}) T_s^{\infty} + \\ & e^{-m_a t_{act} - m_s t^{slp}} T_j. \end{aligned} \tag{2.22}$$

Subtracting T_j from both sides yields (2.21). □

With above lemmas, the first main result of this chapter is presented as the theorem below:

Theorem 2.8 *Given a system as stated above and a periodic thermal management* PTM *(t^{on}, t^{off}), the peak temperature of the processor is a linear combination of T_a^{∞} and T_s^{∞}, which is given as:*

$$T^{\star} = \lambda T_a^{\infty} + (1 - \lambda) T_s^{\infty}, \tag{2.23}$$

where

$$\lambda = \frac{1 - e^{-m_a t^{act}}}{1 - e^{-m_a t^{act} - m_s t^{slp}}}.$$

Proof We prove Thm. 2.8 by contradiction. For brevity, polynomial expression $\lambda T_a^{\infty} + (1-\lambda) T_s^{\infty}$ is denoted as $T^{\diamond}$.

First, suppose that the peak temperature $T^{\star}$ is reached in the ith period and $T^{\star} < T^{\diamond}$. Since Lem. 2.5 indicates the local peak temperature in a period is reached at the end of t^{act}, we have $T_i = T^{\star} < T^{\diamond}$. Rewriting (2.21) yields that $T_{i+1} - T_i > 0$, which contradicts the presumption that T_i is the peak temperature of the processor.

Similarly, assume that the peak temperature $T^{\star}$ is reached in jth period and $T_j = T^{\star} > T^{\diamond}$. Therefore we have:

$$T_j - T_{j-1} > T^{\diamond} - T_{j-1} \tag{2.25}$$

According to Lem. 2.7:

$$\begin{aligned} T_j - T_{j-1} &= (1 - e^{-m_a t^{act} - m_s t^{slp}})[\lambda T_a^{\infty} + (1-\lambda) T_s^{\infty} - T_{j-1}] \\ &= (1 - e^{-m_a t^{act} - m_s t^{slp}})(T^{\diamond} - T_{j-1}). \end{aligned}$$

Since $(1 - e^{-mt}) < 1$, the following inequality yields:

$$T_j - T_{j-1} < T^{\diamond} - T_{j-1} \tag{2.26}$$

which is in conflict with (2.25). In conclusion, $T^{\star} = T^{\diamond}$. □

Next, the boundaries of t^{on} and t^{off} are explored, then two approaches are proposed to minimize T^{*}.

2.6 Real-Time Calculus Routine

In this section, real-time analysis is first presented to give deadline constraints on the PTM scheme. Then, two algorithms are presented to compute the solution of the PTM scheme with different levels of accuracy and speed.

2.6.1 Service Bound of PTM

Real-time interface is employed in this chapter to analyze how to ensure all events complete within their deadlines. With the hardware model

described before and a given PTM (t^{on},t^{off}), the lower service curve of the processor is written as:

$$\beta_R(\Delta) = \max\left(\left\lfloor\frac{\Delta}{t}\right\rfloor \cdot t^{vld},\ \Delta - \left\lceil\frac{\Delta}{t}\right\rceil \cdot t^{inv}\right), \tag{2.27}$$

where t is the period, t^{vld} and t^{inv} are obtained from (2.17). According to Thm. 2.4, to satisfy the deadline constraints, the lower service curve of the processor $\beta_R(\Delta)$ should satisfy the following inequality:

$$\beta_R(\Delta) \geq \beta_B(\Delta),\ \forall\Delta \geq 0, \tag{2.28}$$

where $\beta_B(\Delta)$ is the service bound for the workload modeled by $\mathbf{EM}(N)$.

For a single event stream ($N = 1$), $\beta_B(\Delta)$ can be simply formulated as:

$$\beta_B(\Delta) = \bar{\alpha}^u(\Delta - D) \tag{2.29}$$

For multi-event streams ($N \geq 2$), the service bound $\beta_B(\Delta)$ in (2.28) should be computed based on the scheduling policy. Note that only the service bound $\beta_B(\Delta)$ has to be revised. The other parts of our algorithms can remain untouched. Suppose the scheduling policy of earliest deadline first (EDF) is adopted, the service bound for the N event streams is [50]:

$$\beta_B(\Delta) = \sum_{i=1}^{N} \bar{\alpha}_i^u(\Delta - D_i). \tag{2.30}$$

It's worth noting that EDF is not necessarily the only one scheduling policy can be adopted here. For example, when fixed priority (FP) scheduling is employed, the service bound can be calculated according to another formula [51] and fits in with our algorithms as suitable as EDF.

2.6.2 Principles of our Algorithms

In this chapter, our goal is to find the optimal $< t^{on}, t^{off} >$ under the deadline constraints. Apparently brutal searching the whole two-dimensional space is the least efficient way to find the solution and thus is not adopted in our approach.

Based on (2.23), one can find that the derivative of $T^\star$ with respect to t^{on} is:

$$\frac{\mathrm{d}T^\star}{\mathrm{d}t^{on}} = (T_a^\infty - T_s^\infty)\frac{m_a e^{-m_a(t^{on}+t^{swoff})}[1 - e^{-m_s(t^{off}-t^{swoff})}]}{[1 - e^{-m_a t^{act} - m_s t^{slp}}]^2} > 0 \tag{2.31}$$

Therefore, for a given t^{off}, $T^{\star}$ can be minimized by searching the minimal t^{on} under the service curve constraint, (2.28). Based on this feature, we can design algorithms searching the best solution of PTM based on below two principles:

1. For a given t^{off}, the optimal t^{on} which leads to he minimal peak temperature is the minimum of the t^{on}s satisfying the real-time constraint (2.28).
2. The best pair of $< t^{on}, t^{off} >$ can be found by searching t^{off} in its feasible region while following above principle to obtain t^{on}.

2.6.3 Feasible Region of t^{off}

In order to discover the minimal t^{on}, the feasible region of t^{off} should be determined first such that one can assure the solution to the minimal t^{on} exists. For example, when the input is a single event stream and $t^{off} = D$, coming events in worst-case will miss their deadlines before they are processed, considering additional t^{swon} time units are required to switch the processor on. According to the hardware model, we directly know that t^{off} has to be no less than t^{swoff} to cover the timing overhead of model-switching. To avoid situations similar to the example, t^{off} must be bounded by an upper bound, which is calculated according to the maximum service curve in [50]:

$$t^{off}_{max} = \max\left\{t^{off} : \beta^{\top}_{R}(\Delta) \geq \beta_{B}(\Delta),\, \forall \Delta \geq 0\right\}, \tag{2.32}$$

where $\beta^{\top}_{R}(\Delta)$ can be formulated as follows when we take t^{swon} into account:

$$\beta^{\top}_{R}(\Delta) = \max\{0,\, \Delta - t^{off} - t^{swon}\} \tag{2.33}$$

Moreover, from Section 2.4.1, t^{off} should be larger than the mode-switching overhead $t^{off} > t^{swoff}$. Finally, the feasible region of t^{off} can be depicted as $t^{off} \in [t^{swoff}, t^{off}_{max}]$.

2.6.4 Obtaining the minimal t^{on}

Precise Solution

Based on the constraint (2.28), when t^{off} is fixed, the precise solution of minimal t^{on} can be calculated.

Definition 2.9 (Precise t^{on}) *Given t^{off}, the precise t^{on} which not only satisfies real-time constraint* (2.28) *but also is thermal optimal can be given as:*

$$t^{on}_{prc} = \min\left\{t^{on} : \beta_R(\Delta) \geq \beta_B(\Delta),\, \forall\Delta \geq 0\right\}. \tag{2.34}$$

This solution can be found by testing the t^{on}s starting from t^{swon} with step ε until the minimal t^{on} satisfying (2.28) is discovered. By this method, the minimum of t^{on} can be obtained with high accurateness while the time consumption is significant. To reduce the computational overhead, another method which can find an approximated solution efficiently is presented below.

Approximated Solution

In this section, an fast method is proposed to compute the minimum of t^{on}. The basic idea of this method is adopting the bounded-delay function [22, 50] to calculate an approximate minimal t^{on}.

Definition 2.10 (Bounded-Delay Function) *A bounded-delay function for interval length Δ is defined by the slope η and the bounded-delay t^{off}:*

$$bdf(\Delta, \eta, t^{off}) = \max[0, \eta(\Delta - t^{off})] \tag{2.35}$$

Now, given a t^{off}, the proposed method first finds the bounded-delay function defined by t^{off} and the slope $\eta(t^{off})$ which is given as:

$$\eta(t^{off}) = \inf\{\rho : \rho(\Delta - t^{off}) \geq \beta_B(\Delta), \forall\Delta \geq 0\} \tag{2.36}$$

An example of this bounded-delay function is shown in Fig. 2.8 in red color dashed lines. Then, the approximation of minimal t^{on} can be calculated by solving $\frac{t^{on}}{t^{on}+t^{off}} = \eta(t^{off})$.

The advantage of this method is twofold: (1) the slope $\eta(t^{off})$ can be obtained by using bisection method, which is highly efficient and require little computational effort, (2) the peak temperature of the processor controlled by PTM calculated in this method is a unimodal function of t^{off}, which makes the golden-section method feasible for searching best t^{off}. Utilizing both advantages, we can get the solution of $< t^{on}, t^{off} >$ efficiently.

When the mode-switching overhead is ignored, the approximate minimal t^{on} can be calculated as $t^{on}_{apx} = \frac{\eta(t^{off})\cdot t^{off}}{1-\eta(t^{off})}$ (Refer to Fig. 2.8 for the

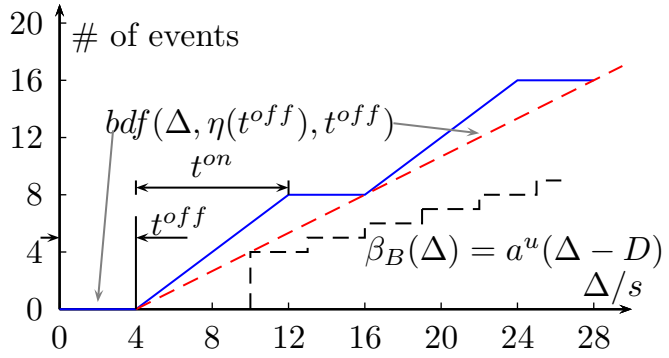


Figure 2.8: Obtaining the approximate minimal t^{on} based on the bounded-delay function when the mode-switching overhead is not considered.

derivation). Since we take the time overhead into account, this equation is revised as $t^{vld}_{apx} = \frac{\eta(t^{inv}) \cdot t^{inv}}{1-\eta(t^{inv})}$. Based on (2.17), the revised approximate t^{on} is denoted as:

$$\begin{aligned} t^{on}_{apx} &= t^{vld}_{apx} + t^{swon} \\ &= \frac{\eta(t^{off} + t^{swon})}{1 - \eta(t^{off} + t^{swon})} \cdot (t^{off} + t^{swon}) + t^{swon} \\ &= \frac{\eta(t^{off} + t^{swon})}{1 - \eta(t^{off} + t^{swon})} \cdot t^{off} + \frac{t^{swon}}{1 - \eta(t^{off} + t^{swon})} \\ &= \frac{\eta(t^{inv})}{1 - \eta(t^{inv})} \cdot t^{off} + \frac{t^{swon}}{1 - \eta(t^{inv})} \end{aligned} \tag{2.37}$$

Definition 2.11 *A linear function termed as* $RVT(\eta, t_1, t_2)$ *is defined as:*

$$RVT(\eta, t_1, t_2) = \frac{\eta \cdot t_1}{1 - \eta} + \frac{t_2}{1 - \eta}$$

Definition 2.12 (Approximated t^{on}) *Given* t^{off}*, the approximated* t^{on} *which not only satisfies real-time constraint* (2.28) *but also is thermal sub-optimal can be given as:*

$$t^{on}_{apx} = RVT(\eta(t^{off} + t^{swon}),\ t^{off},\ t^{swon}) \tag{2.39}$$

Example 2.13 *Consider an periodic event stream with period being* 100*ms. The* WCET *and the relative deadline of the events are given as* 10*ms and* 120*ms, respectively. Suppose the mode-switching overheads are* $t^{swon} = t^{swoff} =$

$5ms$. *Then, given a* $t^{off} = 55ms$, *we can first calculate the slope* $\eta(t^{inv}) = \frac{10}{120-55-5} = \frac{1}{6}$. *Further, the valid time interval* $t^{vld} = 12ms$, *which is given by* $\frac{t^{vld}}{t^{vld}+t^{inv}} = \eta(t^{inv}) = \frac{1}{6}$. *Finally, we have the approximated solution* $t^{on} = t^{vld} + t^{swon} = 17ms$, *which is identical with* $RVT(\eta(t^{inv}), t^{off}, t^{swon})$;

2.7 PTM Algorithms

Based on the precise and the approximated solutions of minimal t^{on}, two algorithms with different levels of accuracy and efficiency are presented to minimize $T^{\star}$, namely PMPT (precisely minimizes the peak temperature) and AMPT (approximately minimizes the peak temperature).

2.7.1 Algorithm PMPT

For a given t^{off}, the corresponding thermal-optimal t^{on} in this algorithm is calculated based on (2.34), which derives a precise solution. With t^{on}_{prc}, the corresponding peak temperatures $T^{\star}$ of all the tested t^{off}s can be computed from Thm. 2.8, then corresponding t^{off} of the minimal $T^{\star}$ in all the tested points is the optimal solution.

Note that the function $T^{\star}(t^{off})$ is irregular and has several local minima in the domain of t^{off}. As shown in Fig. 2.9. Therefore, a thorough search with a fixed step ε in the feasible region of t^{off} is implemented to find the global minimal $T^{\star}$.

Algo. 1 outlines the pseudo-code of algorithm PMPT. It takes as input the thermal model **TM**, the input event model $\mathbf{EM}(N)$, the time overheads of mode-switching and the accuracy coefficients ε and ξ. The service bound is obtained based on the input and scheduling policy (line 1) and then the upper bound of t^{off} is generated (line 2). Then the optimal solution and minimal $T^{\star}$ are initialized in line 3. Lines 4-13 iteratively discover the t^{on}_{prc} and calculate the peak temperature $T^{\star}$ for all t^{off}s in the feasible region with a step ε. The t^{inv} is computed by (2.17) to derive the lower service curve (line 5). Then t^{on}_{prc} is found by examining every candidate of t^{on} from the lower bound, t^{swon}, with a step $\mathcal{S}$ (line 6). Afterwards, t^{on}_{prc} and t^{off} are revised based on (2.16) to derive the peak temperature (line 7). The peak temperature is calculated based on Thm. 2.8 (line 8) and then compared to $T^{\star}_{min}$. If the newly derived one is lower, the corresponding $< t^{on}, t^{off} >$ and $T^{\star}$ are assigned to the optimal solution and $T^{\star}_{min}$, respectively (lines 9-12).

Algorithm 1 PMPT

Input: **TM**, **EM**(N), t^{swon}, t^{swoff}, ε, ξ
Output: t^{on}_{opt}, t^{off}_{opt}

calculate $\beta_B(\Delta)$ based on **EM**(N) and the scheduling policy
get t^{off}_{max} from (2.32)
$T^{\star}_{min} = T^{\infty}_{a}$, $t^{on}_{opt} = 0$, $t^{off}_{opt} = 0$
for $t^{off} = t^{swoff}$ to t^{off}_{max} with step ε **do**
    get $t^{inv} = t^{off} + t^{swon}$ from (2.17)
    find t^{on} by testing (2.34) with step ξ
    compute t^{act} and t^{slp} by (2.16)
    $T^{\star}(t^{act}, t^{slp}) = \lambda(t^{act}_{min}, t^{slp})T^{\infty}_{a} + [1 - \lambda(t^{act}, t^{slp})]T^{\infty}_{s}$
    if $T^{\star}(t^{act}, t^{slp}) < T^{\star}_{min}$ **then**
        $t^{on}_{opt} \leftarrow t^{on}$, $t^{off}_{opt} \leftarrow t^{off}$
        $T^{\star}_{min} \leftarrow T^{\star}(t^{act}, t^{slp})$
    end if
end for

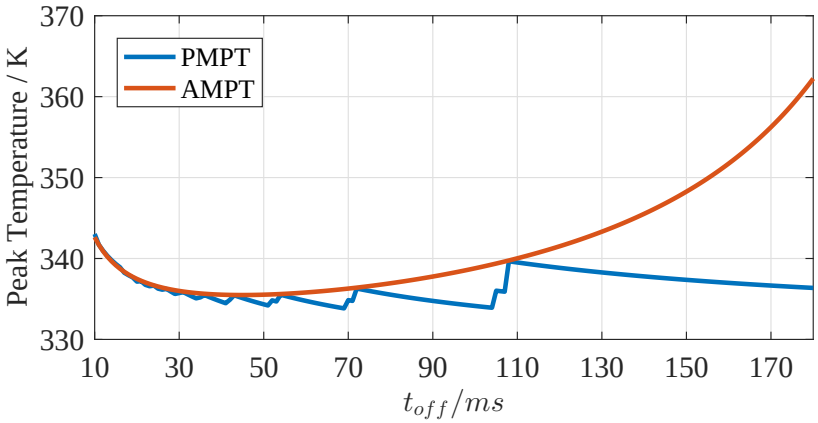


Figure 2.9: The relationship between the peak temperature and t^{off} when the corresponding t^{on} is calculated based on the precise and approximated methods.

2.7.2 Algorithm AMPT

In this algorithm, the minimal t^{on} is obtained directly from the approximation in (2.34) with less computation. Then according to Thm. 2.8, the

peak temperature can be formulated as a function of t^{off}:

$$\begin{aligned} T^{\star} &= PT(t^{off}) \\ &= T_s^{\infty} + \frac{1 - a \cdot e^{\frac{-m_a t^{swon}}{1-\eta}} \cdot e^{\frac{-m_a \eta}{1-\eta} t^{off}}}{1 - b \cdot e^{\frac{-m_a t^{swon}}{1-\eta}} \cdot e^{(m_s - \frac{-m_a \eta}{1-\eta}) t^{off}}} (T_a^{\infty} - T_s^{\infty}) \end{aligned} \tag{2.40}$$

where $a = e^{-m_a t^{swoff}}, b = e^{(m_s - m_a) t^{swoff}}$ and $\eta = \eta(t^{off} + t^{swon})$. Based on a set of systemic experiments (the details are included in appendix), we conjecture that $PT(t^{off})$ is a unimodal function which has only one minimum in the feasible region of t^{off}. Therefore the gold section search can be utilized to find the optimal t^{off} instead of searching all t^{off}s exhaustively. The pseudo-code is detailed in Algo. 2.

Algorithm 2 AMPT

Input: **TM**, **EM**(N), t^{swon}, t^{swoff}, ε
Output: t_{opt}^{on}, t_{opt}^{off}

1: calculate $\beta_B(\Delta)$ based on **EM**(N) and the scheduling policy
2: get t_{max}^{off} from (2.32)
3: $t_s^{off} \leftarrow t^{swoff}$, $t_e^{off} \leftarrow t_{max}^{off}$
4: define function: $f_1(x, y) = x + 0.312(y - x)$
5: define function: $f_2(x, y) = x + 0.618(y - x)$
6: $t_a^{off} \leftarrow f_1(t_s^{off}, t_e^{off})$, $t_b^{off} \leftarrow f_2(t_s^{off}, t_e^{off})$ ▷ Two tested points selected by gold section
7: $T_1^{\star} \leftarrow PT(t_a^{off})$, $T_2^{\star} \leftarrow PT(t_b^{off})$
8: **while** $t_b^{off} - t_a^{off} > \varepsilon$ **do**
9: **if** $T_1^{\star} > T_2^{\star}$ **then**
10: $t_2^{off} \leftarrow t_b^{off}$, $t_s^{off} \leftarrow t_a^{off}$, $t_a^{off} \leftarrow t_b^{off}$
11: $t_b^{off} \leftarrow f_2(t_s^{off}, t_e^{off})$, $T_1^{\star} \leftarrow T_2^{\star}$, $T_2^{\star} \leftarrow PT(t_b^{off})$
12: **else**
13: $t_2^{off} \leftarrow t_a^{off}$, $t_e^{off} \leftarrow t_b^{off}$, $t_b^{off} \leftarrow t_a^{off}$
14: $t_a^{off} \leftarrow f_1(t_s^{off}, t_e^{off})$, $T_2^{\star} \leftarrow T_1^{\star}$, $T_1^{\star} \leftarrow PT(t_a^{off})$
15: **end if**
16: **end while**

Algo. 2 has the same input as Algo. 1 except the accuracy coefficients ξ. The service bound and the upper bound of t^{off} are first derived (lines 1-2). Then the two endpoints of golden section selection are initialized as

Table 2.2: Thermal and hardware model parameters

G	C	$\varphi_i = \varphi_a$	θ_i	θ_a	T_{amb}	$t_{swon} = t_{swoff}$
0.3 $\frac{W}{K}$	0.03 $\frac{J}{K}$	0.1 $\frac{W}{K}$	-25 W	-11 W	300 K	0.1 ms

the lower and upper bounds of the feasible region of t^{off} (line 3). Based on the two setting functions defined in line 4 and line 5, the initial two tested points and their peak temperatures are calculated in line 6 and line 7. Lines 8 to 16 purely do the golden section selection to discover the optimal t^{off} such that $PT(t^{off})$ reaches its minimum. Since golden section selection is a well known algorithm, the details are not addressed herein.

2.7.3 Case Studies

In this section, we study the viability and efficiency of our algorithms and compare them with two approaches in [2, 55]. The simulations are implemented in Matlab (32 bit) using RTC-toolbox. All the results are obtained from a simulation platform with an Intel i7 4770 processor and 16 GB memory.

System Description

The thermal and power parameters are set as described in Tab. 2.2 [55, 80]. The task streams set studied in [101, 50] is used for our case studies and the parameters are summarized in Tab. 2.3. Earliest deadline first (EDF) is adopted as the scheduling policy for multi-event scenarios. The (p, j, d, c) event model is adopted to specified an input stream S_i by its period p, jitter j, minimal inter-arrival distance d of the stream and the worst-case execution time c. Note that other common timing models of event streams can also be employed in our case studies with the concept of arrival curve. We choose the (p, j, d, c) model because it is a commonly used model and the arrival curve can be easily generated by an existing formula. The relative deadline D_i is defined as $D_i = \chi * p_i$ and varies according to the deadline factor χ.

The online approach cool shaper (CS) studied in [55], which relies on not only the upper arrival curve but also the actual arrivals of the coming events to dynamically shut down the processor, and the approach TAPR (thermal-aware periodic resources) studied in [2] are adopted for the comparison. The input event model used in TAPR is sporadic task

Table 2.3: Event stream setting

	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}
p (msec)	198	102	283	354	239	194	148	114	313	119
j (msec)	387	70	269	387	222	260	91	13	302	187
d (msec)	48	45	58	17	65	32	78	-	86	89
c (msec)	12	7	7	11	8	5	13	14	5	6

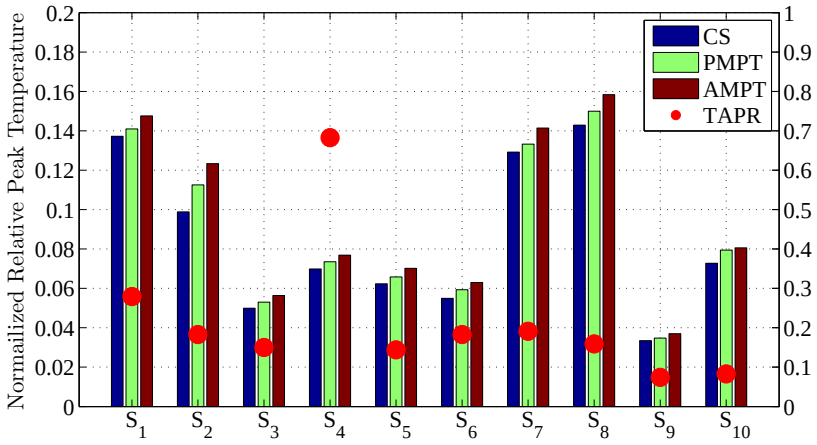


Figure 2.10: Normalized Relative Peak Temperature produced by the tested approaches for single event stream scenarios with $\chi = 1$. The right Y axis indicates the NRPT of approach TAPR.

(c, D, P), which is characterized by a worst-case execution time c, a (relative) deadline D and a minimum inter-arrival separation P. This model does not contain all the information of our (p, j, d, c) event model. Therefore, we revised P in a sporadic task as $\max[(p - j), d]$ to satisfy the worst-case deadline constraints. With these setups, Our algorithms are compared for both single and multi-event scenarios.

Simulation Results

First, we compare the minimal peak temperature derived by the four approaches. It is worth noting that the differences between the numerical values of those minimal peak temperature are hard to distinguish compared to their much larger absolute values. Thus the Normalized Relative Peak Temperature ($NRPT$), which is defined in the following,

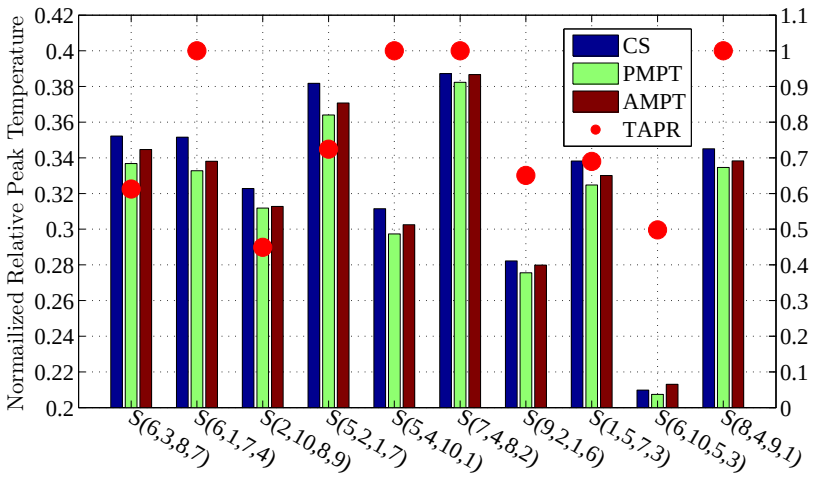


Figure 2.11: Normalized Relative Peak Temperature produced by the tested approaches for ten sets of randomly selected four-events stream scenarios with $\chi = 1$ by applying EDF scheduling. The right Y axis indicates the NRPT of approach TAPR.

is employed as the index to evaluate the approaches:

$$NRPT_A = \frac{T_A^\star - T_s^\infty}{T_a^\infty - T_s^\infty} \tag{2.41}$$

where $NRPT_A$ and $T_A^\star$ is the Normalized Relative Peak Temperature and the minimal peak temperature produced by approach 'A', respectively. From its definition, a smaller $NRPT$ indicates that the approach can better minimize the peak temperature.

Fig. 2.10 describes the $NRPT$ for all the single event streams. Fig. 2.11 to Fig. 2.12 reveal the results for four-events and five-events scenarios, respectively. Fig. 2.13 shows the derived minimal peak temperature w.r.t. different relative deadlines for the four approaches while taking all streams as input. Since the results of TAPR are much higher than those of the other three approaches, we display the results of TAPR with another Y axis in these four figures. Note that in multi-event scenarios, the arrival curves in CS must be approximated for EDF scheduling. Otherwise, the arrive curves will be too complicated and cause memory overflow for the JVM in Matlab [1].

From Fig. 2.10 to Fig. 2.13, we state below observations. (1) In all these cases, approach PMPT generates better or no worse results than approach AMPT, this is expected because PMPT brutally searches all the

possible solutions to get the precise t_{on} while AMPT relies on the approximate t_{on} to minimize the peak temperature. (2) For algorithms PMPT and AMPT, the minimized peak temperatures in four-events and five-event scenarios are much higher ($NRPT$s stay inside $[0.2, 0.45]$) compared to single event scenarios ($NRPT$s stay inside $[0.04, 0.16]$). This is caused by the fact that the processor has to handle more workload in multi-event scenarios and thence generates more heat. (3) As shown in Fig. 2.13, the peak temperature decreases as the relative deadline increases, since the processor can stay at sleep mode longer for each mode switch. The peak temperature however will not further decrease after certain threshold is reached. (4) The minimal peak temperature in CS is generally the lowest in single event stream scenarios as CS works online and can dynamically turn off the processor according to actual workload. It's worth noting that since the heat generated by online calculating and monitoring of CS is not considered in our simulation, the peak temperature in CS will be higher when it comes to practical application. Moreover, CS approach also has to pay high penalty of the offline computation time while PMPT and AMPT approaches can achieve similar effect with much lower computation expense, which we will show later. In multi streams scenarios, however, CS yields higher peak temperature than PMPT and AMPT, which is resulted from the approximation of input arrival curves. We have made better approximations to improve the results but with trivial feedback. (5) By and large, the peak temperature derived by TAPR is the highest. The reason is the limitation of its event model where the non-determinism of pjd pattern cannot be properly modeled and the modified $P = \max[(p - j), d]$ overestimates the incoming workload. As shown in Fig. 2.10, there exists an extraordinary point, which is the $NRPT$ of task S_4. The reason is that S_4 has the largest jitter j and the second smallest minimal inter-arrival distance d, which exacerbates the effect of the event model unsuitableness. Consequently, we can see that the peak temperature generated by TAPR reaches the upper bound in the multi-event cases as long as S_4 is involved in input streams.

We also report the timing overhead of deriving a PTM scheme. Since our PTM approaches are offline computed and need negligible runtime overhead, only the offline computing part of CS is taken into account. We adopt the computation time for finding the optimal W_{unit}, which is the critical parameter for CS, as the computational overhead of CS. Fig. 2.14 shows the computation expense of the four approaches for ten sets of randomly selected four-event streams and Fig. 2.15 demonstrates

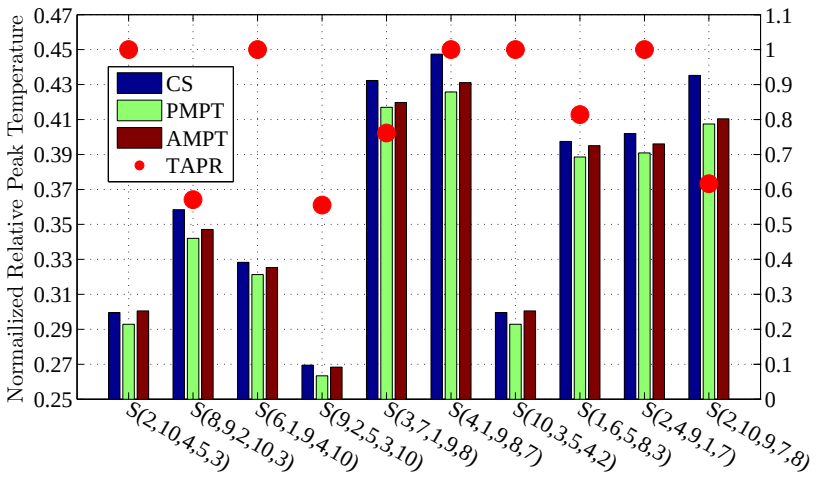


Figure 2.12: Normalized Relative Peak Temperature produced by the tested approaches for ten sets of randomly selected five-events stream scenarios with $\chi = 1$ by applying EDF scheduling. The right Y axis indicates the NRPT of approach TAPR.

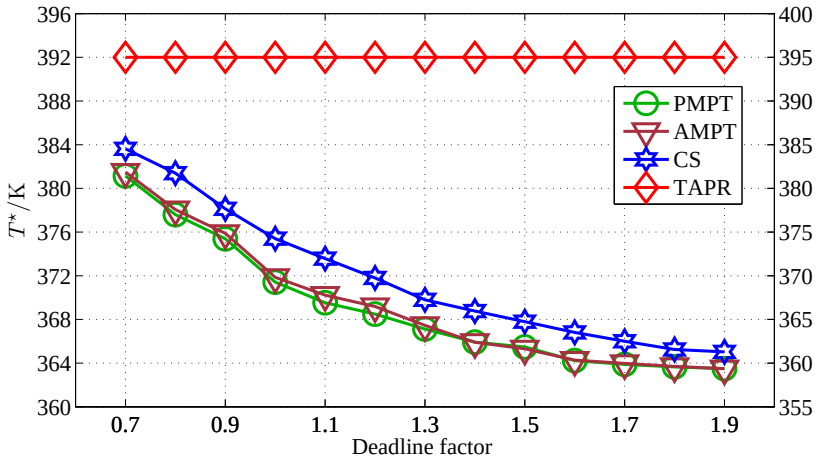


Figure 2.13: Peak Temperature generated by the tested approaches w.r.t. different relative deadlines for ten-events stream scenario with EDF scheduling.

how the computation expense in ten-event stream scenario varies as the relative deadline factor changes. We make below observations: (1) The computation overhead of cool shaper is the highest, which is about one up to four orders of magnitude larger than that of our PTM approaches. (2) In the second figure, the computation overhead of PMPT increases

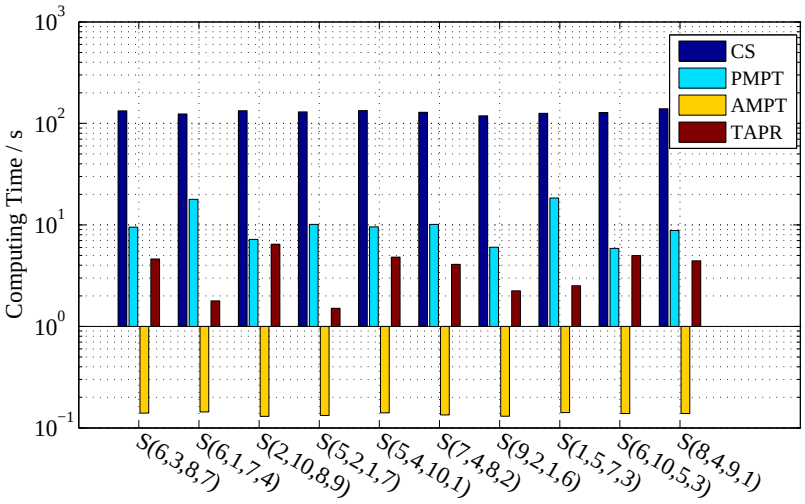


Figure 2.14: Computing time of the tested approaches for randomly selected four-events stream scenarios with $\chi = 1$ by applying EDF scheduling.

w.r.t. the relative deadline. The reason is that the number of the tested points of t_{off} and t_{on} increases as the relative deadline increases when ε and ξ are fixed. (3) The time consumptions of AMPT are always the lowest and stable, which are within half a second. (4) Compared to PMPT, the timing overhead of AMPT is about one or two orders of magnitude lower. In conclusion, our PTM algorithms are much faster in terms of computation overhead but generate peak temperatures close to or even better than the ones of CS online approach.

2.8 Summary

In this section, we present Periodic Thermal Management to minimize the peak naïve temperature for a single-core hard real-time system in which the input event streams are characterized by arrival curves. The temperature of the system is controlled by applying dynamic power management techniques. The proposed PTM approach periodically switches the system to low power-consumption state according to pre-computed scheme. With the worst case deadline constraint, we propose one algorithm that can provide precise solutions and one approach to yield approximated solutions with lower computation time. To verify the effectiveness and efficiency, we present several implementations of our approaches with single event and multi-event streams. Experimental re-

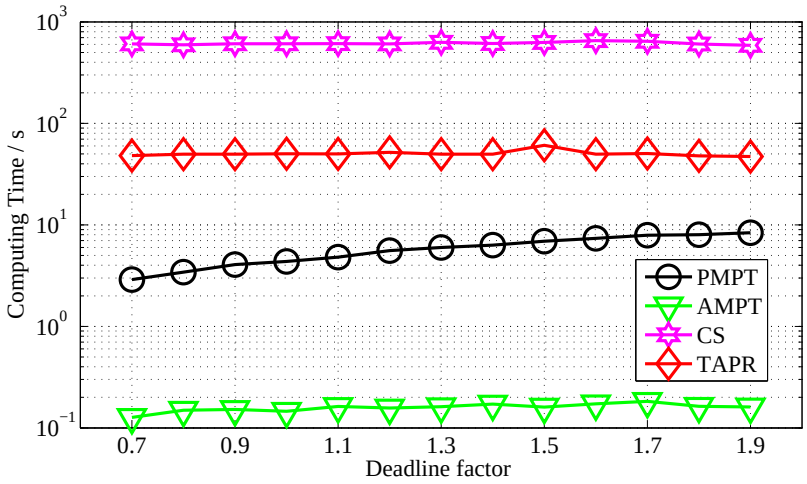


Figure 2.15: Computing time of the tested approaches w.r.t. different relative deadlines for ten-events stream scenario with EDF scheduling.

sults show that our algorithms can derive periodic thermal management schemes with negligible runtime overheads while the peak temperature can be constrained to similar or even better level of online approach in the literature.

Our PTM scheme requires negligible runtime overhead since the scheme is pre-computed in design phase, making itself suitable for real-time system having little computation power. Moreover, PTM can be easily implemented on a processor by simply using a hardware timer. In next chapter, we will investigate how to implement our PTM to pipelined multi-core systems by reversely using the Pay Burst Only Once principle.

Chapter 3

Pipelined System Thermal Management

With the ever-increasing demand of computational performance, multi-core architecture is now widely adopted by major chip manufacturers. To date, processors having 64 or more cores are available in the market. The architecture with such a high degree of parallelism poses designers a challenge: how to extract and exploit parallelism from applications efficiently.

Pipelined computing, which can increase the throughput of a stream application, is a promising paradigm for real-time systems. The pipelined computing model connects a set of processing units in series, where the output of one unit is the input of the next one, and executes the sub-tasks of the stream application. By this way, the sub-tasks can be executed simultaneously, that is, parallel processing is performed. Therefore, pipelined computing can exploit the hardware performance advantage of multi-core processor efficiently and increase the throughput of the application.

For real-time pipelined systems, especially for hard real-time systems, ensuring the latency bounded by a specific constraint is crucial for the the system correctness. However, as power density is increasing exponentially under Moore's Law, the peak temperature on modern processors is rapidly elevated, which seriously threats the reliability and performance of the system. It is studied that a $10 - 15$°C difference in operating temperature can result in a $2\times$ difference in the lifespan of a device [21]. Since reducing the temperature usually requires less power consumption, which means lower performance and larger latency, the

trade-off between real-time performance and temperature constraints should be carefully analyzed. Therefore, it's an important and challenging task to design a scheduling policy for a pipelined real-time system on a multi-core processor such that the peak temperature is minimized and the end-to-end deadline constraint is satisfied.

3.1 Overview

This chapter focuses on the aforementioned issue and addresses the optimization problem by reversely using the Pay-Burst-Only-Once (PBOO) principle. Our work is inspired by the work of Chen et al. [23], which minimizes the total power consumption for pipelined stage systems. However, their work cannot be directly transplanted to temperature optimization, due to the reasons: (1) although temperature is a strong function of power, power management techniques that are effective for energy saving may not be suitable for temperature managing [111], which has already been theoretically proved by [7]. (2) The quadratic programming formulation of the power problem cannot be reused, since the peak temperature is calculated based on convolution operation while energy consumption is computed based on integral operation. Therefore, the problem of temperature minimization demands new analysis and optimization techniques.

We consider a multi-core processor which tackles applications which can be divided into sub-tasks. The sub-tasks can be mapped and executed on different cores which communicate with each other via FIFOs (First-In-First-Out). Every core has two power consumption states, 'active' and 'sleep'. To model general event arrivals, the concept of arrival curve [96, 60, 98] is adopted as the input task application model. The leakage power dependence on temperature is considered and simplified by a precise linear approximation [42]. We adopt the well known HotSpot thermal model and the RC thermal network to model our system as a Linear Time Invariant (LTI) system. The power gating technology is employed to control the temperature and PTM investigated in previous chapter is adopted to minimize the peak temperature. A comprehensive analysis on the peak temperature of the processor under PTM is presented. Based on the analysis results, two algorithms are proposed to calculate the peak temperature in different levels of accuracy and speed. The optimization problem of searching the PTM schemes is transformed into a set of sub-problems which are easier to solve. Two algorithms are proposed to solve the sub-problem for different peak temperature calcu-

lation methods. One algorithm is an approximated one and based on the gradient descend method. The other one is based on the simulated annealing algorithm and offers more accurate results. We investigate the effectiveness and efficiency of our approach by implementing on two real life platforms: a homogeneous ARM multi-processor and the Intel Single-chip Cloud Computer (SCC). The scalability of our approach is demonstrated by testing our approach in systems with up to 24 cores.

The rest of this chapter is organized as follows: Section 3.2 gives a brief introduction of related work. Our system models are introduced in Section 3.3 and Section 3.5 shows a motivation example and presents the problem statement. We analyze the peak temperature and give algorithms to calculate it in Section 3.6 and Section 3.6.2, respectively. Section 3.7 discusses real-time analysis and formalizes our optimization problem which is transformed into a set of sub-problems. Section 3.8 presents algorithms to solve the sub-problem. Section 3.9 details the case studies and Section 3.10 concludes this chapter.

3.2 Related work

In this section, we review previous work on thermal-aware system scheduling policies for pipelined systems and multi-core processors.

At first, we briefly introduce the important work on studying the mechanism of thermal management and thermal modelling. Brooks and Martonosi [15] introduced the dynamic thermal management (DTM) and presented policies and mechanisms for implementing DTM for current and future CPUs. To accurately evaluate the thermal profile for the micro-architectures, Skadron et al. [93, 52] proposed a compact architectural-level thermal-modeling methodology, named HotSpot. Hotspot enables designers to consider the thermal impacts on interconnects during early design stages as it includes a high-level on-chip interconnect self-heating power and thermal model. They also developed an homonymic toolbox which can set up the thermal model based on the input processor floorplan and parameters.

Now, the related work is categorized based on if it is intended for pipelined computing

Pipelined Computing Chen and et al. [23] utilized PBOO principle and presented an approach to optimize the power consumption of a pipelined system under the deadline constraint. A quadratic program-

ming formulation of the problem is proposed and two methods are studied to solve the problem. As stated in Section 3.1, a power optimal approach may not be thermal optimal. Therefore, a new approach is needed for the problem of peak temperature minimization. There exist several thermal management approaches for pipelined systems. However, previous work on this topic either considers no hard deadline constraints [3, 32] or just reduces the deadline misses percentage into a lower range [72]. Our approach can give the hard real-time guarantee by ensuring the end-to-end delay is under the hard deadline constraint.

Multi-core Processors There has been significant work on thermal management for multi-core processors and we inspect the work that is closely related to our topic. Aiming to minimize the chip peak temperature while satisfy the hard real-time constraints of an MPSOC (Multiprocessor System-on-Chip), Thidapat and et al. [21] addressed the problem of assigning and scheduling tasks on the MPSOC. They presented a mixed-integer linear programming (MILP) formulation of the problem and then gave an optimal solution as well as a flexible heuristic framework for the MILP formulation. Due to the thermal analysis difficulties, this approach examines only steady-state temperatures without considering the transient behavior. In this chapter, we provide a peak temperature formulation which considers the transient temperature. Jungseob and Nam [63, 64] studied how to optimize and improve the throughput of a power- and thermal-constrained multi-core processor. Their research didn't provide any hard real-time guarantee, therefore cannot be applied to hard real-time systems directly. We consider the task deadline constraints in this chapter and ensure that the peak temperature is minimized under the constraints. Yong and et al. [39] presented a feedback thermal control framework named Real-Time Multicore Thermal Control which dynamically enforces both the desired temperature and the CPU utilization bounds for multicore real-time systems, through DVFS. Buyoung [110] addressed the problem of avoiding thermal hotspot on a multi-core chip by employing a runtime thermal aware scheduler (TAS) using job-migration and power-gating techniques. In [45], Pradeep extended the concept of Thermal-Resiliency to multi-core architecture and then adopted a control-theoretic framework to ensure hard-real-time deadlines in a dynamic thermal environment while maintaining the thermal constraints. However, to simplify the complexity of timing analysis, above work all assumed simple task models, i.e., either periodic or sporadic task model. In this chapter, the task streams are modeled by a more general concept, the arrival curve, therefore we can preserve more

information such as the non-determinism of the event arrivals in the model.

In summary, compared to related work, our work achieves the following improvements:

- providing hard real-time guarantee. Compared to: [34], [72] , [64], and [63]
- handling non-deterministic event arrivals. Compared to: [38], [100], [32], [45], [44], [110], and [39]
- considering leakage power dependency on temperature. Compared to: [64], [34], and [63]
- considering transient thermal behavior. Compared to: [21], and [32]
- thermal optimization. Compared to: [23], [19], and [53]

3.3 system model

Notation: In this chapter, all matrices and vectors are denoted by bold characters.

Definition 3.1 *For two $m \times n$ matrices $\mathbf{A}$ and $\mathbf{B}$, we consider $\mathbf{A} \geq \mathbf{B}$ if $\mathbf{A}_{ij} \geq \mathbf{B}_{ij}$ holds for all $0 \leq i \leq m$ and $0 \leq j \leq n$.*

3.3.1 Hardware Model

In this chapter, a multi-core processor which can handle partitioned applications is considered. The sub-tasks of a partitioned application can be mapped and executed on different cores which communicate with each other via FIFOs (First-In-First-Out). An example of pipelined architecture can be found in Fig. 3.1. Without loss of generality, we denote the stage or the core number as n. Each core has two power dissipation modes, namely 'active' and 'sleep' mode. In 'active' mode, the cores work with higher power consumption and tackles input events in a fixed frequency. The cores can be switched to 'sleep' state for a lower power consumption when there is no workload to handle. We also consider the mode-switching overhead. To switch the core i from 'active' mode to 'sleep' mode and back, t_i^{swoff} and t_i^{swon} time units are required, respectively. During mode-switching, the power consumption equals that in 'active' mode. Moreover, no coming event can be

handled in mode-switching or 'sleep' mode. Due to time overhead in mode-switching, the time length for which a core is switched to 'active' ('sleep') mode must be larger than t^{swon} (t^{swoff}), that is, $t_i^{off} > t_i^{swoff}$ and $t_i^{on} > t_i^{swon}$. For brevity, we define $\mathbf{t^{swoff}} = (t_1^{swoff}, t_2^{swoff}, \cdots, t_n^{swoff})$ and $\mathbf{t^{swon}} = (t_1^{swon}, t_2^{swon}, \cdots, t_n^{swon})$ for an n-core processor.

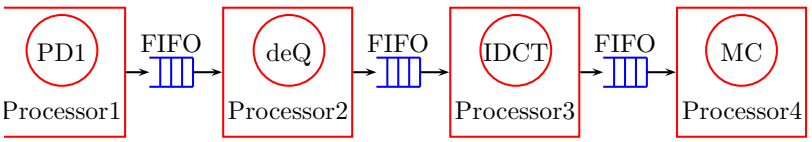


Figure 3.1: H.263 decoder on pipelined hardware architecture.

3.3.2 Application Model

We study the streaming applications which can be split into several sub-tasks. Then, the worst-case execution times (WCET) of the sub-tasks on all stages are represented by $\mathbf{c} = \{c_1, c_2, \cdots, c_n\}$, where c_i denotes the WCET of the sub-task executed on ith stage. The end-to-end relative deadline of the application is denoted as D.

To model general task arrivals and available resources in any time interval Δ, the event-based arrival curve $[\alpha^u(\Delta), \alpha^l(\Delta)]$ and the time-based service curve $[\beta^u(\Delta), \beta^l(\Delta)]$ investigated in Chapter 2 are adopted. Note that instead of transforming an event-based arrival curve to a time-based one, which is adopted in Chapter 2, we transform the time-based service curve to event-based formation, since the smallest granularity of the application stream between two stages is one event. Similarly, The transformation can be done as $\bar{\beta}^u(\Delta) = \lfloor \beta^u(\Delta)/c \rfloor$ and $\bar{\beta}^l(\Delta) = \lceil \beta^l(\Delta)/c \rceil$.

3.3.3 Thermal Model

The well established thermal model HotSpot is employed to model the multi-core processor [52]. The vertical layout of the processor is modeled by four layers, which are the heat sink, heat spreader, thermal interface, and silicon die layers. Each layer is divided into a number of blocks according to the processing components on the die. The total number of the thermal blocks of a processor with n components is $N = 4n + 12$, including the extra eight and four blocks on the heat sink and heat spreader layers, respectively. Moreover, the thermal blocks are

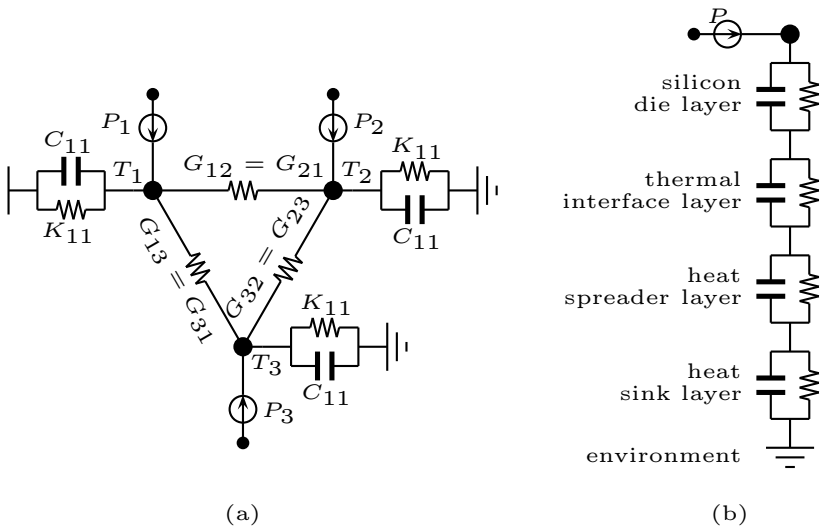


Figure 3.2: (a) The equivalent RC circuit of the silicon layer for a floorplan with three processing components, and (b) The vertical layout model.

ordered in a way that all the processing components occupy the beginning part of the order list. To predict the temperature evolution, we take the advantage of the well-known electro-thermal analogy, i.e., the RC thermal network. Every thermal block is mapped onto a node of the thermal circuit. An example of the model can be found in Fig. 3.2. Finally, the temperature vector $\mathbf{T}(t)$ can be determined by a set of first-order differential equations [97]:

$$\mathbf{C} \cdot \frac{\mathrm{d}\mathbf{T}(t)}{\mathrm{d}t} = (\mathbf{P}(t) + \mathbf{K} \cdot \mathbf{T^{amb}}) - (\mathbf{G} + \mathbf{K}) \cdot \mathbf{T}(t) \tag{3.1}$$

where $\mathbf{C}$ is the thermal capacitance matrix, $\mathbf{P}$ is the power dissipation vector, $\mathbf{K}$ is the thermal ground conductance matrix, $\mathbf{G}$ is the thermal conductance matrix and $\mathbf{T^{amb}}$ is the ambient temperature vector which is defined as $\mathbf{T^{amb}} = T^{amb} \cdot [1, \cdots, 1]'$, where T^{amb} is the ambient temperature.

We assume the total power consumption $\mathbf{P}$ is the sum of the power $\mathbf{P^d}$ due to dynamic current and the power $\mathbf{P^l}$ due to leakage current [43, 81]. Dynamic power $\mathbf{P^d}$ is assumed to be constant in each power mode, i.e., $\mathbf{P^a}$ and $\mathbf{P^s}$, in the 'active' and 'sleep' mode, respectively. The dependency relationship between the leakage power and the temperature can be closely approximated by a linear function of the processor tempera-

ture, which has been widely adopted [42, 97, 43, 68, 86]:

$$\mathbf{P^l} = \mathbf{W} \cdot \mathbf{T}(t) + \mathbf{V} \tag{3.2}$$

where $\mathbf{W}$ is a diagonal matrix with constant coefficients and $\mathbf{V}$ is a vector with constant coefficients. Therefore, $\mathbf{P}$ can be represented as:

$$\mathbf{P}(t) = \begin{cases} \mathbf{W} \cdot \mathbf{T}(t) + \mathbf{V^a} & \text{if in active mode} \\ \mathbf{W} \cdot \mathbf{T}(t) + \mathbf{vs.} & \text{if in sleep mode} \end{cases} \tag{3.3}$$

where $\mathbf{V^a} = \mathbf{V} + \mathbf{P^a}$ and $\mathbf{vs.} = \mathbf{V} + \mathbf{P^s}$. Rewriting (3.1) with the power formulation (3.3), we can obtain the state space representation of the thermal model:

$$\frac{\mathrm{d}\mathbf{T}(t)}{\mathrm{d}t} = \mathbf{A} \cdot \mathbf{T}(t) + \mathbf{B} \cdot \mathbf{u}(t) \tag{3.4}$$

where $\mathbf{A} = -\mathbf{C}^{-1} \cdot (\mathbf{G} + \mathbf{K} - \mathbf{W})$, $\mathbf{B} = \mathbf{C}^{-1}$, and $\mathbf{u}(t)$ is the input vector. If component j is in active mode or mode-switching, $u_j(t) = V_j^a + K_{jj} \cdot T^{amb}$, otherwise $u_j(t) = vs._j + K_{jj} \cdot T^{amb}$. Since $\mathbf{A}$ and $\mathbf{B}$ are constant, the thermal model is a first order linear time invariant system and the closed-form representation of the temperature is:

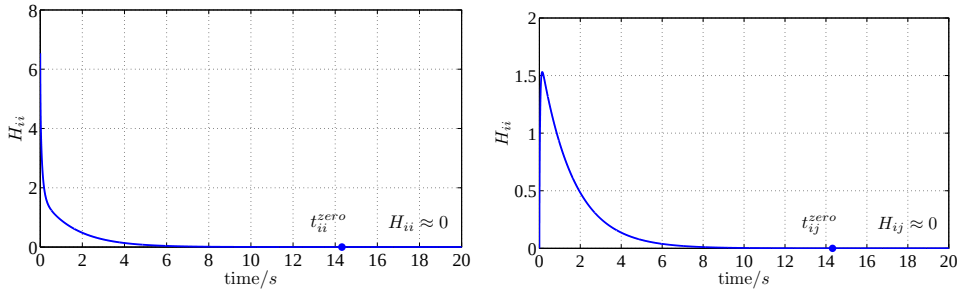
$$\mathbf{T}(t) = e^{\mathbf{A} \cdot t} \cdot \mathbf{T}^0 + \int_{-\infty}^{\infty} \mathbf{H}(\xi) \cdot \mathbf{u}(t - \xi) \mathrm{d}\xi \tag{3.5}$$

where $\mathbf{T}^0$ is the initial temperature vector and $\mathbf{H}(t) = e^{\mathbf{A} \cdot t} \cdot \mathbf{B}$ is the matrix describing the impulse response between any two nodes in the thermal model. Fig. 3.3 shows two examples of the impulse response between two nodes. The self-impulse response $H_{ii}(t)$ is a non-negative decreasing function, as shown in Fig. 3.3. Regarding $H_{ij}(t)$ where $i \neq j$, we adopt the conjecture proposed in [97] that $H_{ij}(t)$ is a non-negative unimodal function [1]. Since our thermal system is a physical system, it is straightforward that $H_{ij}(t) = u_i(t) = 0$ when $t \leq 0$. Denoting $\mathbf{T^{init}}(t) = e^{\mathbf{A} \cdot t} \cdot \mathbf{T}^0$, the temperature of node i yields:

$$T_i(t) = T_i^{init}(t) + \sum_{j=1}^{N} T_{ij}^{conv}(t). \tag{3.6}$$

where $T_{ij}^{conv}(t)$ is the convolution between $H_{ij}(t)$ and $u_j(t)$,

$$T_{ij}^{conv}(t) = H_{ij}(t) \otimes u_j(t) = \int_0^t H_{ij}(\xi) \cdot u_j(t - \xi) \mathrm{d}\xi \tag{3.7}$$



(a) (b)

Figure 3.3: The impulse response from node j to node i, when (a) $i = j = 1$, (b) $i = 1, j = 2$. The thermal model comes from the three-core ARM model in [97] and the t^{zero} points are determined when H_{ii} or H_{ij} is less than 2×10^{-4}.

Due to the temporal and spatial variation in temperature, calculating the peak temperature presents significant difficulties. After closely examining the model, we make two observations that scale down the exploring space in temporal and spatial dimensions:

1. The outputs of the thermal system, i.e., the temperatures of all the nodes, are bounded by the active state steady temperatures, which indicates our system is BIBO (bounded-input, bounded-output) stable [109, 40]. An LTI system is said to be BIBO stable if it produces a bounded-output:

$$|y(t)| < V_1 < \infty \tag{3.8}$$

when the system has a bounded-input:

$$|x(t)| < V_2 < \infty \tag{3.9}$$

where V_1 and V_2 are constants. It's also proven in [40] that BIBO stability is assured if and only if $\int_{-\infty}^{\infty} |H(t)|dt < \infty$. Since $H_{ij}(t)$ is a non-negative function, one can prove that $H_{ij}(t)$ will finally equals zero as t increases. An intuition explanation is shown in Fig. 3.3, where $H_{ij}(t) \approx 0$ after a certain time instance t_{ij}^{zero}, which is determined by system properties as well as the accuracy tolerance. Besides, as $\mathbf{T^{init}}(t) = \mathbf{H}(t) \cdot \mathbf{C} \cdot \mathbf{T}^0$, one can easily prove that $T_i^{init}(t)$

[1] In this chapter, function $f(x)$ is a unimodal function if for some value m, it is non-increasing for $x \leq m$ and non-decreasing for $x \geq m$.

has the same property, too. Therefore, in the following of this chapter, we adopt $H_{ij}(t) = 0$ and $T_i^{init}(t) = 0$ when t is sufficiently large.

2. The peak temperature can only occur on the processing component nodes. The intuition is that according to Second law of thermodynamics, in our thermal model, heat can only flow from a hotter node to a colder one. Since heat are generated from the processing components, the temperature on them will be higher. Therefore, to get the peak temperature of our system, only the temperature of processing component nodes should be calculated.

Moreover, the thermal model is regulated by following circumstances.

- The ambient temperature vector $\mathbf{T^{amb}}$ is assumed to be constant.
- Denote the sleep state steady temperature as $\mathbf{T^{sleep}}$, we let the initial temperature $\mathbf{T}^0 = \mathbf{T^{sleep}}$.

With the system described above, we show a motivation example and the problem statement in next section.

3.4 Real-Time Calculus Background

In this section, we introduce the basic definitions and theorems adopted in real-time calculus which are needed in the following of this chapter. We simply present the results that are useful and omit the detailed mathematical derivation. The details of results below can be found in [60].

3.4.1 Wide Sense Increasing Functions

A function $f(t)$ is wide-sense increasing if and only if $f(s) \leq f(t)$ for all $s \leq t$. We denote by $\mathcal{F}$ the set of wide-sense increasing functions such that $f(t) = 0$ for $t \leq 0$. The parameter t is a real number and is continuous. By convention, we consider the function $f(t)$ is left-continuous. An example of function belonging to $\mathcal{F}$ is listed below.

Definition 3.2 (Affine functions) *An affine function is defined by $r \geq 0$ (the 'rate') and $b \geq 0$ (the 'burst'), which can be given as:*

$$\gamma_{r,b}(t) = \begin{cases} rt + b & \text{if } t > 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.10}$$

Definition 3.3 (Rate-latency functions) *A rate-latency function is defined by $R \geq 0$ (the 'rate') and $L \geq 0$ (the 'latency'), which can be given as:*

$$\beta_{R,L}(t) = \begin{cases} R(t-L) & \text{if } t > L \\ 0 & \text{otherwise} \end{cases} \tag{3.11}$$

3.4.2 Basic Mathematical Results

Firstly, we recall the concept of infimum and supremum. Let $\mathcal{S}$ be a nonempty subset of $\mathbb{R}$. If there exist a number M such that $s \geq M$ for all $s \in \mathcal{S}$, $\mathcal{S}$ is termed as a lower bounded set. The completeness axiom states that every nonempty subset $\mathcal{S}$ of $\mathbb{R}$ that is bounded from below has a greatest lower bound. Then, the infimum of $\mathcal{S}$ is defined as this greatest lower bound and is denoted by $\inf \mathcal{S}$. Similarly, the supremum of of nonempty subset $\mathcal{S}$ of $\mathbb{R}$ that is bounded from up is the lowest upper bound can is denoted by $\sup \mathcal{S}$.

Now, we introduce the basic operations defined in Real-Time Calculus.

Definition 3.4 (Min-Plus Convolution) *Let $f(t)$ and $g(t)$ be two functions belonging to $\mathcal{F}$, then the min-plus convolution of $f(t)$ and $g(t)$ is defined as:*

$$(f \otimes g)(t) = \inf_{0 \leq s \leq t} \{f(t-s) + g(s)\}. \tag{3.12}$$

Moreover, if $t < 0$, $(f \otimes g)(t) = 0$.

Definition 3.5 (Min-plus Deconvolution) *Let $f(t)$ and $g(t)$ be two functions belonging to $\mathcal{F}$, then the min-plus deconvolution of $f(t)$ and $g(t)$ is defined as:*

$$(f \oslash g)(t) = \sup_{u \geq 0} \{f(t+u) - g(u)\}. \tag{3.13}$$

Theorem 3.6 (Output stream) *Assume an event stream, constrained by arrival curve $\alpha(\Delta)$, traverses a system offering service curve $\beta(\Delta)$, then the output stream is constrained by the arrival curve $\hat{\alpha}(\Delta) = \alpha \oslash \beta$.*

3.4.3 Pay Burst Only Once

Suppose an event flow with arrival curve α traverses two computation system $\mathcal{S}_1$ and $\mathcal{S}_2$ in sequence. To obtain the end-to-end delay bound on the systems, a straightforward method is calculating the isolated delay bounds in each system and then summing them up. However, the following theorem offers another way to calculate the bound.

Theorem 3.7 (Concatenation of Nodes) *Assume a workload flow traverses system $\mathcal{S}_1$ and $\mathcal{S}_2$ in sequence. Assume that $\mathcal{S}_i$ offers a service curve of β_i, $i = 1, 2$ to the flow. Then the concatenation of the two systems offers a service curve of $\beta_1 \otimes \beta_2$ to the flow, where $\otimes$ is the min-plus convolution.*

With Thm. 3.7, the phenomenon of Pay Burst Only Once is illustrated by an example.

Example 3.8 *Consider an event stream constrained by affine arrival curve $\gamma_{r,b}$ is handled by the concatenation of two nodes each of which offers a rate-latency service curve $\beta_{R_i,L_i}(\Delta)$, $i = 1, 2$. We assume $r < R_1$ and $r < R_2$. Now we compare the delay bounds calculated by two methods described below.*

1. *the method applying Thm. 3.7.*
2. *the aforementioned sum-all method.*

From Thm. 3.7, the aggregate service curve is $\beta^{agg} = \beta_{R_0,L_0}$, where $R_0 = \min_i(R_i)$ and $L_0 = \sum_i^{L_i}$. Then from Thm. 2.3, we have the delay bound:

$$D_0 = \frac{b}{R_0} + L_0 \tag{3.14}$$

Now the second method is applied. The delay bound at first node is:

$$D_1 = \frac{b}{R_1} + L_1 \tag{3.15}$$

From Thm. 3.6, the event stream to the second node , i.e., the output of the first node, is constrained by arrival curve $\alpha_2 = b + r \times (t + L_1)$. Then the delay bound at second node is:

$$D_1 = \frac{b + rL_1}{R_2} + L_2 \tag{3.16}$$

*One can easily derive that $D_0 < D_1 + D_2$, that is, the method considering the aggregate service curve offers better delay bound than the method considering the service curve of each node in isolation does. The reason is the second method considers the burst b in the original event stream more than once, i.e, twice, while the former method consider the **burst only once**, which is termed as **pay burst only once**.*

3.5 Motivation and Problem statement

With Thm. 3.7 and Pay Burst Only Once principle, a pipelined system composed of n stages each of which offers service curve β_i can be considered as a black-box system which offers an aggregate service curve of $\beta = \beta_1 \otimes \beta_2, \cdots, \beta_n$. Then, the end-to-end delay bound can be calculated sequentially. The Pay-Burst-Only-Once principle points out that the result derived from Thm. 3.7 better bounds the real end-to-end delay than the straightforward method does.

In this chapter, our goal is to calculate the lower service curve with given end-to-end deadline instead of to obtain the end-to-end delay with given service curve. Therefore, Thm. 3.7 and PBOO principle can be used reversely to get a better approximated lower service curve to the actual service curve demanded for the given end-to-end deadline. Since better approximation indicates lower service curve, lower peak temperature for the system is expected, which is the motivation of our work.

In our work, the bounded delay function is widely used to simply the analysis and thus is worth introducing.

Definition 3.9 (Bounded delay function) *A Bounded delay function (*BDF*) specified by constants ρ and b is defined as:*

$$bdf(\Delta) = \max[0, \rho(\Delta - b)]. \tag{3.17}$$

3.5.1 Motivation Example

In this chapter, we deploy Periodic Thermal Management to manage the temperature of the chip by periodically switching each stage between two power consumption modes with an individual pair of (t^{on}, t^{off}). Therefore, two vectors, $\mathbf{t^{off}} = (t_1^{off}, t_2^{off}, \cdots, t_n^{off})$ and $\mathbf{t^{on}} = (t_1^{on}, t_2^{on}, \cdots, t_n^{on})$, should be determined offline to specify the PTMs deployed on the system. For the details of PTM, we refer to [1]. Now, we present a motivation example to illustrate the advantages of applying Pay-Burst-Only-Once for thermal optimization. For comparison, the PTM schemes are derived from two approaches: the PBOO based approach (PBOO) and the one which partitions the end-to-end deadline into sub-stage deadlines for each stage, namely SDP (Sub-Deadline Partition).

In this example, an event stream with arrival curve $\alpha = 0.15\Delta + 2$ and deadline $D = 35ms$ passes through a two-stage pipelined system. For simplicity, the worst-case execution times of the event stream in first and

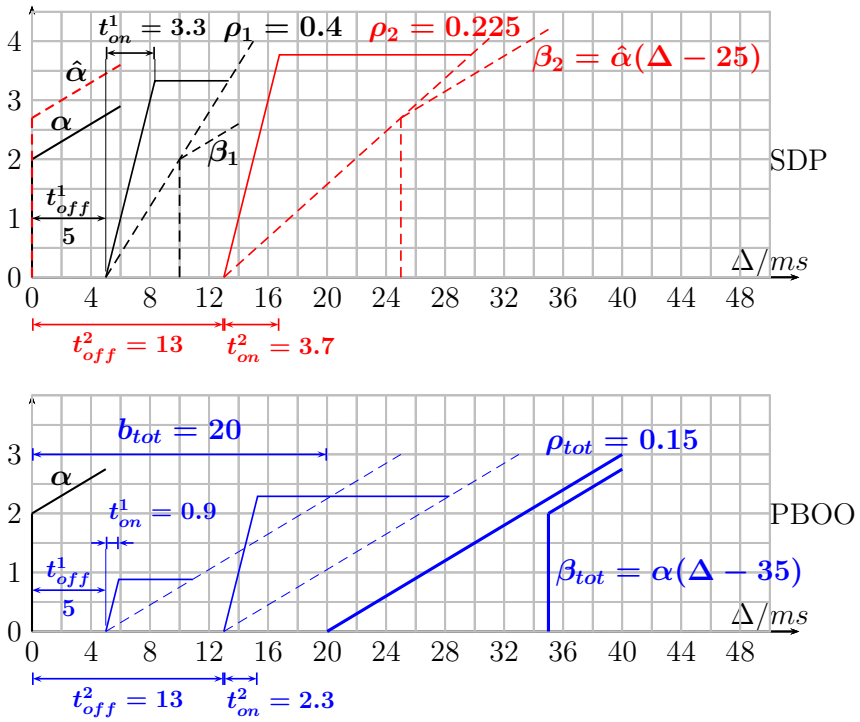


Figure 3.4: An example of calculating PTM schemes by methods SDP and PBOO.

second stage are set as $c_1 = c_2 = 1ms$, respectively. We discuss the case that $\mathbf{t^{off}} = (5, 13)ms$ and compare $\mathbf{t^{on}}$s generated by the two methods. Fig. 3.4 graphically illustrates the derivation process corresponding to the two methods.

Let's first examine the strategy of Sub-Deadline Partition. We divide the deadline D into two sub-deadlines, $D_1 = 10ms$ and $D_2 = 25ms$, in this case.

To simplify the process of calculating $\mathbf{t^{on}}$, we adopt the bounded-delay function as an assistant service curve. For instance, for the first stage, with the deadline $D_1 = 10ms$, the service demand should be $\beta_1 = \alpha(\Delta - 10)$. Denote the service curve provided by first state as β_1^{tdma}, which is is a TDMA curve specified by t_1^{off} and t_1^{on}. We can guarantee $\beta_1^{tdma} \geq \beta_1$, if $\beta_1^{tdma} \geq bdf_1 \geq \beta_1$. Since $t_1^{off} = 5ms$, the b in bdf_1 is set as $b_1 = t_1^{off} = 5ms$. Then, $bdf_1 \geq \beta_1$ yields the minimal slope of bdf_1 as $\rho_1 = (2 - 0)/(10 - 5) = 0.4$ and finally we have $t_1^{on} = 3.3ms$, which is

derived from $\frac{t_1^{on}}{t_1^{on}+t_1^{off}} = c_1 * \rho_1 = 0.4$ to ensure $\beta_1^{tdma} \geq bdf_1$. To derive the service demand for the second stage, the output arrive curve $\hat{\alpha}$ from the first stage is needed. From Thm. 3.6,

$$\hat{\alpha} = \alpha \oslash \beta_1^{tdma} = 0.15\Delta + 2.7 \tag{3.18}$$

In the same way, we have the slope of the BDF in second stage as $\rho_2 = 0.225$ and finally $t_2^{on} = 3.7ms$.

Now, the PBOO method is utilized to get $\mathbf{t^{on}}$. Unlike the procedure in SDP, the total service demand is first obtained as $\beta_{tot} = \alpha(\Delta - 35)$ and then a BDF $bdf_{tot} = \max[0, \rho(\Delta - b)]$ should be determined such that the deadline constraint can be met as long as $\beta_1^{srv} \otimes \beta_2^{srv} \geq bdf_{tot} \geq \alpha^u(\Delta - D)$. From the analysis discussed in Section 3.7, we can set b_{tot} as its minimum: $b_{tot} = t_1^{off} + t_2^{off} + c_1 + c_2 = 20ms$, then the slope of bdf_{tot} can be determined: $\rho_{tot} = 0.15$, which is much smaller than ρ_1 and ρ_2, therefore, resulting in smaller $t_1^{on} = 0.9ms$ and $t_2^{on} = 2.3ms$.

The pessimism in the SDP method comes from paying an additional burst and delay when $\hat{\alpha}$ is calculated for the second stage, as PBOO principle points out [60]. Moreover, as the stage number increases, this effect is accumulated and then causes more pessimistic results. On the other hand, method PBOO directly calculates the total service demand and then retrieves $\mathbf{t^{on}}$ for every stage, which pays the burst only once and gets better results. Since lower partition of t^{on} means lower temperature of the processor, it is expected that employing PBOO will achieve lower peak temperature than using SDP. Therefore, by reversely using PBOO principle, we can avoid paying the burst repeatedly and therefore better optimize the peak temperature for pipelined systems, especially for which having large number of stages.

3.5.2 Problem Statement

Now our problem is defined as follows:
Given an n-stage pipelined platform specified by the above hardware and thermal models, an event stream with end-to-end deadline D, and the WCET*s* $\mathbf{c} = (c_1, c_2, \cdots, c_n)$*, our goal is to find the* PTM *schemes characterized by* $\mathbf{t^{off}}$ *and* $\mathbf{t^{on}}$ *such that the peak temperature is minimized while the deadline constraint is satisfied.*

To deal with this problem, the first question is how to get the peak temperature with known Periodic Thermal Management, which is discussed in next section.

3.6 Calculating Peak Temperature

In this section, we discuss how to get the peak temperature of a multi-core system which adopts PTM. It is straightforward that one can get the peak temperature by simulating a sufficient long trace of the temperature with the help of any thermal simulation toolbox, for example, the HotSpot toolbox. However, this method doesn't utilize the feature of PTM schemes at all and thus is ineffective for our optimization problem. Therefore, the peak temperature for PTM schemes is worth discussing such that high efficiency is achieved. Firstly, we analyze how the temperatures of the nodes evolve under PTM and present a formulation of the peak temperature. Then, based on the formulation, two algorithms are proposed to calculate the peak temperature of our system with different levels of accuracy and speed.

3.6.1 Peak Temperature Analysis

Based on the first observation in the thermal model, some notations are first defined. Let (1) t_{ij}^{zero} and t_k^{init} denote the certain time point after which $H_{ij}(t)$ and $T_k^{init}(t)$ can be approximated as zero, respectively, and (2) n and N indicate the numbers of total processing components and thermal blocks in thermal model, respectively. Now some basic lemmas are shown for further analysis.

Lemma 3.10 *For any $0 \leq i, j \leq N$ and $t \geq t_{ij}^{zero}$, $T_{ij}^{conv}(t)$ is either a periodic function in the domain of t, if $j \leq n$, or a constant function, if $n < j \leq N$.*

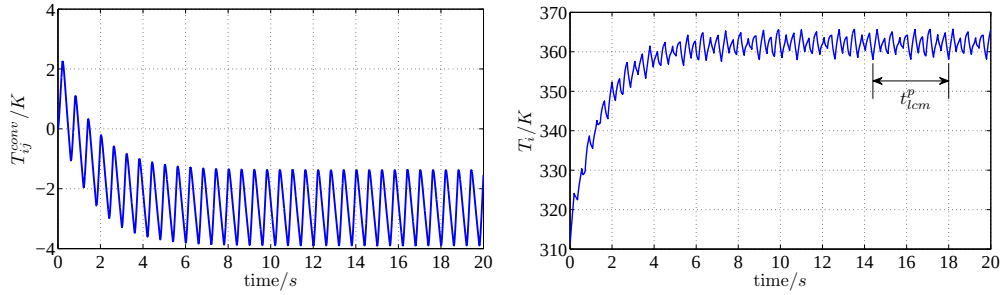
Proof Since $H_{ij}(t) = 0$ for $t \geq t_{ij}^{zero}$, one can easily prove that $T_{ij}^{conv}(t) = \int_0^{t_{ij}^{zero}} H_{ij}(\xi) \cdot u_j(t-\xi)\mathrm{d}\xi$ when $t \geq t_{ij}^{zero}$. Then, we discuss case by case.

- ***case*** $j \leq n$. Denote the period of the input for component j as t_j^p, we have $u_j(t) = u_j(t + t_j^p)$ for $t \geq 0$. Therefore, when $t \geq t_{ij}^{zero}$ we have:

$$T_{ij}^{conv}(t) = \int_0^{t_{ij}^{zero}} H_{ij}(\xi) \cdot u_j(t + t_j^p - \xi)\mathrm{d}\xi = T_{ij}^{conv}(t + t_j^p) \quad (3.19)$$

  which proving $T_{ij}^{conv}(t)$ is a periodic function of t and the fundamental period is t_j^p.

- ***case*** $n < j \leq N$. In this case, node j is not a processing node and thus $u_j(t) = K_{jj}T^{amb}$ is constant. When $t \geq t_{ij}^{zero}$, $T_{ij}^{conv}(t) =$



(a) T_{ij}^{conv} changes periodically after certain time instance

(b) T_i changes periodically after certain time instance

Figure 3.5: Examples of T_{ij}^{conv} and T_i varying with time, where $i = 1, j = 2$ in the figures. The thermal model comes from the three-core ARM model in [97] and $\mathbf{t^{off}} = [0.16, 0.4, 0.2]s$ while $\mathbf{t^{on}} = [0.2, 0.2, 0.2]s$.

$\int_0^{t_{ij}^{zero}} H_{ij}(\xi) \cdot u_j(t-\xi)\mathrm{d}\xi$, which is a definite integral and therefore constant. □

Lemma 3.11 *When $t \geq t^{end}$, the temperature of node i, $T_i(t)$, is a periodic function whose period t^{plcm} is the least common multiple of $t_1^p, t_2^p, \cdots, t_n^p$, where $t^{end} = \max_{1 \leq i,j \leq N} \{t_i^{init}, t_{ij}^{zero}\}$.*

Proof According to the first observation in thermal model, T_i^{init} in (3.6) is approximated as 0 when $t \geq t^{end}$. Then, the temperature of node i is $T_i(t) = \sum_{j=1}^{N} T_{ij}^{conv}(t)$. From Lem. 3.10, if $t \geq t^{end}$, $T_{ij}^{conv}(t)$ is either a periodic function or a constant function. In other words, temperature $T_i(t)$ is the sum of a set of periodic and constant functions. For real implementation, the periods of the PTM schemes should be rational numbers that rounded to certain unit, for example, $0.1ms$. Consequently, the ratios of the periods of the individual scheme are ratios of integers. Therefore, we conclude that $T_i(t)$ is a period function and its period, t^{plcm}, is the least common multiple of the periods of all the PTM schemes. □

For clear demonstration, Fig. 3.5 shows examples of T_{ij}^{conv} and T_i. It is worth noting that the i, j, and the thermal model are the same with those in Fig. 3.3. We find that after t_{ij}^{zero}, which is shown in Fig. 3.3 and is around $14s$, T_{ij}^{conv} changes periodically according to the period of the second PTM. In addition, it can be observed that the period of T_i in Fig. 3.5b is about $3.6s$, which agrees with the result calculated from the

periods of PTM schemes. Next, more lemmas about peak temperature analysis are given.

Lemma 3.12 $\mathbf{T}(t) < \mathbf{T}(t + t^{plcm})$ *when* $t < t^{end}$ *and* $\mathbf{T}(t) = \mathbf{T}(t + t^{plcm})$ *for* $t \geq t^{end}$.

Proof Since t^{plcm} is the least common multiple of $t_1^p, t_2^p, \cdots, t_n^p$, it is clear that $\mathbf{u}(t + t^{plcm}) = \mathbf{u}(t)$. For brevity, let $t_2 = t + t^{plcm}$ and $t_1 = t$. From (3.5), we have:

$$\begin{aligned} \mathbf{T}(t_2) - \mathbf{T}(t_1) = \quad & e^{\mathbf{A}\cdot t_2} \cdot \mathbf{T}^0 + \int_0^{t_2} \mathbf{H}(\xi) \cdot \mathbf{u}(t_2 - \xi) \mathrm{d}\xi - \\ & e^{\mathbf{A}\cdot t_1} \cdot \mathbf{T}^0 - \int_0^{t_1} \mathbf{H}(\xi) \cdot \mathbf{u}(t_2 - \xi) \mathrm{d}\xi \qquad (3.20) \\ = \quad & (e^{\mathbf{A}\cdot t_2} - e^{\mathbf{A}\cdot t_1}) \cdot \mathbf{T}^0 + \\ & \int_{t_1}^{t_2} \mathbf{H}(\xi) \cdot \mathbf{u}(t_2 - \xi) \mathrm{d}\xi \qquad (3.21) \end{aligned}$$

When $t_1 \geq t^{end}$, the matrix $\mathbf{H}$ is considered as a zero matrix, as demonstrated in Section 3.3.3. Since $\mathbf{H}(t) = e^{\mathbf{A}\cdot t} \cdot \mathbf{B}$, one can prove that $e^{\mathbf{A}\cdot t_2}$ and $e^{\mathbf{A}\cdot t_1}$ are zero matrices, too. Therefore, we have $\mathbf{T}(t_2) - \mathbf{T}(t_1) = 0$, that is, $\mathbf{T}(t) = \mathbf{T}(t + t^{plcm})$.

When $t_1 < t^{end}$, denote $\mathbf{u^s}$ as the input vector in the case that all nodes stay in sleep state. It is clear that $\int_{t_1}^{t_2} \mathbf{H}(\xi) \cdot \mathbf{u}(t_2 - \xi) \mathrm{d}\xi > \int_{t_1}^{t_2} \mathbf{H}(\xi) \cdot \mathbf{u^s} \mathrm{d}\xi$ because $\mathbf{H}(t_1)$ is positive and $\mathbf{u}(t_2 - \xi) > \mathbf{u^s}$ (As the concept of 'period' is used, we do not consider the scenario that $\mathbf{u} = \mathbf{u^s}$). In addition, since the sleep state steady temperature $\mathbf{T^{sleep}}$ is obtained from (3.4) by making $\frac{\mathrm{d}\mathbf{T}(t)}{\mathrm{d}t} = 0$, one can derive that

$$\mathbf{B} \cdot \mathbf{u^s} = -\mathbf{A} \cdot \mathbf{T^{sleep}} \qquad (3.22)$$

Now, replacing $\mathbf{H}(\xi)$ in (3.21) with $e^{\mathbf{A}\cdot\xi}\cdot\mathbf{B}$ yields:

$$\begin{aligned}
\mathbf{T}(t_2)-\mathbf{T}(t_1) &> (e^{\mathbf{A}\cdot t_2}-e^{\mathbf{A}\cdot t_1})\cdot\mathbf{T}^0+\int_{t_1}^{t_2}e^{\mathbf{A}\cdot\xi}\cdot\mathbf{B}\cdot\mathbf{u}^{\mathbf{s}}\mathrm{d}\xi \\
&= (e^{\mathbf{A}\cdot t_2}-e^{\mathbf{A}\cdot t_1})\cdot\mathbf{T}^0+\mathbf{A}^{-1}\cdot e^{\mathbf{A}\cdot t_2}\cdot\mathbf{B}\cdot\mathbf{u}^{\mathbf{s}}- \\
&\quad \mathbf{A}^{-1}\cdot e^{\mathbf{A}\cdot t_1}\cdot\mathbf{B}\cdot\mathbf{u}^{\mathbf{s}} \\
&= (e^{\mathbf{A}\cdot t_2}-e^{\mathbf{A}\cdot t_1})\cdot\mathbf{T}^0-\mathbf{A}^{-1}\cdot e^{\mathbf{A}\cdot t_2}\cdot\mathbf{A}\cdot\mathbf{T}^{\mathbf{sleep}}+ \\
&\quad \mathbf{A}^{-1}\cdot e^{\mathbf{A}\cdot t_1}\cdot\mathbf{A}\cdot\mathbf{T}^{\mathbf{sleep}} \\
&= (e^{\mathbf{A}\cdot t_2}-e^{\mathbf{A}\cdot t_1})\cdot\mathbf{T}^0-e^{\mathbf{A}^{-1}\cdot\mathbf{A}\cdot t_2\cdot\mathbf{A}}\cdot\mathbf{T}^{\mathbf{sleep}}+ \\
&\quad e^{\mathbf{A}^{-1}\cdot\mathbf{A}\cdot t_1\cdot\mathbf{A}}\cdot\mathbf{T}^{\mathbf{sleep}} \\
&= (e^{\mathbf{A}\cdot t_2}-e^{\mathbf{A}\cdot t_1})\cdot\mathbf{T}^0-e^{\mathbf{A}\cdot t_2}\cdot\mathbf{T}^{\mathbf{sleep}}+e^{\mathbf{A}\cdot t_1}\cdot\mathbf{T}^{\mathbf{sleep}} \\
&= (e^{\mathbf{A}\cdot t_2}-e^{\mathbf{A}\cdot t_1})\cdot(\mathbf{T}^0-\mathbf{T}^{\mathbf{sleep}}) \\
&= \mathbf{0} \qquad (3.23)
\end{aligned}$$

which proves that $\mathbf{T}(t_2)>\mathbf{T}(t_1)$. □

Lemma 3.13 *For node i, the peak temperature $T_i^\star$ equals the maximal temperature which is reached after t^{end}.*

Proof We prove this lemma by contradiction. Let t_i^{peak} denote the time slot when the peak temperature $T_i^\star$ is reached. Suppose that $t_i^{peak}\leq t^{end}$. From Lem. 3.12, one can find another time slot $t_2\geq t^{end}$ such that the temperature at t_2 is higher than $T_i^\star$, which conflicts with the assumption that $T_i^\star$ is the peak temperature. Therefore, we conclude that the peak temperature $T_i^\star$ is reached after t^{end} and thus equals the maximal temperature reached after t^{end}. □

Lemma 3.14 *The peak temperature of node i is:*

$$T_i^\star=\max_{t^{end}\leq t\leq t^{end}+t^{plcm}}\sum_{j=1}^{n}T_{ij}^{conv}(t)+T_i^{const} \qquad (3.24)$$

where T_i^{const} is a constant indicating the sum of influence from all non-processing nodes to node i when $t\geq t^{end}$, which is defined as,

$$T_i^{const}=\sum_{j=n+1}^{N}T_{ij}^{conv}(t^{end})=\sum_{j=n+1}^{N}\int_0^{t^{end}}H_{ij}(\xi)\cdot u_j^s\mathrm{d}\xi. \qquad (3.25)$$

Proof From Lem. 3.13, we can know that $T_i^\star$ can be obtained by finding the maximum of $T_i(t)$ for $t \geq t^{end}$. According to Lem. 3.11, when $t \geq t^{end}$, $T_i^\star$ is a periodic function and its period is t^{plcm}. Therefore the maximum of $T_i(t)$ is the local maximum in any period when $t \geq t^{end}$ and can be formulated as,

$$T_i^\star = \max_{t^{end} \leq t \leq t^{end}+t^{plcm}} \sum_{j=1}^{N} T_{ij}^{conv}(t) \tag{3.26}$$

From Lem. 3.10, $T_{ij}^{conv}(t)$ is a constant number when $n+1 \leq j \leq N$. Therefore, the peak temperature of node i can be formulated as (3.24).□

Based on above lemmas and definitions, we present our first important result in the following theorem.

Theorem 3.15 *For a multi-core processor with hardware and thermal models described above, when the* PTM *schemes characterized by* $\mathbf{t^{off}}$ *and* $\mathbf{t^{on}}$ *are applied, the peak temperature of the processor can be formulated as:*

$$T^\star = \max\{T_1^\star, T_2^\star, \cdots, T_n^\star\} \tag{3.27}$$

where $T_i^\star$ is formulated according to Lem. 3.14.

Proof Based on the second observation of thermal model, the peak temperature of the processor must be the peak temperature of the processing component nodes, which are the first n nodes in the model. Therefore, we have $T^\star = \max\{T_1^\star, T_2^\star, \cdots, T_n^\star\}$. □

Next, two algorithms are presented to calculate the peak temperature with different accuracies based on the analysis results obtained in this section.

3.6.2 Peak Temperature Calculating Algorithms

In this section, we present two algorithms, namely Accurate Neighbors Peak Temperature (ANPT) and Fast Bounding Peak Temperature (FBPT), to calculate the peak temperature of our system with different accuracies and speeds.

From Lem. 3.14, it is clear that to obtain the accurate maximum of $T_i(t)$, one should calculate the evaluation of $T_i(t)$ for at least one period, t^{plcm}. However, in the worst-case, for example, when the PTM periods are coprime numbers, t^{plcm} probably grows exponentially as the stage number n increases, which seriously prohibits the speed as well as the scalability of our approach. Therefore, we propose two algorithms ANPT and

FBPT that calculate the peak temperature with different levels of approximation. Algorithm ANPT offers a relatively accurate result with the expense of computing power is bounded while FBPT gives a less accurate peak temperature but requires much less computation.

Pre-Computing Matrices and Variables

One can get the peak temperature of the system based on the basic results in Section 3.6. However, it's worth noting that the matrices and variables that only depend on system inherent properties need to be pre-computed to avoid calculating them repeatedly in subsequent calculations. According to the peak temperature analysis, matrices and variables $\{t^{end}, \mathbf{u^a}, \mathbf{u^s}, \mathbf{H} = \{\mathbf{H}(t) : 0 \leq t \leq t^{end}\}$, and $T^{const}\}$ should be pre-computed. For clarity, we denote them by symbol **TM** in the following of this chapter. Next, we discuss the fast and simple algorithm FBPT.

Fast Bounding Peak Temperature Algorithm

Denoting the maximum of $T_{ij}^{conv}(t)$ as T_{ij}^{maxp} when $t \geq t^{end}$, we have following inequality from Lem. 3.14.

$$T_i^\star = \max_{t^{end} \leq t \leq t^{end}+t^{plcm}} \left\{ \sum_{j=1}^{n} T_{ij}^{conv}(t) \right\} + T_i^{const} \leq \sum_{j=1}^{n} \left\{ \max_{t^{end} \leq t \leq t^{end}+t^{plcm}} T_{ij}^{conv}(t) \right\} + T_i^{const} \tag{3.28}$$

For brevity, denote $\bar{T}_i^\star = \sum_{j=1}^{n} T_{ij}^{maxp} + T_i^{const}$. Inequality (3.28) indicates $\bar{T}_i^\star$ safely bounds the peak temperature of node i. From a set of systematic experiments, we observe that $\bar{T}_i^\star$ is close to the real maximum of $T_i(t)$ in value. The reason of this phenomenon is: due to heat transfer delay between two nodes, the oscillation amplitude of $T_{ij}^{conv}(t)$ is considerably weak compared to the magnitude of $T_{ii}^{conv}(t)$ when $t \geq t^{end}$, especially for the scenario that nodes i and j are far away from each other on the floorplan of the processor. Therefore the error caused by T_{ij}^{maxp} is acceptable, making $\bar{T}_i^\star$ be a good approximation of the actual result. Adopting this approximation has two advantages: first, the calculation of $\bar{T}_i^\star$, as shown in (3.29), can be performed quickly since the element operation, that is, computing T_{ij}^{maxp}, requires little resource according to its definition; second, we conjecture that the peak temperature offered by this method has only one minimum based on a set of systematic experiments, thus the gradient-descent-search can be utilized to find the

Algorithm 3 Fast Bounding Peak Temperature Computation

Input: **TM**, $\mathbf{t^{off}}$, $\mathbf{t^{on}}$, $\mathbf{t^{swoff}}$, and $\mathbf{t^{swon}}$
Output: The peak temperature of the system $T^\star$
1: $\mathbf{t^{p}} \leftarrow \mathbf{t^{off}} + \mathbf{t^{on}}$, $\mathbf{t^{act}} \leftarrow \mathbf{t^{on}} + \mathbf{t^{swoff}}$,
2: $\mathbf{t^{slp}} \leftarrow \mathbf{t^{off}} - \mathbf{t^{swoff}}$, $T^\star \leftarrow -\infty$
3: **for** each processing component node $i \leq n$ **do**
4: $\quad T_i^\star \leftarrow 0$
5: $\quad$ **for** each processing component node $j \leq n$ **do**
6: $\quad\quad t_{ij}^{max} \leftarrow t^{end} + \text{periods}_j$
7: $\quad\quad$ construct input trace **Ptrace** from $t = 0s$ to $t = t_{ij}^{max}$
8: $\quad\quad$ thermal trace $\mathbf{T_{ij}^{conv}} \leftarrow \mathbf{IFFT}\{\mathbf{FFT}\{H_{ij}\} * \mathbf{FFT}\{\mathbf{Ptrace}\}\}$
9: $\quad\quad T_{ij}^{maxp} \leftarrow \max_{t^{end} \leq t \leq t_{ij}^{max}} \mathbf{T_{ij}^{conv}}(t)$
10: $\quad\quad T_i^\star \leftarrow T_i^\star + T_{ij}^{maxp}$
11: $\quad$ **end for**
12: $\quad T_i^\star \leftarrow T_i^\star + T_i^{const}$
13: **end for**
14: $T^\star \leftarrow \max\{T_1^\star, T_2^\star, \cdots, T_n^\star\}$

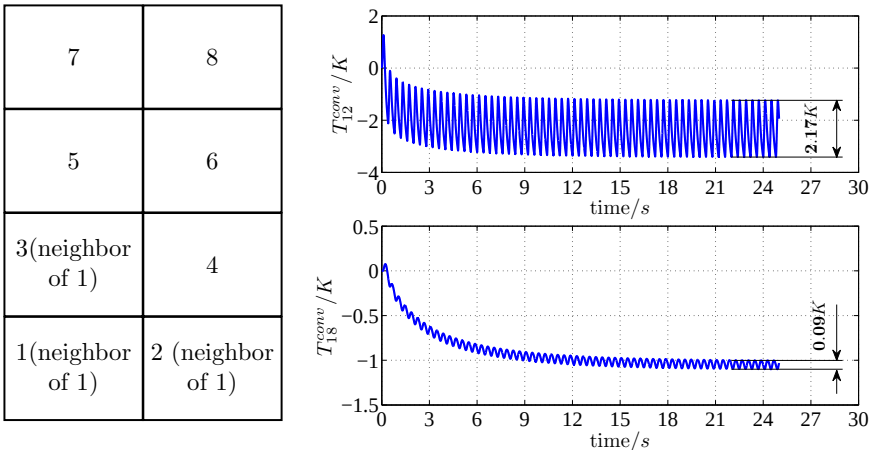
optimal solution. By adopting this definition, our approach significantly reduces the storage space requirement and time expense.

$$T_{ij}^{maxp} = \max_{t^{end} \leq t \leq t^{end} + t_j^p} T_{ij}^{conv}(t) \tag{3.29}$$

The pseudo-code is profiled in Algo. 3. It is worth noting that the input $\mathbf{t^{off}}$ and $\mathbf{t^{on}}$ are revised to $\mathbf{t^{slp}}$ and $\mathbf{t^{act}}$ to comprise the mode-switching overhead. Then the peak temperature of all the processing nodes are computed (lines 3 to 12) and the highest one is assigned to the peak temperature $T^\star$ (line 14). In the implementation, convolution (3.7), that is, $T_{ij}^{conv}(t)$, is actually a discrete convolution, thus can be converted to a circular convolution which is implemented with the Fast Fourier Transform(**FFT**) to reduce the complexity(line 8).

Accurate Neighbors Peak Temperature

Algorithm FBPT is suitable for the scenario that don't require high result accuracy. However, when high accuracy is desired, Algorithm FBPT fails to meet the requirement. Therefore, to give a relatively accurate peak



(a) (b)

Figure 3.6: (a): An example of neighbor nodes on a block mapping of the silicon die layer in an Octa-processing-component model. (b): Thermal influence form node 2 and node 8 to node 1 in left sub-figure. The PTM schemes on node 2 and node 8 are the same: $t^{off} = 100ms$, $t^{on} = 100ms$.

temperature, we propose another algorithm, ANPT, in this section. Before the algorithm is presented, the concept of 'neighbor' is introduced.

Definition 3.16 (Neighbors) *For a node i in the thermal model, its neighbors are the same-layer-nodes which have common boundary with it. Note that we also consider the node itself as its neighbor in this chapter.*

An example is shown in Fig. 3.6a, where nodes 1 to 3 are the neighbors of node 1 according to the definition.

The basic idea of ANPT originates from the observation that the oscillation of the thermal influence from non-neighbor nodes is much less than that from neighbor nodes. Fig. 3.6b displays T_{12}^{conv} and T_{18}^{conv}, which are the thermal influence form node 2 and node 8 to node 1 in Fig. 3.6a. As shown in the figure, when the same t^{off} and t^{on} are adopted, the oscillation amplitude of T_{18}^{conv} is less than $0.1K$, which is around 4% of that of T_{12}^{conv}. This is caused by the heat transfer delay, as mentioned in previous section. Therefore, to calculate $T_i^\star$, the influence from non-neighbor nodes can still be approximated by FBPT to save computing effort because they have little impact on $T_i^\star$. Meanwhile, the calculation accuracy of $T_i^\star$ can be significantly improved by calculating the very real thermal influence from its neighbor nodes. It is worth noting that in this way, ANPT achieves the scalability since the number of neighbor nodes

Algorithm 4 Accurate Neighbors Peak Temperature Computation

Input: **TM**, $\mathbf{t^{off}}$, $\mathbf{t^{on}}$, $\mathbf{t^{swoff}}$, and $\mathbf{t^{swon}}$
Output: The peak temperature of the system $T^\star$

1: **periods** $\leftarrow \mathbf{t^{off}} + \mathbf{t^{on}}$, $\mathbf{t^{act}} \leftarrow \mathbf{t^{on}} + \mathbf{t^{swoff}}$
2: $\mathbf{t^{slp}} \leftarrow \mathbf{t^{off}} - \mathbf{t^{swoff}}$, $T^\star \leftarrow -\infty$
3: **for** each processing component node $i \leq n$ **do**
4: $\quad T_i^\star \leftarrow 0$
5: $\quad$ **NBS** $\leftarrow \emptyset$
6: $\quad$ **NBS_T** $\leftarrow \emptyset$
7: $\quad$ **for** each processing component node $j \leq n$ **do**
8: $\quad\quad t_{ij}^{max} \leftarrow t^{end} + \text{periods}_j$
9: $\quad\quad$ construct input trace **Ptrace** from $t = 0s$ to $t = t_{ij}^{max}$
10: $\quad\quad$ temperature trace $\mathbf{T_{ij}^{conv}} \leftarrow \mathbf{IFFT}\{\mathbf{FFT}\{H_{ij}\} * \mathbf{FFT}\{\mathbf{Ptrace}\}\}$
11: $\quad\quad$ **if** node j is a neighbor of node i **then**
12: $\quad\quad\quad$ **NBS** $\leftarrow$ **NBS** $\bigcup j$
13: $\quad\quad\quad$ **NBS_T** $\leftarrow$ **NBS_T** $\bigcup \{\mathbf{T_{ij}^{conv}} : t^{end} \leq t \leq t_{ij}^{max}\}$
14: $\quad\quad$ **else**
15: $\quad\quad\quad T_{ij}^{maxp} = \max_{t^{end} \leq t \leq t_{ij}^{max}} \mathbf{T_{ij}^{conv}}(t)$
16: $\quad\quad\quad T_i^\star = T_i^\star + T_{ij}^{maxp}$
17: $\quad\quad$ **end if**
18: $\quad$ **end for**
19: $\quad t_i^{plcm} \leftarrow$ the least common multiple of $\{t_j^p | j \in \mathbf{NBS}\}$
20: $\quad$ **SUM** $\leftarrow$ **0**
21: $\quad$ **for** each processing component node $j \in$ **NBS do**
22: $\quad\quad$ get $\mathbf{exT_{ij}^{conv}}$ by extending $\mathbf{T_{ij}^{conv}}$ of j in **NBS_T** to length t_i^{plcm}
23: $\quad\quad$ **SUM** $\leftarrow$ **SUM** $+ \mathbf{exT_{ij}^{conv}}$
24: $\quad$ **end for**
25: $\quad T_i^\star = T_i^\star + \max(\mathbf{SUM}) + T_i^{const}$
26: **end for**
27: $T^\star \leftarrow \max\{T_1^\star, T_2^\star, \cdots, T_n^\star\}$

is limited. For instance, in Fig. 3.6, a node can have at most 4 neighbor nodes, including itself. The pseudo code of ANPT is presented in Algo. 4.

Algo. 4 requires the same input as Algo. 3. For any processing node i, when calculating the thermal influence from any processing node j, the algorithm checks if node j is a neighbor of node i (line 11). If

so, node j and the corresponding thermal trace are put into the set **NEIGHBORS** and **NEIGHBORS_TRACE** (lines 12- 13) for calculation in lines 19-24. Otherwise, T_{ij}^{maxp} is directly added to $T_i^\star$ to reduce the time cost (lines 16). After traversing all the nodes (line 18), the algorithm tackles the nodes in **NEIGHBORS** and find their maximal thermal influence to node i by extending the $\mathbf{T_{ij}^{conv}}$s in **NEIGHBORS_TRACE** to the LCM length and calculating their summation (lines 19-24). Finally $T_i^\star$ is obtained and the highest one is assigned to the peak temperature $T^\star$ (line 27).

3.7 Real-time Analysis and Problem Formulations

In this section, we first analyze how to guarantee the deadline constraints. Then, the formulation of our optimization problem is presented. The formulation is broken down by a set of sub-problems easier to solve. Finally, the overall algorithm is given to solve the thermal optimization problem. How to solve the sub-problem is discussed in next section.

3.7.1 Real-time analysis

Before formulating the optimization problem, we first present the timing property analysis to ensure all tasks are finished before their deadlines. For an n-stage pipelined processor that employs PTM schemes characterized by $\mathbf{t^{off}}$ and $\mathbf{t^{on}}$, we define

$$\boldsymbol{R} = \{R_1, \cdots, R_n\},\ R_i = \frac{t_i^{on}}{t_i^{on} + t_i^{off}} \tag{3.30}$$

Then, suppose an application with arrival curve α is processed by the n pipelined stages, the deadline D can be guaranteed if the following condition holds:

$$\bigotimes_{i=1}^{m} \bar{\beta}_i \geq \alpha^u(\Delta - D) \tag{3.31}$$

where $\bar{\beta}_i$ is the event-based service curve provide by stage i and is defined as $\bar{\beta}_i = \lfloor \frac{\beta_i}{c_i} \rfloor$, as mentioned in Section 3.3. Before further analysis, a lemma is presented.

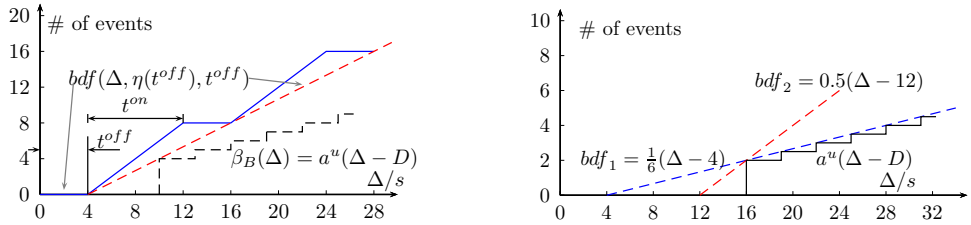


Figure 3.7: (a) The corresponding bounded delay function for a given PTM service curve. The x-axis indicates any time interval with length Δ. (b) A service curve demand $a^u(\Delta - D)$ is upper bounded by two bounded delay functions with different b and ρ.

Lemma 3.17

$$\bigotimes_{i=1}^{n} \bar{\beta}_i \geq \min_{i=1}^{n} \left(\frac{R_i}{c_i}\right)[\Delta - \sum_{i=1}^{n} (t_i^{off} + c_i)] \tag{3.32}$$

Proof It is worth noting that this lemma is already proved in [23] with the assumption that t_i^{on} is a positive multiple of c_i. Now we prove Lem. 3.17 for general cases where t_i^{on} is a real number.

The lower service curve provided by any stage i controlled by PTM (t_i^{off}, t_i^{on}) is formulated as $\beta_i(\Delta) = \max\left(\left\lfloor \frac{\Delta}{t_i^{off}+t_i^{on}} \right\rfloor \cdot t_i^{on}, \Delta - \left\lceil \frac{\Delta}{t_i^{off}+t_i^{on}} \right\rceil \cdot t_i^{off}\right)$ [50], which is a TDMA curve as shown in Fig. 3.7a. Then we can generate a corresponding bounded delay function $bdf_i = \max[0, R_i(\Delta - t_i^{off})]$ which satisfies,

$$\beta_i \geq bdf_i \tag{3.33}$$

Therefore $\bar{\beta}_i = \left\lfloor \frac{\beta_i}{c_i} \right\rfloor \geq \left\lfloor \frac{bdf_i}{c_i} \right\rfloor \geq \left\lfloor \frac{R_i}{c_i}(\Delta - t_i^{off}) \right\rfloor$. Since $\lfloor a \rfloor \geq a - 1$, we have,

$$\bar{\beta}_i \geq \frac{R_i}{c_i}(\Delta - t_i^{off}) - 1 = \frac{R_i}{c_i}(\Delta - t_i^{off} - c_i) \tag{3.34}$$

According to the rule of min-plus convolution of rate-latency service curve $\beta_{R_1,T_1} \otimes \beta_{R_2,T_2} = \beta_{min(R_1,R_2),T_1+T_2}$ in [60], we have,

$$\bigotimes_{i=1}^{n} \bar{\beta}_i \geq \bigotimes_{i=1}^{n} \frac{R_i}{c_i}(\Delta - t_i^{off} - c_i) = \min_{i=1}^{n} \left(\frac{R_i}{c_i}\right)[\Delta - \sum_{i=1}^{n} (t_i^{off} + c_i)] \tag{3.35}$$

To consider the effect of mode-switching to the QoS of our system, the t_i^{on} and t_i^{off} in (3.30) and (3.32) should be revised as $t_i^{on} = t_i^{on} - t_i^{swon}$ and $t_i^{off} = t_i^{off} + t_i^{swon}$, respectively. Then, we state the next theorem as the real-time analysis result.

Theorem 3.18 (Guarantee of Deadline) *Assume an input event stream characterised by arrival curve α traverses an n-stage pipelined system which is controlled by* PTM *schemes depicted by* $\mathbf{t^{off}}$ *and* $\mathbf{t^{on}}$*, the end-to-end deadline D can be guaranteed if the following inequality holds,*

$$\min_{i=1}^{n}(\frac{R_i}{c_i})[\Delta - \sum_{i=1}^{n}(t_i^{off} + t_i^{swon} + c_i)] \geq \alpha^u(\Delta - D) \tag{3.36}$$

where $R_i = \frac{t_i^{on} - t_i^{swon}}{t_i^{on} + t_i^{off}}$

Proof From Lem. 3.17, we have $\bigotimes_{i=1}^{m}\bar{\beta}_i \geq \alpha^u(\Delta - D)$ if (3.36) holds, which ensures taht the deadline is satisfied. □

3.7.2 Formulation and transformation of the Optimization Problem

According to Thm. 3.18, our problem is transformed into an optimization problem.

$$\begin{aligned} \text{minimize} \quad & T^{\star}(\boldsymbol{R}, \mathbf{t^{off}}) \\ \text{subject to} \quad & \min_{i=1}^{n}(\frac{R_i}{c_i})[\Delta - \sum_{i=1}^{n}(t_i^{off} + t_i^{swon} + c_i)] \geq \alpha^u(\Delta - D) \\ & 0 \leq \boldsymbol{R} \leq 1,\ \mathbf{t^{off}} \geq \mathbf{t^{swoff}},\ \mathbf{t^{on}} \geq \mathbf{t^{swon}} \end{aligned} \tag{3.37}$$

Handling this problem directly presents significant difficulty as the exploring space is hard to determine. Therefore, we transform it into a set sub-problems, which is discussed below.

The right side of the inequality (3.36) can be upper-bounded by a set of minimum bounded-delay functions $bdf_{min}(\Delta) = \max[0, \rho(\Delta - b)]$, as shown in Fig. 3.7b. For a given b, the bounded-delay functions is determined by calculating the corresponding ρ:

$$\rho(b) = \inf\{\rho : \rho(\Delta - b) \geq \alpha^u(\Delta - D), \forall\Delta \geq 0\} \tag{3.38}$$

Then, for every individual function $bdf_{min}(\Delta) \geq \alpha^u(\Delta - D)$, the deadline D is satisfied when the following inequity holds:

$$\min_{i=1}^{n}(\frac{R_i}{c_i})[\Delta - \sum_{i=1}^{n}(t_i^{off} + t_i^{swon} + c_i)] \geq bdf_{min}(\Delta, \rho, b) \tag{3.39}$$

which is equivalent to:

$$\min_{i=1}^{n}(\frac{R_i}{c_i}) \geq \rho \quad \& \quad \sum_{i=1}^{n}(t_i^{off} + t_i^{swon} + c_i) \leq b \tag{3.40}$$

Finally, for every individual pair of b and ρ, problem (3.37) is transformed to the following sub-problem:

$$\begin{aligned} \text{minimize} \quad & T^{\star}(\boldsymbol{R}, \mathbf{t^{off}}) \\ \text{subject to} \quad & \min_{i=1}^{n}(\frac{R_i}{c_i}) \geq \rho \\ & \sum_{i=1}^{n}(t_i^{off} + t_i^{swon} + c_i) \leq b \\ & 0 \leq \boldsymbol{R} \leq 1,\ \mathbf{t^{off}} \geq \mathbf{t^{swoff}},\ \mathbf{t^{on}} \geq \mathbf{t^{swon}} \end{aligned} \tag{3.41}$$

3.7.3 Overall algorithm to minimize peak temperature

With formulation (3.41), Algo. 5 provides the pseudo-code of our approach. It's worth noting that b should vary in a feasible region $[b_{min}, b_{max}]$, where $b_{min} = \sum_{i=1}^{n}(t_i^{swoff} + t_i^{swon} + c_i)$ since t_i^{off} is lower bounded by t_i^{swoff}, and $b_{max} = \max\{b : \max(0, \Delta - b) \geq \alpha^u(\Delta - D),\ \forall \Delta \geq 0\}$, which is obtained from [1].

3.8 Solving the sub-problem

This section presents two algorithms to solve the sub-problem (3.41). Each algorithm can be called individually by Algo. 5 to minimize the peak temperature and find the optimal PTM schemes.

Lemma 3.19 *For problem* (3.41)*, R_i can be obtained safely by the following equation:*

$$R_i = c_i \rho \tag{3.42}$$

Algorithm 5 Peak Temperature Optimization

Input: **TM**, α, D, n, $\mathbf{c}$, $\mathbf{t^{swon}}$,$\mathbf{t^{swoff}}$, search step η
Output: $T^{\star}_{min}$, $\mathbf{t^{on}}$, $\mathbf{t^{off}}$

```
calculate [b_min, b_max]
T*_min ← ∞, R ← 0, t^off ← 0
for each b = b_min to b_max with step η do
    get corresponding ρ from (3.38)
    solve sub-problem (3.41) and get
    T*, R_◇, t^off_◇
    if T* < T*_min then
        T*_min ← T*, R ← R_◇,
        t^off ← t^off_◇
    end if
end for
calculate t^on with t_i^on = (R_i t_i^off + t_i^swon)/(1 − R_i)
```

Proof From the definition of R_i, one can derive $t_i^{on} = (R_i t_i^{off} + t_i^{swon})/(1 - R_i)$. Then, we have $\frac{\mathrm{d}t_i^{on}}{\mathrm{d}R_i} > 0$, which means a larger R_i results in a higher partition of t_i^{on}, that is, a higher peak temperature. Therefore R_i should equal its lower bound $c_i\rho$ such that the peak temperature won't be elevated unnecessarily. □

From Lem. 3.19, $T^{\star}(\boldsymbol{R}, \mathbf{t^{off}})$ in (3.41) can be transformed to a function of $\mathbf{t^{off}}$, $T^{\star}(\mathbf{c}\cdot\rho, \mathbf{t^{off}})$, with the constraint:

$$\sum_{i=1}^{n} (t_i^{off} + t_i^{swon} + c_i) \leq b \tag{3.43}$$

Next we present two fast algorithms to solve the sub-problem (3.41) corresponding to the two peak temperature algorithms proposed in Section 3.6.2.

3.8.1 Algorithm FBGD to solve the FBPT based sub-problem

In this case, $T^{\star}(\boldsymbol{R}, \mathbf{t^{off}})$ is calculated with the FBPT algorithm. Intuitively, one can brutally search the whole exploring space to find the optimal solution. However, as the stage number n increases, the exploring space expends (approximated) exponentially and this approach will finally be

infeasible. For example, if $n = 8$ and every t_{off} has 50 candidates in the exploring space, there will be $50^8 = 3.90625 \times 10^{13}$ combinations in total. The brutally searching algorithm needs more than 120 years to finish if the computer can check 10000 combinations per second. Therefore, a more clever algorithm is needed to solve sub-problem (3.41). We first introduce the following conjecture obtained from a set of systematic experiments.

Conjecture 3.20 (Unimodal Peak Temperature) *Given ρ and $\boldsymbol{R} = \mathbf{c} \cdot \rho$, the peak temperature calculated by* FBPT *is a unimodal function of t_i^{off} in the feasible region of t_i^{off}.*

An example is shown in Fig. 3.8, the obtained peak temperature first decrease and then increase as t_i^{off} varies from its lower bound to upper bound while other t^{off}s stay constant.

From Conjecture 3.20, the local minima is also the global minima of the peak temperature. Therefore, inspired by the gradient descent method, we propose an Fast-Bounding-based adaptive-step-Gradient-Descent(FBGD) algorithm to find the optimal $\mathbf{t^{off}}$ in the exploring space.

Algo. 6 outlines the pseudo-code of the algorithm. It takes the thermal model, the stage number, mode-switching overhead vectors, b, ρ, $\mathbf{c}$, a initial step size ξ, and a minimal step size ξ_{min} as input. The iteration starts at the initial point $\mathbf{t^{swoff}}$ (line 3). In every iteration, it first checks whether feeding one current step size ξ to $\mathbf{t^{off}}$ will violate the limit of b (line 5). If so, the current step size should shrink to a smaller value which satisfies constraint 3.43. In this algorithm, the golden section scale is utilized to obtain the new step size ξ (line 6). Then all the possible directions are checked (line 11) to find the direction which leads to the current steepest descent. If the the minimal gradient results in a lower peak temperature, the corresponding direction will be selected to update $\mathbf{t^{off}}$ by adding ξ to $\mathbf{t^{off}}$ (line 16). Otherwise, we adapt the step to a smaller size which is also determined by the golden section rule (line 18). the algorithm executes the iteration until the minimal step size is reached (line 8).

3.8.2 Algorithm ANSA to solve the ANPT based sub-problem

Obtained from algorithm ANPT, the peak temperature varies irregularly in the space of $\mathbf{t^{off}}$. As shown in Fig. 3.8, there exist several cliffs as t_i^{off}

Algorithm 6 FBGD: solve the FBPT based sub-problem with given b and ρ

Input: **TM**, n, b, ρ, $\mathbf{c}$, $\mathbf{t^{swon}}$, $\mathbf{t^{swoff}}$, initial step size ξ, minimal step size ξ_{min}
Output: $T^\star$, $\boldsymbol{R}_\diamond$, $\mathbf{t^{off}}_\diamond$

$\mathbf{e_1} \leftarrow (1, 0, \cdots, 0)$, $\mathbf{e_2} \leftarrow (0, 1, \cdots, 0)$, $\mathbf{e_n} \leftarrow (0, 0, \cdots, 1)$, $\mathbf{g} \leftarrow \mathbf{0}_{1,n}$
$\boldsymbol{R} \leftarrow \rho \cdot \mathbf{c}$, $\mathbf{t^{off}} \leftarrow \mathbf{t^{swoff}}$
$T^\star_{last} \leftarrow T^\star(\boldsymbol{R}, \mathbf{t^{off}})$ calculated by Algo. 3
while true **do**
  if $\sum_{i=1}^{n} (t_i^{off} + t_i^{swon} + c_i) + \xi > b$ **then**
    $\xi \leftarrow 0.618 * (b - \sum_{i=1}^{n} (t_i^{off} + t_i^{swon} + c_i))$
  end if
  if $\xi \leq \xi_{min}$ **then**
    break
  end if
  for each $1 \leq i \leq n$ **do**
    $g_i \leftarrow (T^\star(\boldsymbol{R}, \mathbf{t^{off}} + \xi \mathbf{e_i}) - T^\star_{last})$
  end for
  if $\min\{\mathbf{g}\} < 0$ **then**
    find i where $\mathbf{g} == \min\{\mathbf{g}\}$, $\mathbf{t^{off}} \leftarrow \mathbf{t^{off}} + \xi \mathbf{e_i}$,
    $T^\star_{last} \leftarrow T^\star_{last} + \min\{\mathbf{g}\}$
  else
    $\xi \leftarrow 0.618 * \xi$
  end if
end while
$T^\star \leftarrow T^\star_{last}$, $\boldsymbol{R}_\diamond \leftarrow \boldsymbol{R}$,$\mathbf{t^{off}}_\diamond \leftarrow \mathbf{t^{off}}$

increases, indicating the gradient descent method is not suitable any more. Therefore, we determine to deploy an heuristic algorithm to find the sub-optimal solution of sub-problem (3.41) rather than the global minimum, which can only be safely discovered by the brutally searching method. The well-known Simulated Annealing (SA) algorithm is adopted to solve (3.41) in our approach and is denoted as ANSA. Note that the typical SA algorithm searches in the space of $\mathbf{t^{off}}$ without linear constraints. Therefore, the section of updating a new candidate in the SA algorithm needs to be revised to guarantee the new candidate meets the linear constraints in (3.41). Algo. 7 presents the pseudo code of the revised section.

Algo. 7 uses the current $\mathbf{t^{off}}$, b, n, $\mathbf{c}$, $\mathbf{t^{swon}}$, the current simulated an-

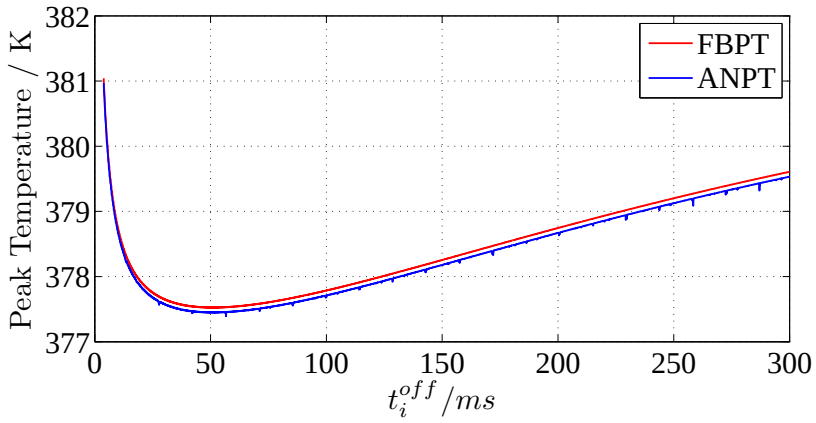


Figure 3.8: Peak temperature obtained by FBPT and ANPT change as t_i^{off} increases, while other t^{off}s keep constant.

Algorithm 7 Update $\mathbf{t^{off}}$ with linear constraint (3.40) for SA

Input: $\mathbf{t^{off}}$, b, n, $\mathbf{c}$, $\mathbf{t^{swon}}$, Current SA Temperature $\mathbf{T^{SA}}$, Initial SA Temperature T_{init}^{SA},
lower bound $\mathbf{bd^l}$, upper bound $\mathbf{bd^u}$
Output: $\mathbf{t^{off}}$
if $\sum_{i=1}^{n}(t_i^{off} + t_i^{swon} + c_i) \geq b$ **then**
  $\mathbf{t^{off}} \leftarrow \mathbf{bd^l}$
  return
end if
$k \leftarrow$ an uniformly distributed random integer between 1 and n
temp upper bound $bd^{\dagger} \leftarrow \min[bd_k^u, b - \sum_{i=1}^{n}(t_i^{off} + t_i^{swon} + c_i) + t_k^{off}]$
$step \leftarrow$ an uniformly distributed random number between -1 and 1
$\eta \leftarrow T_k^{SA}/T_{init}^{SA}$
$t_k^{off} \leftarrow t_k^{off} + \eta * (bd^{\dagger} - bd_k^l) * step$
$\mathbf{t^{off}} \leftarrow \max[\mathbf{t^{off}}, \mathbf{bd^l}]$, $\mathbf{t^{off}} \leftarrow \min[\mathbf{t^{off}}, \mathbf{bd^u}]$

nealing temperature $\mathbf{T^{SA}}$, the initial SA temperature, and the lower and upper bounds of $\mathbf{t^{off}}$ to update the new candidate. $\mathbf{bd^l}$ and $\mathbf{bd^u}$ are the strict constraints that $\mathbf{t^{off}}$ must not violate, which can be set as $\mathbf{t^{swoff}}$ and $(b - \sum_{i=1}^{n} (t_i^{swoff} + t_i^{swon} + c_i)) \cdot [1, \cdots, 1]'$, respectively. The algorithm first check if the current $\mathbf{t^{off}}$ satisfies the constraint (3.43) (line 2). If (3.43) is violated, $\mathbf{t^{off}}$ will be assigned to its lower bound, indicating an error $\mathbf{t^{off}}$ is given (line 3). Otherwise, a t_i^{off} in $\mathbf{t^{off}}$ is chosen randomly to be updated by adding a random number between $-\eta * (tbd - bd_k^l)$ and $\eta * (tbd - bd_k^l)$ to it (line 10). η is a control coefficient to ensure the convergence of the algorithm. Finally, the new $\mathbf{t^{off}}$ is checked with $\mathbf{bd^l}$ and $\mathbf{bd^u}$ and revised to the safe region if necessary (line 11).

3.9 Case Studies

We evaluate the effectiveness and feasibility of our proposed approaches in this section. Four approaches are compared: (1) our FBPT based adaptive-step Gradient-Descent algorithm (FBGD), (2) the ANPT based Simulated Annealing algorithm (ANSA) (3) Brutally Searching based PBOO algorithm (BS), and (4) the brutally searching Sub-Deadline Partitions algorithm introduced in Section 3.5 (SDP).

3.9.1 Setup

The algorithm BS is adopted to validate the effectiveness of FBGD. It solves the sub-problem by brutally checking all the candidates of $\mathbf{t^{off}}$ in the exploring space with a given fixed step and returns the best one. For fair comparison, BS also uses FBPT to calculate the peak temperature, same with FBGD. The algorithm SDP brutally examines all the possible sub-deadline partitions and returns the one yielding the lowest peak temperature. It adopts ANPT to get the peak temperature for a fair competition with PBOO based algorithms. Approach ANSA terminates its iteration once the change in best peak temperature is less than 1×10^{-3}K or the iteration number reaches the upper bound $2500 \times n$. Moreover, the initial point in ANSA is set as the optimal solution given by FBGD to eliminate bad solutions.

We implement the approaches on two simulated platforms: (1) a homogeneous multi-processor ARM platform with eight cores (ARM), (2) the Single-Chip Cloud Computer (SCC), a processor created by Intel that has 48 distinct physical cores [46]. The power and thermal parameters

Table 3.1: WCETs of the applications in 3-stage and 4-stage scenarios (unit: *ms*)

Application	3-stage	4-stage
H263	[8.5 , 5.4 , 2.2]	[1.3 , 7.2 , 5.4 , 2.2]
MP3	[12.6 , 8.1 , 4.59]	[1.8 , 10.8 , 8.1 , 4.59]
MADplayer	[7.4 , 5.1 , 6.8]	[3.4 , 4.4 , 5.1 , 6.8]

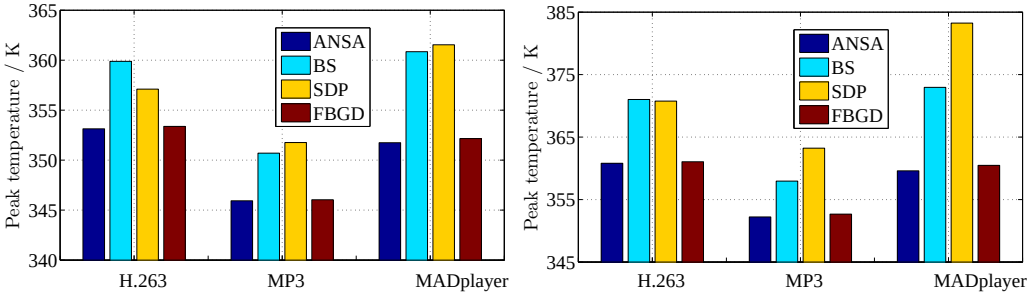
of the two platforms come from [97, 83] and parameter calibration. The processor floor-plan of SCC is obtained from [46] and the thermal matrices **G**, **C** and **K** are obtained from the HotSpot toolbox. All the simulations are performed on a computer with an Intel i7-4770 processor and 16GB memory. Regarding determining the layout of activated cores, we select and activate the n cores whose locations are close to core #1 and the remaining cores stay in 'sleep' state.

Our simulation runs the peak temperature optimization for three partitioned applications: (1) the H.263 decoder application modeled by four tasks [75], (2) the MP3 decoder application which can be split into five tasks [75], and (3) the MADplayer application that consists of five tasks [106]. Their worst-case execution times are determined and scaled from [75] and [106], which are listed in Tab. 3.1. All the mode-switching overheads are set as $\mathbf{t^{swoff}} = \mathbf{t^{swon}} = (1, \cdots, 1)ms$. The activation periods of the three applications are set as $55ms$, $60ms$ and $30ms$, respectively. The relative deadline D is determined by deadline factor δ: $D = \delta \times p$, where p is the activation period.

3.9.2 Results

The three applications are executed with deadline factor $\delta = 1$ on the two platforms. The generated peak temperature are examined for three- and four- stage scenarios with different step sizes. Fig. 3.9 and Fig. 3.10 provide results on ARM platform while Fig. 3.11 and Fig. 3.12 show those from SCC platform. From the figures we can see that: (1) In all cases, the peak temperatures obtained from approaches ANSA and FBGD are less than those from SDP approach. (2) For the two platforms, the temperature difference between our PBOO based algorithms and algorithm SDP gets bigger when stage number increases. The temperature difference between SDP and FBGD is around 15K for the application MADplayer in 4-stage scenario, as shown in Fig. 3.10b. The above two observations can be explained by that SDP pays burst for more times

when stage number increases and therefore returns higher peak temperatures. (3) The temperature differences between ANSA and FBGD are minor in value, or in other words, unnoticeable, in most cases. The reason is that the error of method FBGD to real peak temperature is trivial due to that the activated stages are few. As demonstrated in Fig. 3.13b and Fig. 3.14, ANSA delivers lower peak temperature in large stage-number cases. (4) Approach FBGD offers lower peak temperature than those from BS, especially when larger step size is adopted, which proves the effectiveness of FBGD. This is due to that FBGD uses the adaptive step size and thus can control the error of the solution while BS searches the best solution with a fixed step and returns an inaccurate solution. The error of approach BS is considerably large when the search is coarse-grained. We notice that the temperature of BS on ARM platform is even higher than that from SDP for application H.263 in 3-stage scenario. (5) Compared to ARM, the peak temperature and the gaps between different approaches on SCC platform are much lower, which is owed to (a) the difference in the thermal parameter, such as chip thickness, heat sink size. (b) SCC has 48 cores, only turning on three or four cores won't warm the whole chip sufficiently, therefore the heat can be conducted to the environment faster and results in lower peak temperature.

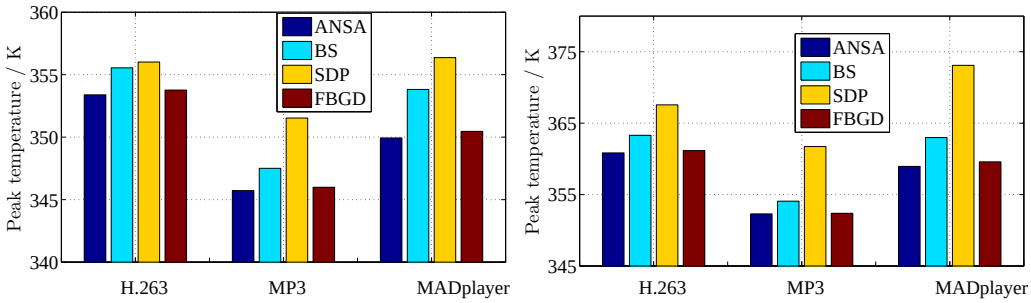


(a) ARM: 3-stage and step size: 4ms

(b) ARM: 4-stage and step size: 4ms

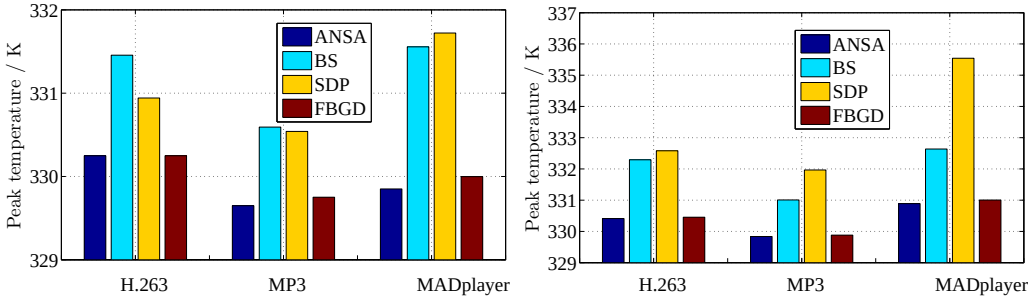
Figure 3.9: Peak Temperature produced by the tested approaches with $\delta = 1$ and step size is $4ms$ when the three applications are executed on platform ARM with different stage numbers.

To further confirm the effectiveness and feasibility of our approaches, we simulate a randomly generated application on the two platforms and then increase the stage number n from 2 up to 8 on ARM and to 24 on SCC, respectively. The WCETs of the sub-tasks are randomly generated



(a) ARM: 3-stage and step size: 2ms (b) ARM: 4-stage and step size: 2ms

Figure 3.10: Peak Temperature produced by the tested approaches with $\delta = 1$ and step size is $2ms$ when the three applications are executed on platform ARM with different stage numbers.

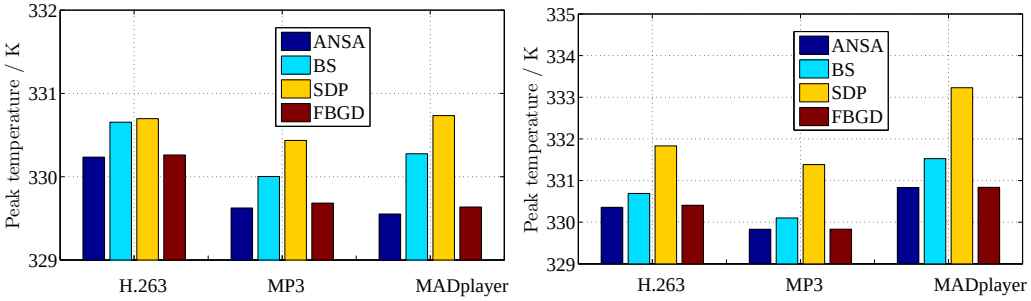


(a) Intel SCC: 3-stage and step size: 4ms (b) Intel SCC: 4-stage and step size: 4ms

Figure 3.11: Peak Temperature produced by the tested approaches with $\delta = 1$ and step size is $4ms$ when the three applications are executed on platform SCC with different stage numbers.

between $[4.2, 5.6]ms$ and the application is activated every $100ms$. The deadline D is set as $D = 40 + 5 \times n$ to comprise the WCET consumed in the new deployed stage. The results, including the time expense and the peak temperature, are shown in figures from 3.13 to 3.15. Due to that SDP and BS suffer from exploring space explosion as stage number increases, we terminate their simulation when n reached 7 and 11, respectively.

We first examine the computing time required by the four approaches, as shown in Fig. 3.13a and Fig. 3.15. Observe that the time required by FBGD is generally the least and the curve is nearly flat as n increases, which indicates FBGD is feasible for pipelined systems with many stages. We note that the cost of ANSA varies in a certain range

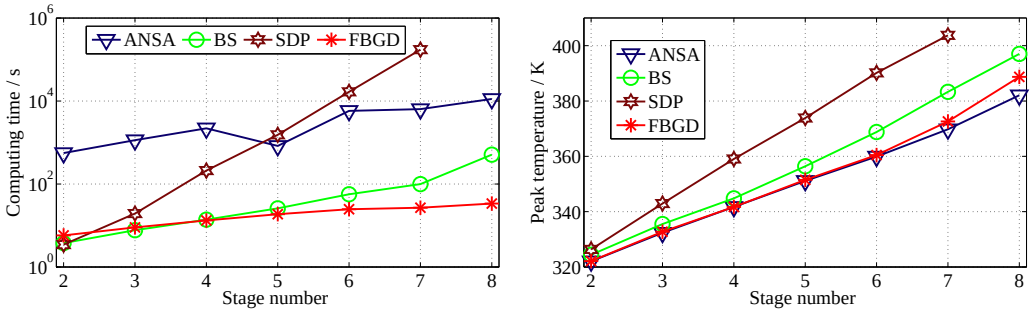


(a) Intel SCC: 3-stage and step size: 2ms (b) Intel SCC: 4-stage and step size: 2ms

Figure 3.12: Peak Temperature produced by the tested approaches with $\delta = 1$ and step size is $2ms$ when the three applications are executed on platform SCC with different stage numbers.

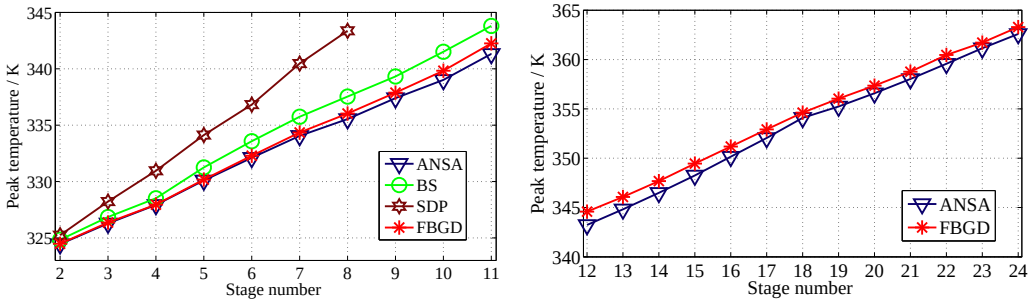
and is about two-magnitude larger than that of FBGD in most cases. It is expected since ANSA calculates the peak temperature by algorithm ANPT, which is more accurate and computation expensive. The figures also show that the computing time consumed by SDP grows exponentially as stage number grows and becomes the highest one when $n \geq 6$. This is because SDP examines all the possible deadline partitions, the amount of which increases exponentially as the stage number increases. Moreover, computing the service demand for every following stage requires numerical min-plus convolution, which incurs significant computation and memory overhead. Similarly, we find that the time overhead of BS grows exponentially as stage number increases. Therefore, we can say that SDP and \are not scalable with the stage number regarding the requirement for computing resource.

Then, we discuss the results of the four approaches in peak temperature. (1) Fig. 3.13b and Fig. 3.14 demonstrate that SDP offers the highest temperature in all the cases. Similarly the temperature difference between SDP and our approaches widens as stage number increases, which is expected because SDP pays burst more times and therefore generates PTM schemes having larger $\mathbf{t^{on}}$ partition. (2) We can clearly see that the peak temperatures generated by FBGD are always lower that from BS, which further strengthes the effectiveness of FBGD. (3) Approach ANSA gives better results than FBGD when the stage number is larger than 6, otherwise it offers almost the same result with FBGD. Therefore it can be concluded that ANSA is more suitable for large stage number scenarios while FBGD is optimal in small stage number cases. (4) Again, notice that the peak temperature gap between different approaches is bigger



(a) Time expense (b) The best peak temperature

Figure 3.13: The results of the four approaches on ARM from 2-to 8-stage.



(a) 2- to 11- stage cases (b) 12- to 24- stage cases

Figure 3.14: The best peak temperature generated by the four approaches on SCC from 2 to 24 stages.

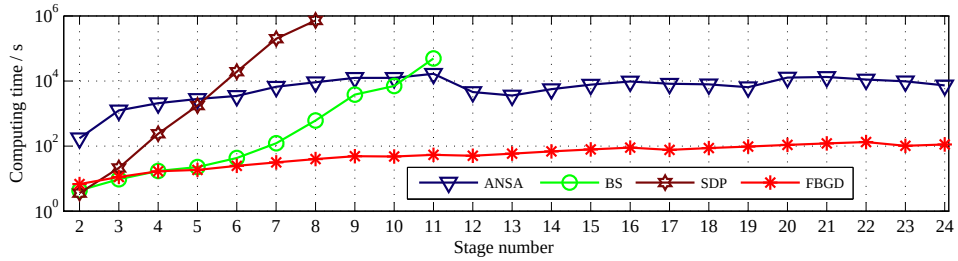


Figure 3.15: The time expense of the four approaches on SCC from 2 to 24 stages.

on platform ARM than SCC, as explained above.

In conclusion, compared to SDP, the proposed approaches FBGD and ANSA can generate much lower peak temperature. Moreover,compared

to approach FBGD, ANSA reduces the peak temperature up to 6.5K on the 8-stage ARM platform as well as nearly 2K on SCC platform. FBGD and ANSA are also proved to be scalable with the stage number.

3.10 Summary

In this chapter we have proposed a new approach to minimize the peak temperature of a pipelined hard real-time system by reversely utilizing the Pay-Burst-Only-Once principle. Two algorithms, FBPT and ANPT, are presented to calculate the peak temperature with different levels of accuracy and complexity. Then, our problem is transformed to an optimization problem which is broken down by a set of sub-problems. Based on the two peak temperature methods, two algorithms are proposed to solve the problem: the FBPT based one offers an approximated solution but is faster while the ANPT based one is timing expensive but gives a more accurate solution. We conduct simulations of our approaches on two actual platforms for real life applications and the results show that our approaches can reduce the peak temperature more efficiently than the approach without PBOO, especially for many-stage scenarios. It is also shown that the time expenses of our two algorithms grow slowly as the stage number increases, which indicates the algorithms are scalable with the number of stages.

In next chapter, we will discuss an adaptive thermal management approach which controls the execution of events according to the real event arrivals and execution times instead of the worst-case assumptions.

Chapter 4

Adaptive Periodic Thermal Management

The thermal management approaches investigated in previous chapters have one common feature: they are all static methods, i.e, the optimal PTM schemes are calculated in design phase. To provide hard real-time guarantee, the static PTM approaches make worst-case assumptions in below aspects.

- The execution time of the events.
- The arrival pattern of the events.

In other words, the static approaches consider the events arrive strictly in accordance with the upper arrival curve and each event requires the worst-case execution time to finish. These worst-case scenarios, however, rarely happen in real world. Thus they could offer pessimistic results due to those conservative assumptions. Therefore, to effectively optimize the peak temperature of real-time systems, new approach should consider the runtime variabilities of event arrivals and execution time. Such approach is recognized as an adaptive approach.

The challenges of designing an adaptive thermal approach are how to effectively extract the runtime variabilities of events and then adopt them to reduce the peak temperature. To overcome these challenges, three basic questions should be answered: (1) *what* kind of information about the system and events should be gathered such that the runtime variabilities can be adequately represented? (2) in *what* pattern the system behaves between two adaption instants? (3) *how* to use the gathered information to derive thermal management schemes at each adaption instant such

that the deadline constraints can be met while the peak temperature is optimized? Note that the efficiency is also important since the approach works in online manner and thus introduces overhead, which hampers the final effectiveness.

4.1 Overview

In this chapter, we address the aforementioned concerns and propose the Adaptive Periodic Thermal Management (APTM) to optimize the peak temperature of pipelined multi-core systems under real-time constraints. The concepts of arrival curve and service curve [96, 60, 98] are adopted as the workload and service model such that our approach can handle general event arrivals. Our approach APTM is an online and offline combined approach. With the thermal property knowledge of the processor obtained from offline simulation, APTM works in online manner to manage the stages at each adaption instant.

The considered system handles applications can be split into sub-tasks. Each core has three power consumption modes, namely, 'active', 'idle', and 'sleep'. At each adaption instant, an APTM scheme is applied to each core in the pipeline until next adaption instant. Each APTM scheme is specified by a pair of parameters (t_i^{on}, t_i^{off}). Given an APTM scheme, the core first switches to 'sleep' state and stay for t^{off} time units, whichever state it is currently at. Then it switches on and off periodically to control the temperature according to the scheme until next adaption. When the core is in 'active' mode but has no event to handle, it automatically switches to 'idle' state to save energy.

At each adaption instant, the dynamic counter technique [59] is adopted to give precise prediction of future event arrivals based on arrival history. The states of the FIFO (First In First Output) buffers between cores, which reflect actual event executions, are also collected and utilized in our approach. The gathered dynamic information is then utilized to derive the APTM schemes. According to the information, our approach first determines which stages can adopt APTM schemes and which should be always turned on until next adaption such that the real-time constraints are satisfied. Then, the APTM schemes for the feasible stages are calculated. Moreover, to minimize the peak temperature, the unique thermal properties of the stages are also used during the calculation. Two thermal curves, i.e., the `warming curve` and the `cooling curve` are proposed to model the thermal properties of each stage in

different scenarios. Several light-weight algorithms are presented to determine APTM schemes at online adaption instants. Moreover, an offline algorithm is also given to search the key parameter of online adaption.

The rest of this chapter is organized as follows: The related work is briefly introduced in section 4.2. Section 4.3 describes our system models and presents problem statement. A motivation example is presented in section 4.4. Section 4.5 presents backgrounds of our approach. Then, the real-time constraints are analyzed in section Section 4.6. The heuristic scheme is discussed in section 4.7 and section 4.8. Section 4.9 presents the evaluation of APTM. Section 4.10 concludes.

4.2 Related works

Pipelined multi-core architecture has been widely adopted for high performance. Several approaches can be found on thermal optimization of pipelined systems. Cox et al. proposed a fast thermal-aware approach for streaming applications based on a 3D MPSoC model under the throughput constraints in [32]. The mapping of the multiple applications is determined at design time such that the peak temperature is minimized under throughput constraints. This approach is based on task-mapping technology and assumes periodic task model. Cheng et al. presented an offline approach to minimize the peak temperature of pipelined systems under real-time constraints in [2]. The approach computes a PTM scheme for each stage to determine the switch on/off pattern during design phase. The above two approaches search the optimal solution that minimizes the peak temperature in offline manner, considering the worst-case execution time and worst-case event arrivals, which rarely happen in real systems. Thus their results could be pessimistic due to the runtime variability of event arrivals and execution time. There are also several approaches on this topic. However, they either don't consider hard deadline constraints [3] or just reduce the deadline miss percentage into a low range [72]. Designed for hard real-time systems, our approach can guarantee the worst-case delay is no larger than the deadlines of the tasks.

Chen et al. explored how to apply dynamic power management in adaptive manner to optimize leakage power consumption for pipelined multi-core systems under deadline constraints in [24]. As aforementioned, an effective power management may not be suitable for thermal management. In addition, the proposed approach has one major drawback.

At each adaption instant, the approach computes a set of time lengths for which the stages are allowed to sleep. After adaption, the stages sleep for the corresponding time length, and then stay active until next adaption instant. This simplifies the real-time analysis during online execution. However, this method demands a high adaption frequency, in other words, short adaption intervals. Otherwise, the results get worse very quickly since the stages could be active unnecessarily between two adaption instants. In our approach, the PTM schemes are adopted to the stages during adaption such that the results is much less influenced by the adaption period. Therefore, our approach can still offer acceptable results with large adaption periods.

There has been significant work on thermal management of multi-core architectures. The concept of Thermal-Resiliency is extended to multi-core systems by Pradeep et al. in [45]. Combining a control-theoretic framework, they proposed an approach which can maintain thermal constraints and provide hard real-time guarantees. In [103], Wang et al. addressed the problem of minimizing the peak temperature of a real-time application executed on multi-core platforms. Three computationally efficient algorithms are presented for deploying applications to individual devices. Above two approaches both assumed simple task models, i.e., either periodic tasks or sporadic task model. The authors of [79] addressed the DVS scheduling problem on multicore systems under both temperature and energy constraints. Since the problem is NP-hard, two algorithms, an accurate one and an approximated one, are proposed to give results in different levels of accuracy. A stochastic thermal control approach is proposed in [69] to reduce the chip-wide temperature gradient of a multi-core processor handling multiple stochastic real-time task streams. The technique of job migration among active and passive cores of the stream is adopted to reduce the chip-wide temperature gradient. While making great contributions to the field, the above two approaches cannot provide hard real-time guarantees.

To model general event arrivals, Real Time Calculus [96] are proposed to abstract task arrivals in time domain into arrival curve. In [78], Perathoner et al. presented an adaptive scheme for the scheduling of arbitrary event streams by combining optimistic and pessimistic DVFS scheduling. Adaptive online power managements have also been proposed in [49, 59] to adaptively reduce the power consumption of the processor by procrastinating the processing of arrived events as late as possible. The above approaches, unfortunately, are designed for single core processors and cannot be applied to multi-core architectures. Au-

thors of [58] introduced an online DVFS management scheme for multi-core processors running hard real-time tasks. The technique in [59] is adopted to predict future event arrivals based on the arrival history. Although aiming at minimizing the temperature, the online DVFS approach doesn't consider any temperature feedback or thermal property of the cores, which can help to reduce the temperature further. In contrast, our approach implements Dynamic Power Management to manage temperature utilizes the sampled temperature and the unique thermal properties of each core in determining PTM schemes to obtain lower peak temperature.

The related work is briefly introduced in this section. In next section, the definitions and notations that are used in this chapter are presented.

4.3 system model

Notation: In this chapter, matrices and vectors are represented by bold characters.

4.3.1 Hardware and Thermal Model

In this chapter, we consider the pipelined multi-core system adopted in Chapter 3. Therefore, we also have the following constraint due to mode-switching overhead.

$$t^{off} > t^{swoff} \tag{4.1}$$

In addition to the hardware model, the thermal and power models in Chapter 3 are also used to get the temperature evolution.

It is worth noting that the implementation our approach is not limited by such thermal and power models. In runtime, our approach only needs the temperature of the stages to make scheduling decisions. For processors with physical thermal sensors, the temperature can be obtained by reading the sensors. For processors without hardware thermal sensors on each core, soft thermal sensors [62] can be employed to estimate the temperature of a single core.

4.3.2 Adaptive Periodic Thermal Management

In this chapter, we study how to minimize the peak temperature for coarse-grained pipelined multi-core processors. An online approach

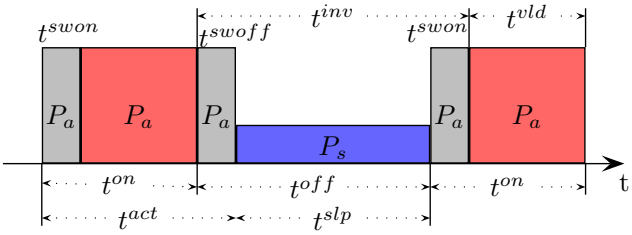


Figure 4.1: The adaptive periodic thermal management schemes after two adaption instants.

named Adaptive Periodic Thermal Management is proposed to adaptively switch the cores to 'sleep' in the run time.

At each adaption instant, an APTM scheme is applied to each core in the pipeline and is updated at the next adaption instant. The APTM scheme applied to the stage $\mathbb{P}_i$ is specified by a pair of parameters (t_i^{on}, t_i^{off}). The period is calculated as $t_i^{prd} = t_i^{on} + t_i^{off}$. For brevity, $\mathbf{t^{on}}$ and $\mathbf{t^{off}}$ denote the vectors $[t_1^{on}, t_2^{on}, \cdots, t_n^{on}]$ and $[t_1^{off}, t_2^{off}, \cdots, t_n^{off}]$, respectively. Given an APTM scheme, whichever state it is currently at, stage $\mathbb{P}_i$ first switches to 'sleep' state and stays for t_i^{off} time units. Then, it switches to 'active' or 'idle' state and keeps for t_i^{on} time units. This procedure repeats until the next adaption instant. An example of APTM scheme is demonstrated in Fig. 4.1. Note that due to the switching overhead, the valid/invalid time interval in each period for handling workload should be revised as:

$$t^{vld} = t^{on} - t^{swon} \tag{4.2}$$
$$t^{inv} = t^{off} + t^{swon} \tag{4.3}$$

In addition, we define the valid partition as:

$$K^{vld} = t^{vld} / t^{prd} \tag{4.4}$$

4.3.3 Problem Statement

At each online adaption instant, our approach should offer an APTM (t_i^{on}, t_i^{off}) scheme to each stage $\mathbb{P}_i$ in the pipelined system. The values of t_i^{on} and t_i^{off} should be chosen prudently. First, it's clear that the peak temperature can be reduced by increasing the sleep interval

or decreasing the active interval. However, this action should be done carefully, otherwise the deadline constraint of current or future events may be violated. Moreover, the sleep intervals for all stages are not independent with each other due to deadline constraints. The influence of such interaction between two stages on the temperature distribution should be handled carefully. Finally, increasing the sleep interval length under real-time constraints causes the active interval to increase simultaneously. Thus, setting a large t_i^{off} results in a large t_i^{on}, which may raise the peak temperature. Therefore, at each adaption instant, for each stage, our approach should (1) calculate the real-time constraints to $\mathbf{t^{on}}$ and $\mathbf{t^{off}}$; (2) properly select and balance t_i^{on} and t_i^{off} for all stages under the real-time constraints so that the peak temperature can be minimized.

In conclusion, the problem can be formally defined as: *For a given n-stage pipelined platform modeled in Section 4.3, a real-time application specified by* **TASK**, *at each adaption instant during runtime we should determine the* APTM *schemes applied to all stages so that the peak peak temperature of the processor is minimized and the worst-case end-to-end delay of any task is less than the deadline D.*

4.4 Motivation of Our Work

In this section, the motivation of our approach is presented by comparing it with an offline approach in a concrete example.

In contrast to the approach Offline Pay-Burst-Only-Once (O-PBOO) in Chapter 3, which searches the optimal ($\mathbf{t^{off}}$, $\mathbf{t^{on}}$) in offline manner and then applies them to the system, APTM adaptively applies $\mathbf{t^{on}}$ and $\mathbf{t^{off}}$ to the stages according to the current workload and the states of processors. Compared to O-PBOO, APTM utilizes the following two facts to reduce the temperature.

`Dynamic slack` An offline approach analyzes the pipelined system by considering the worst-case pattern of event arrivals, i.e., the upper bound $\alpha^u(\Delta)$ and the lower bound $\alpha^l(\Delta)$, which represent the maximal and minimal number of event arrivals in any time interval with length Δ, respectively. However, the worst-case scenarios rarely happen. In reality, events arrive with a variable delay bounded by the arrival curve $\alpha(\Delta)$. For example, consider a task specified by period $p = 50$ ms and jitter $j = 50$ ms. The worst-case event arrival is a burst with two events released at the same time. However, in reality, for example, events may arrive at

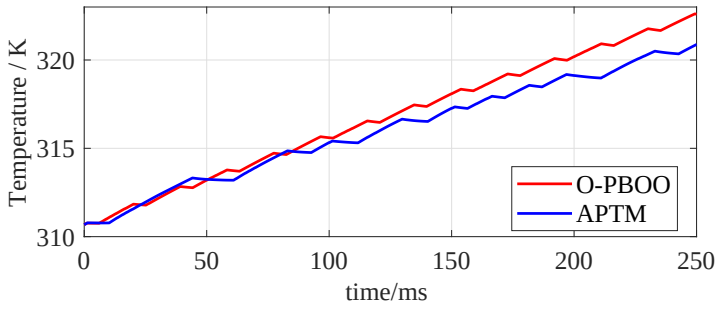


Figure 4.2: The temperature of the first core in the ARM 3-stage platform when the two methods are applied to manage it.

time $[0, 40, 110, 150, 210, 230]$ ms. We term the difference between the worst-case and real event arrivals as the `dynamic slack`.

`Execution slack` When performing offline analysis, static approaches used the worst-case execution time (WCET) to bound the execution time of a task on the target system. However, due to the inherent variability of execution time, most events, in reality, finishes before their WCETs. The fact is termed as `execution slack`, which occurs as a result of the difference between the real execution time and WCET of an event.

By recording the history of event arrivals and monitoring the filling level of FIFOs between stages, APTM can effectively utilize above slacks and achieve significant temperature reduction compared to offline approaches. Now, we show the advantage of APTM by a concrete example.

In the example, O-PBOO and APTM are applied to manage the temperature of a pipelined system with three stages. The task is specified by period $p = 50$ ms and jitter $j = 50$ ms. The worst-case execution times on all stages are $\mathbf{c} = [4, 4, 4]$ ms. We assume the deadline of the task equals its period. Events are released at time $[0, 0, 50, 100, 150, 200, 250]$ ms. The real execution times of the events on three stages are set as the random numbers between 1.6 ms and 4 ms. The adaption periods of APTM is set as 50 ms.

We run the two approaches for the task trace and obtain the temperature evolutions. The temperature of the first stage is depicted in Fig. 4.2. For a clean figure, we consider the power dissipations in 'idle' state and 'active' are the same in the simulation. Since we only want to compare the relative performance of the two approaches, this simplification is acceptable. As displayed, APTM offers the lowest temperature after three

adapts at time $t = 100$ ms. The higher temperature given by O-PBOO is caused by considering the worst-case task timing parameters in design phase. In other words, O-PBOO must ensure the end-to-end deadlines are met with the assumption that all events require WCET to finish and the event arrival burst may happen at anytime during real execution. We also extend the task trace to 60 seconds and run both approaches again. The final peak temperatures of O-PBOO and APTM are 391 K and 375 K, respectively. This further strengthens our observation.

In summary, utilizing and managing aforementioned slacks, APTM can significantly reduce the peak temperature, which is the main motivation of our work. In the next section, we discuss how the `dynamic slack` and `execution slack` are explicitly utilized and managed by our approach.

4.5 Utilizing the Two Slacks

In Section 4.4, we investigated the facts of `dynamic slack` and `execution slack`. In this section, we present how to effectively utilize them in APTM.

4.5.1 Demanded Service Of Unfinished Events

Our approach utilizes `execution slack` by monitoring the numbers of unfinished events in the FIFOs as well as the states of the ongoing job in each stage. With the obtained dynamic information, we get the demanded service of unfinished events.

Unfinished events could be stored in FIFOs or the core processing it at an adaption instant. Let $ET_i(t)$ represent the union set of events stored in $FIFO_i$ and the core on stage $\mathbb{P}_i$ at time t. The number of events in $ET_i(t)$ is denoted as $|ET_i(t)|$. It is worth noting that, although the absolute deadlines for events in $ET_i(t)$ do not change, their relative deadlines should be revised according to the relative distances between absolute deadlines and time t. Assume events in $ET_i(t)$ are ordered from the earliest absolute deadline to the latest. Then the event stored in the core should be the front of the queue. Moreover, its unfinished part can be upper bounded by:

$$\delta_i(t) = \frac{c_i - t_i^{exe}(t)}{c_i}. \tag{4.5}$$

where c_i is the WCET of the job on stage $\mathbb{P}_i$ and $t_i^{exe}(t)$ is the time length for which the event has already been executed. We represent the de-

manded service curve of set $ET_i(t)$ by notation $\alpha_i^{dm}(t, \Delta)$, which denotes the event-based service demanded by $ET_i(t)$ in time interval $[t, t+\Delta]$ to meet their deadlines. Then, $\alpha_i^{dm}(t, \Delta)$ of $ET_i(t)$ can be given as:

$$\alpha_i^{dm}(t, \Delta) = \begin{cases} j - 1 + \delta_i(t), & D_{i,j} \leq t + \Delta \leq D_{i,j+1} \\ |ET_i(t)| - 1 + \delta_i(t), & t + \Delta > D_{i,|ET_i(t)|} \end{cases} \tag{4.6}$$

where $D_{i,j}$ is the absolute deadline of the j^{th} event in $ET_i(t)$.

Note that the value of $t_i^{exe}(t)$ can be obtained from the execution time monitoring ability of the system. Such functionality is already commonly available on many real-time platforms [10].

4.5.2 Arrival Curve of Future Events $\alpha^{fu}(t, \Delta)$

To effectively exploit the `dynamic slack`, the dynamic counter proposed in [59] is adopted to predict future event arrivals. According to [59], at time t, the number of event arrivals in time interval $[t, t+\Delta]$ can be bounded by $\alpha^{fu}(t, \Delta)$, which is calculated based on the original arrival curve $\alpha(\Delta)$ and the event arrival history upon t. It's worth noting that dynamic counter can be easily implemented as part of the hardware with negligible overhead [48, 59].

4.6 Proposed Approach

In this section, we present our adaptive periodic thermal management approach to reduce the peak temperature of pipelined multi-core processors. At one adaption instant for decisions, APTM should compute $\mathbf{t^{on}}$ and $\mathbf{t^{off}}$ that guarantee the timing constraints of unfinished events as well as future events. To model the system containing unfinished events, we first transform it to a multi-stream system in which the unfinished events in each stage are modeled as a separate stream. With the transformed system, the end-to-end service provided to each stream by the system under APTM can be calculated based on the extended pay-burst-only-once principle. Then a set of constraints is applied to $\mathbf{t^{on}}$ and $\mathbf{t^{off}}$ to satisfy the deadlines of unfinished events as well as future events. Finally, we present a heuristic scheme to solve the thermal optimization problem. Several lightweight algorithms are proposed to computes $\mathbf{t^{on}}$ and $\mathbf{t^{off}}$ which minimize the peak temperature of the pipelined system.

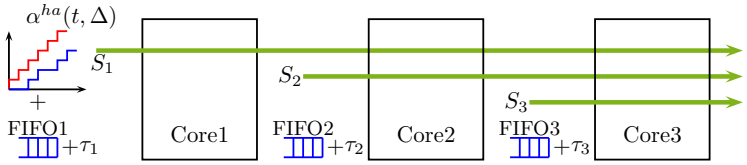


Figure 4.3: An example of the transformation of a 3-stage pipelined multi-core system

4.6.1 System Transformation

At each adaption instant, our approach must offer APTM schemes satisfying the end-to-end deadline constraints of new arriving events and the unfinished events stored in the system. For an offline approach, the Pay-Burst-Only-Once principle can be adopted in real-time analysis. However, as pointed in [37], the unfinished events prevent our approach from utilizing the Pay-Burst-Only-Once principle directly during online adaption. As already demonstrated in [2, 23], approaches without using Pay-Burst-Only-Once account for the burst in the original stream more than once and requires lots of computation, thus leading to pessimistic results. In this chapter, we transform the system to an alternative formation with which the time properties of the system can be analyzed effectively.

For an n-stage pipelined multi-core system, the transformation is done by modelling the unfinished events $ET_i(t)$ in stage $\mathbb{P}_i$ as an individual input stream S_i with demand curve $\alpha_i^{dm}(t,\Delta)$ passing through the following stages, indexed by $J_i = \{i, i+1, \cdots, n\}$. Specifically, the input stream of first stage comprises the new arriving events and unfinished set $ET_1(t)$. An example of transformation is shown in Fig. 4.3. Moreover, let set $J_{i,k}(k > i) = \{k, k+1, \cdots, n\}$ represent all the stages that are traversed both by stream S_i and S_k. Finally, notation $F_i = \{i+1, \cdots, n\}$ denotes the indexes of streams that are interferenced by stream S_i. For instance, considering the example in Fig. 4.3, we have $J_1 = \{1,2,3\}$, $J_{1,2} = \{2,3\}$ and $F_1 = \{2,3\}$.

4.6.2 Real-Time Constraints

Now, we analyze how to guarantee the deadline requirements for unfinished as well as future events at each adaption instant. It's worth noting that the service-providing-ability of the pipelined system is directly determined by t^{vld}s and t^{inv}s, instead of t^{on}s and t^{off}s. Thus, we

use t^{vld}s and t^{inv}s in following analysis and then retrieve back t^{off}s and t^{on}s according to (4.2) and (4.3).

For the aforementioned system transformation, the EPBOO principle in [37] and the theory in [24] can calculate the aggregate service curve provided for stream S_i for scenarios in which every stage provides a service curve in *rate-latency* format. However, under APTM, the service curve of each stage is no longer rate-latency. In addition, the time overhead of mode-switching is also not considered in their work. Therefore, the results in [37, 24] cannot be utilized in our work. In this section, we present a new theorem which lower bounds the end-to-end service curve for stream S_i in the transformed system under APTM.

Lemma 4.1 *At an adaption point, suppose the* APTM *schemes specified by* ($\mathbf{t^{on}}$, $\mathbf{t^{off}}$) *are applied to the system. Denote the end-to-end service curve provided to a stream S_i as β_i^{ete}, if the valid time length in each period t_i^{vld} is positive integer times of c_i, we have*

$$\beta_i^{ete}(\Delta) \geq \kappa_i(\Delta - \sum_{j=i}^{n}(t_j^{inv} + c_j) - \sum_{j=i+1}^{n}\frac{W_j + \delta_j(t)}{\kappa_j}) \tag{4.7}$$

where $\kappa_i = \min_{j\in J_i}(K_i^{vld}/c_i)$, W_i is the number of events stored in $FIFO_i$, and $\delta_i(t)$ is calculated from (4.5).

Proof From the existing result presented in [37] [1], one can obtain (4.8) for the stream of interest S_i with any time instants satisfying $t_1 \leq t_2 \leq \cdots t_{n+1}$.

$$R_i^{n+1}(t_{n+1}) - R_i^1(t_1) \geq \sum_{j\in J_i}\beta_j^l(t_{j+1} - t_j) - \sum_{k\in F_i}\sum_{j\in J_{i,k}}(R_k^{j+1}(t_{j+1}) - R_k^j(t_j))$$

where $R_i^j(t)$ is the workload function of S_i at j^{th} stage and denotes the number of events in stream S_i that arrive at the j^{th} stage in time interval $[0, t)$. Specifically, $R_i^{n+1}(t)$ is the workload function of the output of stream S_i.

From the definition of F_i and $J_{i,k}$, one can easily determine that:

$$\sum_{j\in J_{i,k}}(R_k^{j+1}(t_{j+1}) - R_k^j(t_j)) = R_k^{n+1}(t_{n+1}) - R_k^k(t_k) \tag{4.8}$$

[1] See equation (14) in [37].

At each adaption instant, the right hand side of (4.8) can be safely bounded by the number of unfinished events in $FIFO_k$ and $\mathbb{P}_k$, i.e., $W_k(t) + \delta_k(t)$ as discussed in section 4.5.1. Therefore,

$$\sum_{j \in J_{i,k}} (R_k^{j+1}(t_{j+1}) - R_k^j(t_j)) \leq W_k(t) + \delta_k(t) \tag{4.9}$$

When t_i^{vld} is positive integer times of c_i, the stair-case service curve $\beta_i^l(\Delta)$ of APTM can be lower bounded by a bounded-delay function [23]:

$$\beta_i^l(\Delta) \geq \max(0,\ \frac{K_i^{vld}}{c_i}(\Delta - (t_i^{inv} + c_i))) = [\frac{K_i^{vld}}{c_i}(\Delta - (t_i^{inv} + c_i))]^+ \tag{4.10}$$

where $[z]^+ = \max(0,\ z)$. Let $bdf_i(\Delta)$ denote $[K_i^{vld}/c_i(\Delta - (t_i^{inv} + c_i))]^+$. Then, combining the definition of F_i, equations (4.8), (4.9) and (4.10), we finally have:

$$R_i^{n+1}(t_{n+1}) - R_i^1(t_1) \geq \sum_{j \in J_i} bdf_j(t_{j+1} - t_j) - \sum_{k=i+1}^{n} (W_k(t) + \delta_k(t)) \tag{4.11}$$

The bounded-delay function bdf_i is actually in the rate-latency format. Finally, following the derivation similar to that in [37], one can finally have inequality (4.7). □

With Lem. 4.1, the sufficient condition for satisfying deadline constraints for all tasks under Adaptive Periodic Thermal Management is given as the following theorem.

Theorem 4.2 *Consider an n-stage pipelined multi-core system modeled in Section 4.3. At an adaption point, the system is transformed into an n-stream system, and the* APTM *schemes specified by (*$\mathbf{t^{on}}$, $\mathbf{t^{off}}$*) are applied to it. Then, the worst-case end-to-end delay of any task is guaranteed to be no larger than its deadline D if the following conditions hold for any stream S_i.*

$$\forall i \leq j \leq n,\ t_j^{vld} = g_j \cdot c_j,\ g_j \in \mathbb{N} \tag{4.12}$$

$$\kappa_i(\Delta - \sum_{j=i}^{n}(t_j^{inv} + c_j) - \sum_{j=i+1}^{n} \frac{W_j + \delta_j(t)}{\kappa_j}) \geq \beta_i^{dmd}(\Delta) \tag{4.13}$$

where $\beta_i^{dmd}(\Delta)$ is the demanded end-to-end service curve of stream S_i to meet deadline constraint:

$$\beta_i^{dmd}(\Delta) = \begin{cases} \alpha^{fu}(t, \Delta - D) + \alpha_1^{dm}(t, \Delta) & \text{if } i = 1 \\ \alpha_i^{dm}(t, \Delta) & \text{if } i \geq 2 \end{cases} \tag{4.14}$$

Proof We first prove (4.14). For the case that $i \geq 2$, it's already proved in section 4.5.1. When $i = 1$, the input workload comprises the new arriving events as well as the unfinished events stored in the first $FIFO$ and stage. As studied in section 4.5.2, the new arriving tasks can be tightly and safely bounded by the arrival curve $\alpha^{fu}(t, \Delta)$. Right shifting $\alpha^{fu}(t, \Delta)$ by deadline D yields the demanded service curve of new arriving tasks. Then, adding it with $\alpha_1^{dm}(t, \Delta)$ gives the demanded service curve for first stage.

Now, we prove the theorem. It is clear that the deadline constraints can be met if the lower end-to-end service curve is no less than the demanded service curve for every stream S_i, i.e., $\beta_i^{ete}(\Delta) \geq \beta_i^{dmd}(\Delta)$. Let $\beta_i^{\diamond}$ denote the left side of (4.13). From Lem. 4.1, once conditions (4.12) and (4.13) are satisfied, we have $\beta_i^{ete}(\Delta) \geq \beta_i^{\diamond} \geq \beta_i^{dmd}(\Delta)$. Therefore, the end-to-end delay of any event can be guaranteed to be no larger than its deadline D. □

Now, let's examine condition (4.13) closely. The left side of (4.13) is actually a bounded-delay-function [1]. For a bounded-delay-function $bdf(\Delta) = [\rho_i(\Delta - b_i)]^+$, given b_i, the minimal slope ρ_i satisfying (4.13) is determined by:

$$\rho_i = \min\{\rho | [\rho(\Delta - b_i)]^+ \geq \beta_i^{dmd}(\Delta)\} \tag{4.15}$$

where $[z]^+ = \max(0,\ z)$. It's worth noting that ρ_i can be obtained efficiently by implementing a binary search. For a pair of b_i and ρ_i obtained from (4.15), it is intuitive that condition (4.13) is guaranteed if the following two constraints hold simultaneously [2]:

$$\kappa_i = \min_{j=i}^{n}(K_i^{vld}/c_i) \quad \geq \quad \rho_i \tag{4.16}$$

$$\sum_{j=i}^{n}(t_j^{inv} + c_j) + \sum_{j=i+1}^{n}\frac{W_i + \delta_j(t)}{\kappa_j} \quad \leq \quad b_i \tag{4.17}$$

Then, the remained problem is how to choose a proper pair of (b_i, ρ_i). This problem is equivalent to determining b_i because ρ_i is given by (4.15). In this chapter, we obtain a sub-optimal b_i by:

$$b_i = \lambda b_i^{max} + (1 - \lambda) b_i^{min} \tag{4.18}$$

where λ is a positive real number less than 1, b_i^{max} and b_i^{min} are the upper and lower bounds of b_i for curve $\beta_i^{dmd}(\Delta)$:

$$\begin{aligned} b_i^{max} &= \max\{b|\ \frac{1}{\max_{j=i}^{n}(c_j)}(\Delta - b) \geq \beta_i^{dmd}(\Delta)\} \\ b_i^{min} &= \sum_{j=i}^{n} c_j + \sum_{j=i+1}^{n}\{(\max_{k=j}^{n} c_j)(W_j + \delta_j(t))\} \end{aligned}$$

The key parameter λ can be determined by offline simulation and set as the one yielding the lowest peak temperature, which is described in algorithm 12. Although this method doesn't offer the global optimal solution, it introduces negligible overhead during online adaption and thus is suitable for our approach. Finally, constraint (4.17) is revised as:

$$\sum_{j\in J_i} t_j^{inv} \leq \bar{b}_i = b_i - (\sum_{j=i}^{n} c_j + \sum_{j=i+1}^{n} \frac{W_i + \delta_j(t)}{\bar{\rho}_j}) \tag{4.19}$$

For brevity, vector $\bar{\mathbf{b}}$ denotes the set $\{\bar{b}_1, \bar{b}_2, \cdots, \bar{b}_n\}$. Then, the corresponding slopes obtained from (4.15) are termed as $\bar{\rho} = \{\bar{\rho}_1, \bar{\rho}_1, \cdots, \bar{\rho}_n\}$.

4.6.3 APTM constraint set

Now, the real-time constraint set for a stream S_i has been derived. This constraint set should hold for all streams in the system. Combining all the real-time constraints and the hardware constraints (4.1), we present the final constraint set for stage $\mathbb{P}_i$ as:

$$t_i^{vld} = g_i \cdot c_i,\ g_i \in \mathbb{N} \tag{4.20}$$

$$t_i^{inv} > t_i^{hdc} \tag{4.21}$$

$$K_i^{vld} \geq K_i^{min} = \max_{j=1}^{i}(\bar{\rho}_j)c_i \tag{4.22}$$

$$\sum_{j\in J_i} t_j^{inv} \leq \bar{b}_i \tag{4.23}$$

Where $t_i^{hdc} = t_i^{swon} + t_i^{swoff}$ is the hardware constraint of $\mathbb{P}_i$. For brevity, we name above constraints APTM constraint set and term it by $\mathbf{AC}_i$ for stage $\mathbb{P}_i$. Moreover, the set $\{t_1^{hdc}, t_2^{hdc}, \cdots, t_n^{hdc}\}$ is denoted by $\mathbf{t^{hdc}}$.

At an adaption instant t^a, our approach should give the pipelined system a set of APTM schemes which can minimize the peak temperature $T^\star$ of

the processor under the APTM constraint sets of all stages, which can be formatted as following problem.

$$\begin{aligned} &\text{minimize} && T^{\star}(\mathbf{t^{on}}, \mathbf{t^{off}}) = \max_{t \geq t^a} \{\max \mathbf{T}(t)\} \\ &\text{subject to} && \mathbf{AC}_i, \quad \forall i \in [1, 2, \cdots, n] \end{aligned} \tag{4.24}$$

The peak temperature $T^{\star}$ can be computed by the algorithm proposed in [2]. However, solving such an optimization problem requires significant computation effort due to two reasons. First, calculating $T^{\star}$ incurs the costly convolution operation as we consider heat influence between two blocks in the thermal model. Second, even with an efficient searching algorithm, exploring the n-dimension search space is still heavy in computing. Therefore, it is infeasible to search the accurate solution of such problem with online adaption. To address this issue, one lightweight heuristic scheme is proposed in next section to offer sub-optimal APTM schemes under hard real-time constraints.

4.7 Online Part

In this section, we present one heuristic scheme that determines APTM schemes online by following the guidance of processor thermal properties obtained from offline experiments. In this way, the online computation of exploring search space is significantly saved. The problem is solved in two steps. First, our approach decides which stages are feasible for APTM schemes. Then, the t^{inv}s and t^{vld}s of these stages are determined according to the thermal properties obtained in offline experiments.

4.7.1 Feasible Stages for APTM

At an adaption instant, it's possible that some stages cannot adopt APTM schemes, otherwise the real-time constraints will be violated. The reason is that hardware-constraint (4.21) may conflict with real-time constraint (4.23). For example, consider a 4-stage pipelined system. Assume constraint (4.23) for stream S_3 is already given as $t_3^{inv} + t_4^{inv} \leq 1.5$. Suppose the time-overheads of mode-switching, t^{swon} and t^{swoff}, are all 0.5 ms. Therefore, constraint (4.21) for stream S_3 is: $t_3^{inv} + t_4^{inv} > 2$ ms. Then, it's impossible to find APTM schemes for both stages $\mathbb{P}_3$ and $\mathbb{P}_4$ simultaneously. In this case, at least one or more stages should always be 'active' or 'idle' state to meet constraints (4.21) and constraint (4.23).

We call the stages in this case the `always working stages`. Besides, the other stages that can adopt APTM schemes are called `APTM-feasible stages`.

We propose algorithm 8 to find the `APTM-feasible stages` at an adaption instant. It's clear that adopting an APTM scheme outperforms staying at 'active' or 'idle' state in lowering the temperature. Therefore, our approach assigns higher priority to stages having higher temperature in adopting an APTM scheme. Basically, this algorithm follows two principles: (1) try to put as many stages to `APTM-feasible stages` as possible, (2) a stage having higher temperature is assigned a higher priority in adopting an APTM scheme. In this way, we can balance the temperature between stages and thus reduce the peak temperature.

Algorithm 8 takes the vector of current temperatures $\mathbf{T}(t)$ of all stages, and the APTM constraint sets of all stages **AC** as input. It returns the two sets of stages: **FS** stands for `APTM-feasible stages` while **AWS** stands for `always working stages`. We first sort the temperatures of stages in the descending order and get the sorted index vector (line 1). We assume all the stages are `always working stages` in the beginning (line 2). Note t_i^{delay} represents the required switching overhead to 'active', t_i^{swon}, if the corresponding stage is not in 'active' or 'idle' state at the adaption instant, otherwise $t_i^{delay} = 0$. Then, from the stage with the highest temperature, the hardware constraint (4.21) is checked with respect to the real-time constraint (4.23) of previous stages to determine if the stage is APTM-feasible (lines 4 and 5). Since the first checked stage has the highest priority in adopting APTM, the current stage is put into set **FS** if there is no conflict (line 6). If there is a conflict, it means the stage certainly cannot adopt an APTM scheme, we put it in **AWS** and set the corresponding t_i^{inv} as t_i^{delay}(line 6).

Then, constraint (4.23) in APTM constraint set of stage $\mathbb{P}_i$ is revised as

$$\sum_{j \in J_i \bigcap \mathbf{FS}} t_j^{inv} \leq \bar{b}_i - \sum_{j \in J_i \bigcap \mathbf{AWS}} t_j^{inv} \tag{4.25}$$

Let $\hat{b}_i$ denote the right side of inequality (4.25). Note that for different stages, the left side of constraint (4.25) can be same with each other. For example, consider a 3-stage system, and set $\{\mathbb{P}_1, \mathbb{P}_3\}$ is the `APTM-feasible stages` while $\mathbb{P}_2$ is the `always working stage`. Then we can simplify the total three inequalities into two inequalities $t_1^{inv} + t_3^{inv} \leq \hat{b}_1$ and $t_3^{inv} \leq \min(\hat{b}_2, \hat{b}_3)$. In conclusion, constraints (4.25) for all the n stages

Algorithm 8 Determine `APTM-feasible stages`.

Input: $\mathbf{T}$, $\mathbf{AC} = \{\mathbf{AC}_1, \mathbf{AC}_2, \cdots, \mathbf{AC}_n\}$
Output: **FS**, **AWS**
1: sort $\mathbf{T}$ in descending order, save the sorted indexes in $\mathbf{I}$
2: $\mathbf{t^{inv}} \leftarrow \mathbf{t^{delay}}$.
3: **for** i in vector $\mathbf{I}$ **do**
4: $\quad t_i^{inv} \leftarrow t_i^{hdc}$
5: $\quad$ *feasible* $\leftarrow$ whether $\sum_{j \in J_k} t_j^{inv} \leq \bar{b}_k$ holds $\forall k \in \{1, 2, \cdots, i\}$
6: $\quad$ *feasible* ? $\mathbf{FS} \leftarrow \mathbf{FS} \bigcup i : \mathbf{AWS} \leftarrow \mathbf{AWS} \bigcup i \,\&\, t_i^{inv} \leftarrow t_i^{delay}$
7: **end for**

can be simplified into n_f inequalities:

$$\sum_{j \in \mathbf{FS}(k)} t_j^{inv} \leq \Theta_k, \ \forall k = 1, 2, \cdots, n_f \tag{4.26}$$

where integer n_f represents the number of total `APTM-feasible stages`, $\mathbf{FS}(k)$ denotes the max k elements in the set **FS**, and constant Θ_k is obtained from merging different $\hat{b}_k$s by `min` operation. For the above example, we have $n_f = 2$, $\mathbf{FS} = \{1, 3\}$, $\mathbf{FS}(1) = \{3\}$ and $\mathbf{FS}(2) = \{1, 3\}$. Moreover, we have $\Theta_1 = \min(\hat{b}_2, \hat{b}_3)$ and $\Theta_2 = \hat{b}_1$. Finally, APTM constraint sets (constraints (4.20) to (4.23)) for `APTM-feasible stages` can be transformed into following constraints.

$$\sum_{j \in \mathbf{FS}(k)} t_j^{inv} \leq \Theta_k, \qquad \forall k = 1, 2, \cdots, n_f \tag{4.27}$$

$$t_i^{vld} = g_i \cdot c_i, \ g_i \in \mathbb{N}, \qquad \forall i \in \mathbf{FS} \tag{4.28}$$

$$t_i^{inv} > t_i^{hdc}(t) \text{ and } K_i^{vld} \geq K_i^{min}, \qquad \forall i \in \mathbf{FS} \tag{4.29}$$

4.7.2 APTM schemes for `APTM-feasible stages`

Now, we only need to consider the above constraints for `APTM-feasible stages`. In this section, we study how to determine APTM schemes for the `APTM-feasible stages`.

Let us examine constraints (4.27)–(4.29) carefully. One can observe that (4.28) and (4.29) are simply a set of constraints for only one stage, while (4.27) comprises n_f inequalities and each of them involves the sleep intervals of k stages. This indicates, as mentioned in Section 3.3, the sleep intervals of the stages can be influenced by each other. Computing the

APTM schemes for all stages in **FS** simultaneously under above interrelated constraints is time consuming and complicated to analyze. To simplify the problem, we calculate APTM schemes in a recursive manner. Each iteration, we first find the minimal value, Θ_{kmin}, in the set of Θ_i for the left stages in **FS**. Then we determine APTM schemes only for the stages in $\mathbf{FS}(kmin)$. Since Θ_{kmin} is the minimal value, the other constraints in (4.27) are certainly satisfied and thus can be safely ignored. The final step at each iteration is removing already determined stages from the left stages in **FS**. The iteration continues until there is not stages left in **FS**.

Let **G** denote the set of stages in $\mathbf{FS}(kmin)$ at each iteration. Next, we investigate how to assign t^{inv}s and t^{vld}s for stages in **G**. The considered constraints are:

$$\sum_{i \in \mathbf{G}} t_i^{inv} \leq \Theta_{kmin} \tag{4.30}$$

$$\forall i \in \mathbf{G}: \; t_i^{inv} > t_i^{hdc}(t), \; K_i^{vld} \geq K_i^{min}, \; t_i^{vld} = g_i c_i, \; g_i \in \mathbb{N} \tag{4.31}$$

According to constraint (4.31), the valid time length t_i^{vld} should be integer times of c_i: $t_i^{vld} = g_i c_i$. Suppose all the g_i are already known, then, it's intuitive that the ideal solution for corresponding t_i^{inv} is $t_i^{inv} = t_i^{idl} = t_i^{vld}/K_i^{min} - t_i^{vld}$, since this makes the valid partition K_i^{vld} equal its lower bound K_i^{min}, which means the lowest peak temperature for the same t_i^{vld}. However, the t_i^{inv}s obtained by this method must meet the real-time constraint (4.30), that is:

$$\sum_{i \in \mathbf{G}} t_i^{inv} = \sum_{i \in \mathbf{G}} g_i c_i (1/K_i^{min} - 1) \leq \Theta_{kmin} \tag{4.32}$$

As smaller g_i means smaller left hand side of (4.32), we can get the minimal value of the left hand side $\mathcal{S}_{min} = \sum_{i \in \mathbf{G}} c_i (1/K_i^{min} - 1)$, when all valid time lengths equal their corresponding WCET, i.e., $g_i = 1$, $\forall i \in \mathbf{G}$. Then, we discuss the problem in two cases: (A) $\mathcal{S}_{min} \leq \Theta_{kmin}$; (B) $\mathcal{S}_{min} > \Theta_{kmin}$.

Case A

In this case, at least one set of g_i can be found for each stage in **G** such that the constraint (4.32) is met. Therefore, all t^{inv}s can be set as their ideal solutions for low peak temperature. Then, the problem is solved once the set of g_i is determined. Setting the set of g_i must

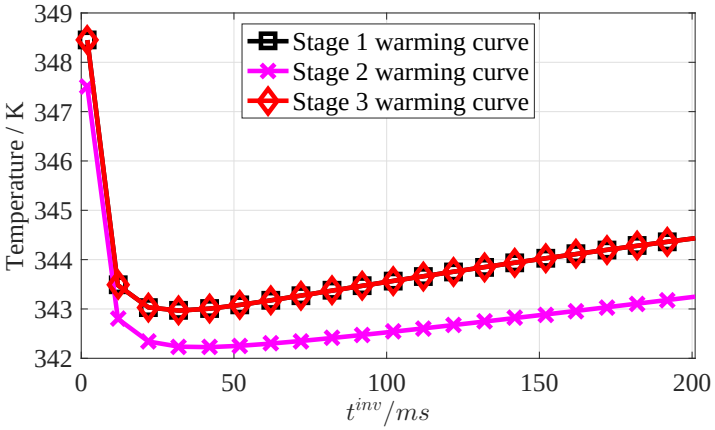


Figure 4.4: The `warming curves` of the stages in a 3-stage pipelined system. The active partition of all the stages are set as $K_i^{vld} = 0.75$.

be done carefully since very large or small values of g_i can cause high peak temperature. For very large values of g_i, the 'active' or 'idle' state lasts very long in each period, which may cause high temperature at the end of valid time interval. On the other hand, smaller g_i means higher switching frequency, which involves more switching overheads and thus also may cause high peak temperature. Thus, setting the set of g_i must be done carefully. In this case, we adopt the `warming curve` of the stages as the guidance in determining the set of g_i.

Definition 4.3 (warming curve) *the `warming curve` of stage $\mathbb{P}_i$ with respect to an active partition K_i^{min} is defined as the relationship between peak temperature $T_i(t_i^{inv}, K_i^{min})$ and the sleep interval t_i^{inv} when, (1) the active partition $K_i^{vld} = K_i^{min}$, that is, $t_i^{vld} = \frac{K_i^{min}}{1-K_i^{min}} t_i^{inv}$, and (2) other stages always stay at 'sleep' state.*

Generally, the `warming curve` models the individual influence of applying identical t^{inv} on each stage to the peak temperature when K^{vld} satisfies certain conditions. An example of `warming curve` can be found in Fig. 4.4. One can observe the curves of two stages could be different. To save memory and calculation during online adaption, the piecewise linear approximations of the `warming curves` are used by our approach. Let Υ_i represent the linear approximations of the `warming curve` of $\mathbb{P}_i$. The set of Υ_i for stages in $\mathbf{G}$ is termed by $\mathbf{R}$. Moreover, $\Upsilon_i(t_i^{inv})$ denotes the value of Υ_i at point t_i^{inv}.

Algorithm 9 Assign APTM for **G** in case A

Input: $\mathbf{G}$, $\mathbf{t^{hdc}}$, $\mathbf{c}$, Θ_k, $\mathbf{K^{min}} = \{K_i^{min} | i \in \mathbf{G}\}$, $\mathbf{R} = \{\mathrm{Y}_i | i \in \mathbf{G}\}$
Output: $\{(t_i^{vld}, t_i^{inv}) | i \in \mathbf{G}\}$
1: step size $\tau_i \leftarrow c_i(1/K_i^{min} - 1),\ \forall i \in \mathbf{G}$
2: $g_i \leftarrow 1,\ t_i^{inv} \leftarrow \tau_i,\ \forall i \in \mathbf{G}$
3: **while** true **do**
4: valid set $\mathbf{V} \leftarrow \{i | \sum_{i \in \mathbf{G}} t_i^{inv} + \tau_i \leq \Theta_k\}$.
5: $\mathbf{V} = \emptyset$? **break** : **goto next line**
6: get increasing reward $\omega_i \leftarrow \frac{\mathrm{Y}_i(t_i^{inv}) - \mathrm{Y}_i(t_i^{inv} + \tau_i)}{\tau_i}$, $\forall i \in \mathbf{V}$
7: $(\forall i \in \mathbf{V}, \omega_i \leq 0)$? **break** : **goto next line**
8: find i where $\omega_i = \max_{i \in \mathbf{V}} \omega_i$
9: $t_i^{inv} \leftarrow t_i^{inv} + \tau_i$, $g_i \leftarrow g_i + 1$.
10: **end while**
11: $t_i^{vld} \leftarrow g_i \times c_i,\ \forall i \in \mathbf{G}$

Algorithm 9 presents the pseudo code of assigning t^{inv}s for the stages in **G** in this case. Firstly, we calculate the constant step size for t_i^{inv}, which is the ideal solution when g_i equals 1 (line 1). We also initialize $g_i = 1$ and $t_i^{inv} = \tau_i$ because they are guaranteed to be feasible for constraints (4.30) and (4.31). Then, the values of t_i^{inv}s are increased with corresponding constant step sizes in a while loop until all of them cannot be increased any more (line 5) or further enlarging any of them will cause higher temperature (line 7). In each loop, we first find the valid stages set, i.e., the t_i^{inv} which can be added by one step size τ_i without violating constraint (4.30). Then, the increasing reward ω_i is derived (line 6) according to Y_i. The reward ω_i normalizes the benefit of increasing t_i^{inv} in lowering the temperature of the system. Finally, we add one step size to the t_i^{inv} having the highest increasing reward ω_i (lines 8 and 9), and update the corresponding g_i. After the while loop, we can directly get $t_i^{vld} = g_i \times c_i$ (line 11).

Case B

In this case, it's clear that all t^{inv}s cannot be their ideal values simultaneously even all valid time lengths t^{vld}s equal their corresponding c. Therefore, the t^{vld}s are set as their lower bound c_i. Of course we can set one or some t^{inv}s as their ideal values in some scenarios, but the other t^{inv}s have to be smaller in value, which could cause severely unbalanced thermal hot spots in the system. Then, according to the APTM

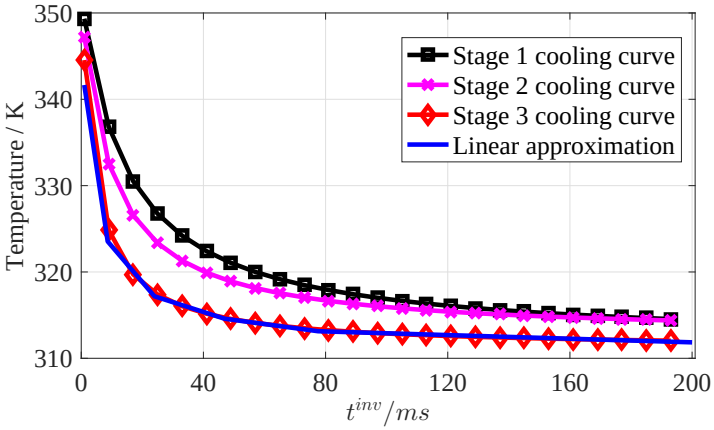


Figure 4.5: The `cooling curves` of the stages in a 3-stage pipelined system. The valid times of the stages are $\{15.2, 10, 4.6\}ms$.

constraint set, we have $t_i^{inv} \leq t_i^{idl} = c_i(1/K_i^{min} - 1)$ for all stages in $\mathbf{G}$.

Finally, the remained problem is how to properly assign t_i^{inv} under (4.30)

Our approach assigns t^{inv}s for `APTM-feasible stages` according to their unique thermal properties. In this case, we adopt the linear approximation of `cooling curve` to model the different abilities of different stages in cooling the system when they are assigned the same t^{inv}.

Definition 4.4 (cooling curve) *The `cooling curve` of stage $\mathbb{P}_i$ with respect to an application* **TASK** *is defined as the relationship between peak temperature $T_i(t_i^{inv}, c_i)$ and t_i^{inv} when, (1) $t_i^{vld} = c_i$, and other stages always stay at 'sleep' state.*

Fig. 4.5 displays the `cooling curves` of the stages in a 3-stage pipelined system. It can be observed that the cooling curves of the three stages are different. Similarly, our approach utilizes the piecewise linear approximations of `cooling curves`. Note that we adopt only the valid part of the linear model, i.e., the part in the domain $t_i^{inv} \leq t_i^{idl}$, since t_i^{inv} must be less than its ideal value.

Now, we discuss the linear model of the `cooling curve` in details. Suppose the valid part of piecewise linear model of the `cooling curve` of stage $\mathbb{P}_i$, denoted as Ω_i, has q_i curve segments, as shown in Fig. 4.6. The slope of the yth segment is termed as $\eta_i(y)$. In addition, one can easily prove that $|\eta_i(y)| > |\eta_i(y+1)|$ as the original cooling curve is a non-increasing function. We denote the set of Ω_i for stages in $\mathbf{G}$ as $\mathbf{O} = \{\Omega_i | i \in \mathbf{G}\}$.

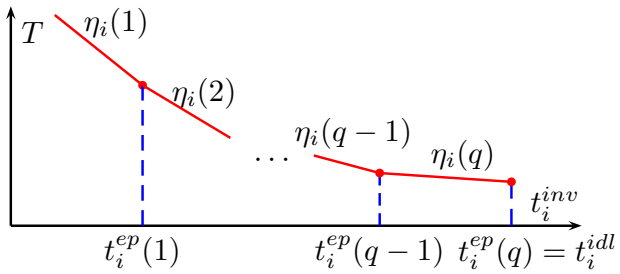


Figure 4.6: The valid part of the linear model of the `cooling curve` of stage $\mathbb{P}_i$. Notation $t_i^{ep}(y)$ denotes the t^{inv} at end point of yth segment, where i stands for i^{th} stage $\mathbb{P}_i$. of the linear model of the `cooling curve` of i^{th} stage $\mathbb{P}_i$. Similarly, $\eta_i(y)$ is the corresponding slope.

The pseudo code of assigning t^{inv}s for the stages in $\mathbf{G}$ in the case $\mathcal{S}_{min} > \Theta_k$ is depicted in algorithm 10. The inputs are the set $\mathbf{G}$, hardware-constraint $\mathbf{t^{hdc}}$ and WCETs $\mathbf{c}$ of the stages, upper bound Θ_k, and Ω, the set of linear model of `cooling curves`. The valid time length t_i^{vld} directly gets its best value, that is, c_i, as aforementioned. We initialize t_i^{inv} as the corresponding hardware constraint t_i^{hdc} (line 1). In the algorithm, we assume the initial value of t_i^{inv} satisfies: $t_i^{hdc} \leq t_i^{ep}(1)$, i.e., the initial t_i^{inv} locates on the first segment in the linear model of cooling curve (line 1). Then, the values of t_i^{inv}s are increased in a while loop until all of them reach their ideal solutions (line 4) or their sum equals Θ_k (line 14). In each loop, we first find the stages that haven't reached their ideal solution (line 3). Only t^{inv}s in this valid set $\mathbf{V}$ will be increased in following steps. After $\mathbf{V}$ is obtained, the calculated step size is assigned to valid t_i^{inv}s by the process of weighting (lines 10 and 14). The weight of each t_i^{inv} is determined by the current cooling effect of stage $\mathbb{P}_i$, which is represented by the slope η_i in the linear cooling curve model Ω_i (lines 5 and 6). The step size z is carefully calculated in each loop such that after assignment, every t_i^{inv} still locates on the segment where it locates before the assignment (lines 7 and 8). Moreover, the step size is also checked whether assigning it t^{inv}s will violate Θ_k (line 9), and is revised in necessary scenarios (line 13).

4.7.3 Summary of the algorithms

Based on the algorithms presented in previous sections, the online algorithm of the Adaptive Periodic Thermal Management (APTM) is summarized in algorithm 11. The algorithm computes thermal-aware APTM

Algorithm 10 Assign APTM for **G** in case A

Input: **G**, $\mathbf{t^{hdc}}$, **c**, Θ_k, **O**
Output: $\{(t_i^{vld}, t_i^{inv})|i \in \mathbf{G}\}$

$t_i^{vld} \leftarrow c_i,\ t_i^{inv} \leftarrow t_i^{hdc},\ y_i \leftarrow 1,\ \forall i \in \mathbf{G}$
while true **do**
  valid set $\mathbf{V} \leftarrow \{i|y_i \leq q_i\}$.
  $\mathbf{V} == \emptyset$? **break** : **goto next line**
  get the segment slope η_i at t_i^{inv} from Ω_i, $\forall i \in \mathbf{V}$
  assigning-weight $\omega_i \leftarrow |\frac{\eta_i}{\sum_{i \in V} \eta_i}|,\ \forall i \in \mathbf{V}$
  next critical point $\tau_i \leftarrow t_i^{ep}(y_i),\ \forall i \in \mathbf{V}$
  step size $z \leftarrow \min_{i \in \mathbf{V}}(\frac{\tau_i - t_i^{inv}}{\omega_i})$
  if $\sum_{i \in \mathbf{G}} t_i^{inv} + z < \Theta_k$ **then**
    $t_i^{inv} \leftarrow t_i^{inv} + z \times \omega_i,\ \forall i \in \mathbf{V}$
    $y_i \leftarrow y_i + 1,\ \forall i \in \mathbf{V} \bigcap \{i|t_i^{inv} == t_i^{ep}(y_i)\}$
  else
    $z \leftarrow \Theta_k - \sum_{i \in \mathbf{G}} t_i^{inv}$
    $t_i^{inv} \leftarrow t_i^{inv} + z \times \omega_i,\ \forall i \in \mathbf{V}$, then **break**.
  end if
end while

Algorithm 11 Adaptive Periodic Thermal Management

Input: Hardware constraint set $\mathbf{t^{hdc}}$, WCET vector **c**, real-time constraint set $\mathbf{K^{min}} = \{K_i^{min}|1 \leq i \leq n\}$ and $\bar{\mathbf{b}} = \{\bar{b}_i|1 \leq i \leq n\}$, offline learning results $\mathrm{Y} = \{\mathrm{Y}_i|1 \leq i \leq n\}$ and $\Omega = \{\Omega_i|1 \leq i \leq n\}$
Output: $\{(t_i^{vld}, t_i^{inv})|i \in \mathbf{G}\}$

1: determine APTM-feasible stages **FS** and always working stages **AWS** with algorithm 8.
2: determine APTM schemes for **FS** with algorithms 9 and 10.
3: apply APTM schemes on stages in **FS** and switch on stages in **AWS**.

schemes for the n-stage pipelined system according to the APTM constraint set and the thermal property knowledge learned in offline experiments. Our approach does not have any requirement on when and how often should the adaption action is performed. The adaption can be implemented periodically or in event-triggered manner, according to the system designer.

4.8 Offline Part Algorithms

So far, the online part of our approach has been elaborated. Our approach is an offline and online combined one for effective thermal optimization and high efficiency. The offline part is discussed in this section.

The offline part includes three parts: (1) acquiring the `warming curves` and (2) `cooling curves` of the system, and (3) finding the best parameter λ used in online adaption for the application. The `warming curves` and `cooling curves` of the system can be easily acquired by implementing different PTM schemes on the system according to their definition and then recording the peak temperature. Due to the simplicity, the detailed algorithms are omitted for brevity.

The pseudo codes of finding λ are listed in algoithm 12. For a task application **TASK**, the parameter λ determining b_i is found by brutally searching its feasible range $(0, 1)$ with a constant step size ε (line 2). The λ is set as the best one which leads to the minimal peak temperature when the system running application **TASK** (line 6). To diminish the influence from the variability in event arrivals and executions, the peak temperature is calculated as the average values of N_s simulations (line 4).

Algorithm 12 find the parameter λ

Input: **TASK**, *cooling curves* $\Upsilon = \{\Upsilon_i | 1 \leq i \leq n\}$, *warming curves* $\Omega = \{\Omega_i | 1 \leq i \leq n\}$, step ε, simulation number N_s
Output: λ

1: $T_{best} \leftarrow \infty$.
2: **for** $\bar{\lambda} = \varepsilon$ to $1 - \varepsilon$ with step ε **do**
3: simulate the application **TASK** N_s times with approach APTM which uses $\bar{\lambda}$ to determine b_i in equation (4.18).
4: get the average peak temperature T_{avg}.
5: **if** $T_{avg} \leq T_{best}$ **then**
6: $T_{best} \leftarrow T_{avg}$, and $\lambda \leftarrow \bar{\lambda}$
7: **end if**
8: **end for**

4.9 Simulation Evaluation

We evaluate the effectiveness and feasibility of our proposed approach APTM in this section. APTM is compared with two existing approaches

in our case studies, i.e., (1) the Balanced Workload Scheme (BWS) proposed in [24]. (2) the Pay-Burst-Only-Once based offline approach (O-PBOO) presented in [2]. BWS is an online energy-optimization approach based on adaptive dynamic power management. O-PBOO is an offline peak-temperature optimization approach which searches the optimal pair of $(\mathbf{t^{off}}, \mathbf{t^{on}})$ in offline manner. The obtained $(\mathbf{t^{off}}, \mathbf{t^{on}})$ is then applied to the system to periodically switch the stages to 'sleep' state. Note that the offline search step of (O-PBOO) is set as the minimal step for a fair comparison. The three approaches are implemented in MATLAB with additional RTC-toolbox and RTS-toolbox. All the simulations are performed on a computer with an Intel i7-4770 processor and 16GB memory.

4.9.1 Setup

In our case studies, the multi-core processor is modeled by the celebrated thermal model HotSpot [52]. The cooling and heating phenomena is modeled by applying the well-known Fourier model. In order to calculate thermal parameters, we take the advantage of the well-known electro-thermal analogy [52, 97, 42, 43, 45, 76], i.e., the RC thermal network. Regarding the power consumption, we consider the total power dissipation contains two parts: dynamic power and leakage power. The dynamic power depends on the supply voltage and running frequency of the core. The leakage power, is mainly influenced by temperature. The dependency relationship between the leakage power and temperature is closely approximated by a linear function of the processor temperature

To investigate the effectiveness and efficiency of our approach on pipelined architectures with different stage numbers, three real life platforms are used in our case studies: homogeneous multi-processor ARM platforms with three and eight cores, and the Single-Chip Cloud Computer (SCC), a processor created by Intel that has 48 distinct physical cores [46]. The power and thermal parameters of the platforms come from [97, 83] and parameter calibration. Moreover, the well-known HotSpot toolbox is adopted to construct the thermal model from the floorplan and thermal parameters of the platform. The thermal parameters of the HotSpot are summarized in Table 4.1. All the mode-switching overheads are set as $\mathbf{t^{swoff}} = \mathbf{t^{swon}} = (1, \cdots, 1)ms$. In most scenarios during case studies, only part of the cores in the platform are activated while other cores are switched off. The layout of the activated n cores is determined by

Table 4.1: Parameter configuration of HotSpot

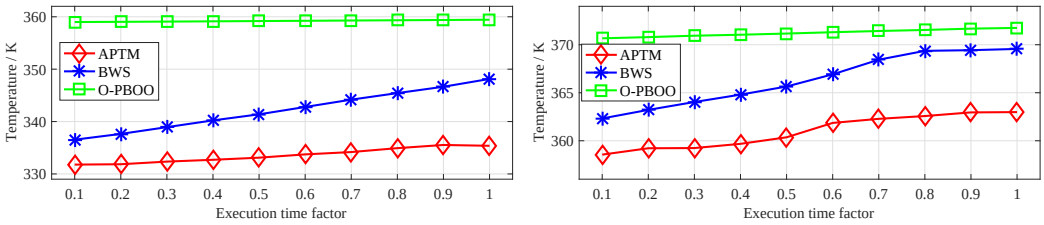
parameter	value
silicon thermal conductivity in $W/(m \cdot K)$	100
silicon specific heat in $J/(m^3 \cdot K)$	$1.75 \cdot 10^6$
chip thickness in m	0.00035
convection resistance in K/W	0.1
heatsink side in m	0.066
heatsink thickness in m	0.001
heatsink thermal conductivity in $W/(m \cdot K)$	400
heatsink specific heat in $J/(m^3 \cdot K)$	$3.55 \cdot 10^6$
interface material thermal conductivity in $W/(m \cdot K)$	4.0
ambient temperature in kelvin	300

selecting the n cores whose locations are the closest to core #1 according to the floorplan.

The adopted arrival curve in this work can model many common timing models of event stream, such as the sporadic event model and the periodic event model. For clarity of demonstration, we adopt the event stream modeled by the PJD timing model in case studies. The PJD timing model specifies an event stream by the period p, jitter j, and minimal inter arrival distance d. The upper arrival curve of a stream in PJD model is $\alpha^u(\Delta) = \min(\lceil \frac{\Delta+j}{p} \rceil, \lceil \frac{\Delta}{d} \rceil)$. The adopted input stream are set as: $p = 100ms$, $j = 150ms$, and $d = 0ms$. The WCETs on the stages are randomly chosen between $3ms$ and $15ms$. The relative end-to-end deadline of the application is set as $120ms$. To sufficiently warm up the platforms, the event trace length of the application is set as 15 seconds. The event trace is generated by the RTS toolbox according to its arrival curve α^u. During simulation, the real execution time of a event is randomly chosen from the range $[e^b, e^w]$, where e^w is the WCET of the event, and e^b is the best-case execution time and determined by the execution-time factor f_e: $e^b = f_e \times e^w$.

4.9.2 Effectiveness at different execution-time factors

We first report the effectiveness of our approach in utilizing the execution slacks to optimize the peak temperature. The simulations are conducted on 3-stage and 8-stage ARM processors with the execution-time factor varying from 0.1 to 1 with step 0.1. The adaption periods of our approach APTM and BWS are set as $40ms$ and $18ms$ in the two



(a) 3-stage (b) 8-stage

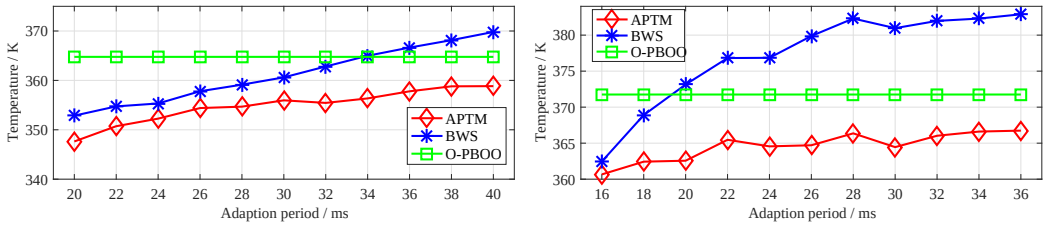
Figure 4.7: The peak temperature from three approaches with different execution-time factors on the ARM processors with (a) three and (b) eight cores.

platforms, respectively. Fig. 4.7a and Fig. 4.7b displays the results of the three approaches obtained from the 3-stage and 8-stage platforms, respectively.

From the figures we can make following observations: (1) In all the cases, our approach APTM outperforms BWS and O-PBOO regarding optimizing the peak temperature, which demonstrates the effectiveness of our approach. (2) Generally, the peak temperatures from three approaches get higher as the execution factor increases. This is expected because (a) the average `execution slack` gets smaller as f_e increases, thus leaving less time slots for the cores to sleep in. (b) for approach O-PBOO, although same $\mathbf{t}^{\mathbf{off}}$ and $\mathbf{t}^{\mathbf{on}}$ are applied in different cases, the time slots when the cores can switch to 'idle' state decreases in average, thus causing higher peak temperature. (3) For the 8-stage platform, the peak temperatures from BWS and APTM grows slower when $f_e \geq 0.8$ compared with the 3-stage platform. The reason is that the variety of execution times of the events has smaller impact on the temperature for 8-stage platform. (4) When there is no execution slack, i.e., $f_e = 1$, approach APTM still provides lower peak temperature than other two approaches. This indicates APTM can also utilize the `dynamic slack` effectively.

4.9.3 Effectiveness at different adaption periods

We also investigate the relationship between peak temperature and the adaption periods. Similarly, the three approaches are implemented on the 3-stage and 8-stage pipeline systems. Moreover, in order to study the effectiveness of our approach in utilizing the workload slacks, we set the execution factor as 1 in the case studies. We vary the adaption period



(a) 3-stage (b) 8-stage

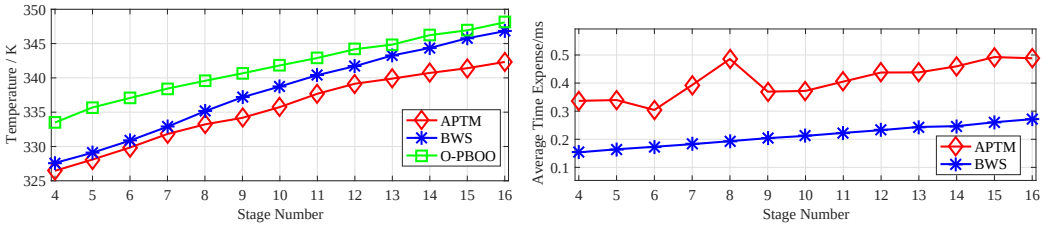
Figure 4.8: The peak temperature from three approaches with different adaption periods on the ARM processor with (a) three and (b) eight cores.

from 20*ms* to 40*ms* on the 3-stage platform, and from 16*ms* to 36*ms* on the 8-stage platform. In addition, the simulation adopts same event trace in different adaption-period scenarios. The results from the 3-stage and 8-stage platforms are plotted in Fig. 4.8a and Fig. 4.8b, respectively.

From the figures, one can observe that: (1) In general, the peak temperatures obtained from APTM and BWS is elevated when the adaption period increases. This result is intuitive as a longer adaption period means less dynamic information is collected and more worst-case parameters are considered. (2) The peak temperature from BWS strongly depends on the adaption period and increases quickly as the period grows. For large adaption periods, BWS even gives worse results than the offline approach O-PBOO. This is because BWS first sleeps the stages for a certain time and then let them be active until next adaption instant. Therefore, the stages will stay at unnecessary 'active' or 'idle' state longer between adaption instants for a larger adaption period. (3) Compared to BWS, the increase speed of peak temperature form APTM is much slower and is still lower than that of O-PBOO with the largest adaption period. The reason is that our approach adopts APTM schemes, which can work well without fast adaption. (4) The peak temperature of O-PBOO is constant regarding the adaption period since it is an offline approach and thus is independent of the adaption period.

4.9.4 Efficiency regarding stage number

We also report the computing times of our approach with respect to the stage number. Since our approach works in online manner, its efficiency is also an important factor that should be considered. The 48-core Intel SCC platform is used in this part. The three approaches are tested in



(a) peak temperature (b) average time expense

Figure 4.9: (a) The peak temperature from three approaches and (b) the average time expense of online adaption of APTM and BWS, on platform IntelSCC with different active stage numbers.

scenarios from 4-stage up to 16-stage with the same stream time settings with above simulations. The WCETs of each sub-tasks are randomly generated in range $[5, 9]ms$. We set the deadline for n-stage scenarios as $(20 + 15n)ms$, and set the adaption period as $5ms$. Fig. 4.9b reports the average computing times of approach APTM and BWS at different stage numbers. Moreover, the peak temperatures in scenarios are also plotted in Fig. 4.9a.

We first discuss the results in Fig. 4.9a. The peak temperature of the three approaches grow as the stage number gets bigger. This is expected because as more cores are activated, more heat is generated. Observe that: (1) our approach APTM outperforms other two approaches in all scenarios, which further strengthens the effectiveness of our approach for pipeline architectures with different stage numbers and varied adaption periods. (2) The temperature from APTM grows slowly while the temperature from BWS increases faster, which demonstrates the scalability of APTM with respect to the stage number. (3) The gaps between temperatures of two approaches on Intel SCC are smaller than those on the 3-stage and 8-stage ARM processors. The reason is platform Intel SCC is not fully activated while other two platform are completely activated. Therefore, the temperatures as well as the gaps are lower on Intel SCC.

Finally, let's discuss the time expense of online adaption. As shown in figure, the average time expenses of APTM are less than $0.5ms$ and grow approximated linearly, which indicates the efficiency of our approach. We can also observe that the average time expenses of APTM are higher than that of BWS, this is expected because APTM considers the unique thermal property of the stages and thus requires more calculations. However, as demonstrated above, our approach works better

than BWS, especially in scenarios having large adaption periods. Therefore, the online time expense of APTM is acceptable considering the achievement in lowering peak temperature.

4.10 Summary

We have presented a new Dynamic Power Management approach named Adaptive Periodic Thermal Management to minimize the peak temperature of pipelined hard real-time systems. Based on the proposed hard real-time constraints, our approach computes adaptive APTM schemes at each adaption instant to control the temperature, during which the unique thermal properties of the stages are considered. Several lightweight algorithms are given for online adaption and one algorithm is also proposed to calculate the parameter used in online adaption. Case studies results demonstrate our approach is scalable w.r.t the number of stages, in terms of both temperature optimization and computation efficiency.

In the next chapter, we present a multi-core thermal framework which can evaluate various thermal management policies on actual hardware platforms in an efficient and reliable manner.

Chapter 5

Multi-core Fast Thermal Prototyping Framework

As technology for microprocessors swifts in the nanometer regime, power density is rapidly increased and has become one of the constraints to higher performance, especially for multi-core processors. Hot temperature, caused by high power density, severely hampers the reliability and performance of microprocessors. The traditional thermal managements which are designed for typical thermal conditions, i.e., physical cooling devices, are challenged by the significant spatial and temporal variation of chip temperature, for the sake of cost-effectiveness [108]. To meet such challenges, Dynamic Thermal Management (DTM) techniques have been proposed to control the temperature actively.

There have been plenty of DTM researches which are based on various temperature control mechanisms such as Dynamic Voltage and Frequency Scaling (DVFS), Dynamic Power Management (DPM), job scheduling and task migration. Designers need to select the proper thermal management policy to manage the temperature on the target platform under various constraints, e.g., peak temperature constraint or real-time constraint. These policies are often evaluated by simulation programs, which simulate the execution, power dissipation and temperature evolution of the processor according to user-defined models. The thermal management results obtained from simulation have little credibility since the adopted processor power and thermal models are usually simplified for efficiency. Moreover, when targeting commodity setups, that is, systems with off-the-shelf hardware and software environments, the timing behaviour of the system is influenced by the op-

erating systems and the computer architecture. These concerns are often ignored in simulation programs. Nowadays, DTM researches show a trend towards multi-core architectures in which multiple cores work concurrently as a set of heat sources. Thermal management policies must properly arrange the execution of different tasks on different cores to optimize the temperature or performance while considering the heat influence between different cores. This makes comparing and selecting thermal management policy more complex.

We argue that validating the effectiveness of all selections in the early design phase on the target architecture is essential to select the right thermal management policy for commodity setups. These validating procedures can be accomplished by prototyping the policies on real hardware platforms with a high-level thermal framework. Such a framework should enable the designers to prototype the policies in a fast and efficient manner. To compare the performance of different policies, it also should offer results that can reflect the real influences of thermal policies to the temperature on the target platform. Specifically, such a framework must

- realize basic thermal-aware controlling mechanisms, i.e., a temperature control mechanisms library,
- allow the implementation of customized thermal management policies with minimal effort,
- evaluate thermal policies according to the temperature of real processors,
- have minimal requirements on the hardware and underlying software for better compatibility.

We study how to develop such a framework in this chapter.

5.1 Overview

The traditional frameworks of evaluating thermal management policies either are based on the power and thermal simulators of a processor [74, 107, 105, 9, 5, 18] or utilize a customized version of one standard operating system kernel to support the new thermal management technique [45, 72]. Therefore, these implementations either have little credibility in validating the effectiveness of the policies on real platforms or are difficult to maintain and place strict requirements on the hardware

and software environment. Moreover, some researchers implement their work in user-space with a standard Linux kernel [39]. However, these implementations are limited to the specified policies and can be hardly re-used for validating other policies.

In this chapter, we propose the Multi-core Fast Thermal Prototyping (McFTP) framework, which is a thermal framework meeting all the aforementioned requirements. First, McFTP utilizes the physical processors to execute real tasks or benchmarks. The temperatures of the cores are obtained by reading hardware thermal sensors built inside the processor instead of using thermal simulators. Second, McFTP implements several basic thermal management mechanisms, including frequency-scaling, sleep state switching, task-migration and job scheduling. With such a thermal library, McFTP enables the comparison and evaluation of a large set of thermal management policies. Third, McFTP defines a Configuration Manipulation Interface (CMI), which separates the policies from the detailed low-level implementations. CMI defines a set of easy-to-use sub-interfaces to control the low-level execution of workload on the physical cores. Thus, customized thermal management policies can be quickly realized as the designer only needs to implement the high-level algorithms of the policies. Finally, McFTP has wide compatibility as it resides in the user-space and has little interaction to the kernel-space. In addition, McFTP has few requirements on the hardware, i.e., only the Advanced Configuration and Power Interface (ACPI) and hardware thermal sensors, which are common features of modern processors. We also implement the proposed framework on the top of POSIX-compliant operating systems targeting a Dell Core-i7 desktop platform and study its performance. The effectiveness of McFTP is demonstrated by two existing thermal and power management policies with 33 benchmarks. The efficiency of McFTP, i.e., the running overheads of proposed framework, is also investigated by experiments on two platforms.

The rest of this chapter is organized as follows: The related work is briefly introduced in section 5.2. Section 5.3 describes our system models and background knowledge. A motivation example is presented in Section 5.4. Section 5.5 presents the configuration manipulation interface. The overall structure of proposed framework is demonstrated in Section 5.6. Section 5.7 presents the implementation of McFTP on POSIX-compliant systems. Experiment results are investigated in Section 5.8. Section 5.9 concludes.

5.2 Related Work

Many researches have been proposed to evaluate multi-core thermal management policies in different levels of accuracy and for different applications. In this section, we briefly discuss the closest thermal evaluation frameworks.

The majority of thermal frameworks are programs that obtain the temperature traces by simulating firstly the power dissipation and then the temperature evolution of the target processor. In general, such frameworks have three major components, namely the processor simulator, the power simulator and the temperature simulator. The processor simulator does the logical simulation of the processor and provides access and usage statistics to relevant architecture and microarchitecture blocks. A famous one is the Gem5 [12], which encompasses system-level architectures as well as processor microarchitectures. It supports various commercial ISAs (Instruction Set Architecture), including Alpha, ARM, SPARC, MIPS, POWER, RISC-V and x86 ISAs. It also supports processors of homogeneous and heterogeneous multi-core architectures. It performs cycle-accurate simulation and computes the number of accesses to all units during the execution of a benchmark. The second component, i.e., the power simulator, computes the power dissipation estimates of the processors and interconnect primitives. Wattch [16], a framework for analyzing and optimizing microprocessor power dissipation, enables architecture-level power dissipation exploration through a cycle-accurate model of a single-core processor. To accurately model the power of multi-core architectures, a novel power, area and timing modeling framework called McPAT [65] is proposed. Finally, the power estimation of the processor is fed to the temperature simulator to compute the temperature trace. A well-known thermal simulator is the HotSpot [52]. It calculates temperature evolution based on an equivalent circuit of thermal resistance and capacitance that correspond to microarchitecture floorplan blocks and essential aspects of the thermal package. Combining the aforementioned or other similar tools, many simulators and frameworks have been presented in literature. Typical examples are [82, 47, 107, 29]. Although the above frameworks can accurately simulate the logical behaviour w.r.t. thermal management policies, the correctness of the temperature evolution strongly depends on and could be limited by the power and thermal parameters, thermal model and floorplan description. Thus, evaluating thermal management policies in such methodology lacks evidence of the effectiveness of the policies.

Instead of adopting software simulators to get the temperature, some researchers validate their policies by implementing them on real platforms based on a customized version of standard operation system kernels. Zhu Changyun et al. implement the proposed ThermOS run-time thermal management algorithms within the Linux 2.6.8 kernel in [112]. Several parts of the kernel, including performance-counter based power modeling and power-thermal budgeting, have been modified in the implementation. Similarly, Hettiarachchi et al. in [45] test their theoretical results on an Intel i7-950 multi-core processor with modified Linux kernel (2.6.33.7.2-rt30 PREEMPT RT). Compared to the thermal-simulator-based methods, such implementations offer more evidence of the results. Since these policies are integrated within the modified kernel, high timing accuracy is also provided. The downside of such implementations is that it could be costly to extend them to new software platforms as they have specified requirements to the operating system kernel. Moreover, some implementations run, at least partly, in the kernel-space and could affect other functionalities of the system. There are also some thermal-aware policies that have been tested in the user-space of a standard operating system. The examples could be the feedback thermal controlling approach in [39] and the hierarchical power management in [73]. The main drawback of these test beds as well as the aforementioned kernel-customizing implementations is that they are merely designed for the proposed policies in their work. Thus, extending them to new thermal management policies could be costly or even impossible since it requires re-modification, re-verification and re-testing of the implementations. The framework proposed in this chapter is designed to be a general platform and can implement a large set of thermal policies with little effort. To the best of our knowledge, this is the first user-space thermal framework that evaluates different thermal management policies by the temperature of processors on real hardware platforms.

5.3 Background

In general, thermal management policies regulate the temperature of microprocessors via controlling the execution of the workloads. In this section, we first discuss the workload model. Afterwards, we briefly discuss thermal management policies and several common temperature control techniques. Finally, the Advanced Configuration and Power Interface (ACPI) as well as power dissipation model is introduced.

5.3.1 Workload Model

The basic unit of the workload model is a *task* τ. An instantiation of a *task* is termed as a *job*. The jobs of a task can arrive with a period p and a jitter j. Moreover, the execution times of the jobs are bounded by the worst-case execution time C_{wc} and best-case execution time C_{bc}. To cope with the definition of real-time systems, a job might have a relative deadline D, which specifies the maximal allowed time between its release and complete instants.

5.3.2 Review of Thermal Management Policies

Thermal management policies aim to find the optimal resource management scheme which can effectively control the peak temperature, thermal gradient and occurrence of hot spots on the chip. Based on when such optimization procedure is performed, thermal management policies can be divided into two groups.

- Offline policy. Offline policies usually solve the resource management problem in design time or compile time according to the information of workloads and hardware platforms.
- Online policy. Online policies work online and adaptively manage the hardware and software resources according to the current state or the history of the system.

There have been plenty of temperature control techniques or mechanisms. Examples could be clock gating, power gating, dynamic voltage and frequency scaling, stop-go, job scheduling and task migration. Although implemented in different hierarchical levels of the system, such mechanisms share the same idea, i.e., controlling the power dissipation characteristics of a microprocessor for lower temperature or smoother heat distribution. Four temperature control mechanisms that have been widely adopted in various thermal management policies can be listed below.

- Dynamic voltage and frequency scaling (DVFS). This mechanism dynamically scales the supple voltage or clock frequency of a microprocessor to reduce the dynamic power dissipation.
- Dynamic power management (DPM). This mechanism dynamically switches a microprocessor to low power dissipation states in which both dynamic and leakage power can be decreased. Note that no workload can be handled in these states.

- Thermal-aware job scheduling. The execution of the jobs can be reordered via this mechanism to optimize the temporal variation of the temperature.
- Thermal-aware task migration. This mechanism dynamically adjusts the task mapping on the microprocessor to balance the temperature on different cores and thus reduces thermal gradient.

A thermal management policy is usually based on one or more of aforemention mechanisms. The proposed framework in this chapter implements all the above mechanisms and supports offline and online thermal management policies that are based on any combination of these common mechanisms.

5.3.3 Advanced Configuration and Power Interface

Review of Power Dissipation

Temperature strongly depends on the power dissipation of microprocessors. Many existing thermal management policies control the temperature by lowering the total power dissipation. The power consumption of a microprocessor consists of the dynamic switching power and the leakage power. The dynamic power can be calculated by below equation.

$$P^d = \alpha C {V_{dd}}^2 f \tag{5.1}$$

where C is the load capacitance, V_{dd} is the supply voltage, f is the clock frequency and α is the activity factor, i.e., the fraction of transistors that switch each cycle on average [17]. The leakage power is caused by leakage current and can be given as:

$$P^l = I_{leakage} V_{dd} \tag{5.2}$$

where $I_{leakage}$ is the leakage current and is influenced by the temperature. There exist various technologies to reduce the dynamic and leakage power consumption. The typical one for reducing dynamic power can be the *Clock Gating*, which removes the clock signal from a circuit and thus cuts off the dynamic power of the gated section. The supply voltage can be lowered or removed to decrease the leakage power consumption. Such technology is termed as *Power Gating*, which can reduce the temperature more effectively since the leakage as well as the dynamic power is lowered.

Advanced Configuration and Power Interface

To enable robust operating system-directed motherboard device configuration and power management of both devices and entire system, the Advanced Configuration and Power Interface (ACPI) [30] is developed as the common industry interface. In ACPI, several power states are defined for processors. These power states can be divided into two classes.

- Processor Performance States (P-states), which specify different levels of performance of operating processors.
- Processor Power States (C-states), which define different power saving levels of idle processors.

A graphical representation of the P-states and C-states are plotted in Fig. 5.1.

P-states are typically implemented with the Dynamic Voltage and Frequency Scaling technologies on microprocessors. When a microprocessor is in P0 state, it provides the maximal performance and may consume the maximal power. A performance state Pj is termed as a higher state than Pi if $i < j$. The microprocessor offers lower performance when it is in a higher performance state. Consequently, the power consumption is reduced. In Linux operating system, the P-states can be controlled manually via the interface provided by ACPI.

Processor power states are designed at C0, C1, C2, C3, ..., Cn. In ACPI, four standard C-states are defined, i.e., C0, C1, C2 and C3; The C0 power state is an active power state where processor can executes instructions. The performance level and power consumption at C0 are defined by the current P-state. The C1 through Cn power states are the processor sleeping states where the processor consumes less power and dissipates less heat. Since the processor does not handle any workload when it's in a sleeping state, more aggressive power saving technologies such as power gating of whole cores can be applied. Temperature can be significantly lowered when a sleeping state is entered. However, exiting a C-state to normal working state introduces a certain latency which depends on the level of the C-state. Generally, the greater power saving when in the C-state , the longer the latency [30]. In [85], the actual wake-up latencies of C-states of several x86 processors are measured for various recover frequencies. When the operating system expects a certain time span before the next task, C-states will be used to save power. The specific C-state is determined based on the trade-off between power saving effect and the restore latency. Unlike P-states, C-states cannot be controlled

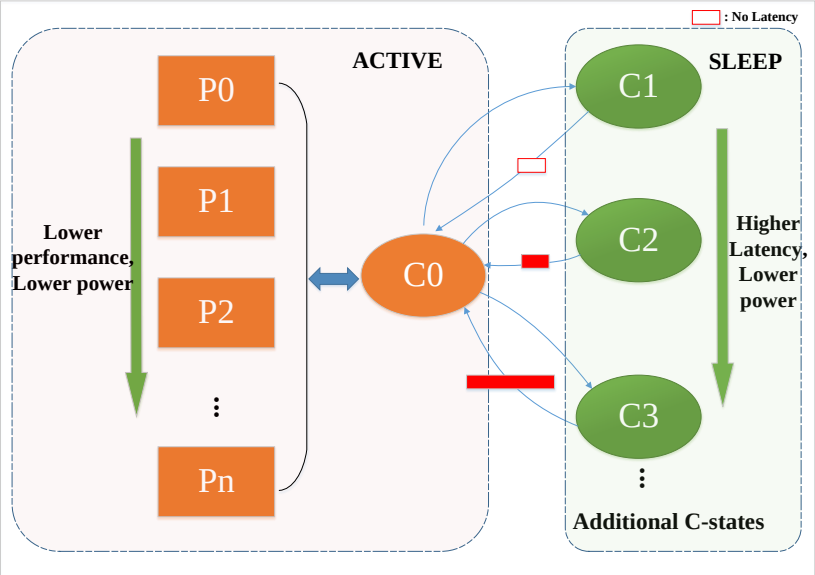


Figure 5.1: Processor P-states and C-states defined in ACPI. The power of C0 state depends on the currently used P-state. The red blocks at the curves connecting C0 and other C-states indicate the latency when the processor returns back to C0. Note the empty red frame indicates no latency.

directly in application level. However, they can be reached indirectly by eliminating workloads on the core, e.g., the Dynamic Concurrency Throttling [33] and the idle waiting policies.

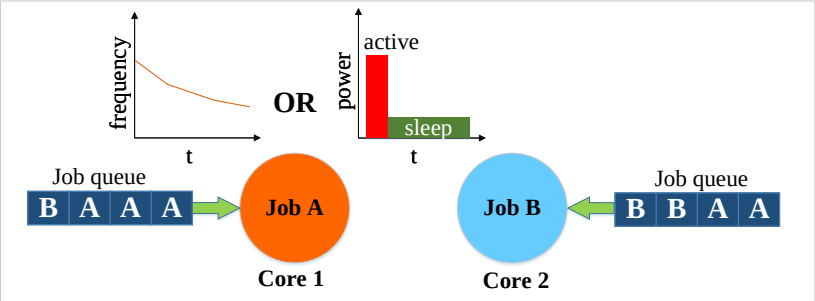
5.4 Challenges and Design Approach

In this section, we discuss the challenges and the design approach of our McFTP framework through a concrete example by adopting different multi-core thermal management policies. Suppose we have a dual-core processor executing two tasks, a hot task *A* and a cool task *B*. A task is termed as hot when executing it leads to a higher temperature on the core. Each core is associated with a buffer storing the waiting jobs. Now, consider the case that the temperature of core 1 is significantly raised by continuously executing jobs of task A while the temperature of core 2 is still in normal range, as shown in Fig. 5.2a. The large thermal gradient between two cores at this scenario hampers the stability and reliability of the processor. Moreover, modern processors usually require the temperature to be lower than certain threshold. Therefore, the temperature

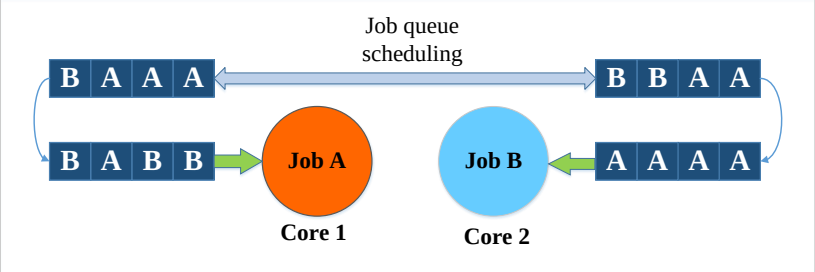
of core 1 should be decreased in this example.

To lower the temperature of a core in a multi-core processor, one may propose to use power management techniques, such as DVFS and DPM, as shown in Fig. 5.2a. The DVFS techniques could be utilized to lower the frequency on the core such that the dynamic power is reduced. In addition, the core can also be switched to C-states by using DPM techniques such as power gating. Power management techniques reduce the temperature at the expense of lowered performance. To maintain the same level of performance, the job queue scheduling technique serves as an alternative method to reduce the thermal gradient, as shown in Fig. 5.2b. One may switch the positions of the two task A jobs at core 1 and the two task B jobs at core 2. In this way, core 1 will execute two cool task jobs so that its temperature can be lowered. Moreover, one can further preempt the current running job on core 1 with the cool task B job in the waiting queue to reduce the temperature. It turns out that temperature can be controlled by diverse types of mechanisms. Note that in the example, we just consider online thermal policies that are based on only one of such mechanisms, not to mention offline policies and hyper policies combining two or more of these mechanisms. It's not clear how to implement these various thermal policies on top of a standard operating system nor how to abstract and extract their common characteristics such that we can reuse one in another. We aim to solve these problems with McFTP.

The objective of McFTP is to provide multi-core system designers a tool which enables the fast evaluation of various type of thermal management policies, e.g., offline or online, DVFS-based or task migration-based, or hyper ones combining two or more temperature control mechanisms, etc. The challenges in the design of McFTP, i.e., the implementation of various thermal management policies and the reuse of their common characteristics, are met by introducing an intermediate interface named configuration manipulation interface (CMI). Four basic thermal controlling mechanisms mentioned in Section 5.3.2, DVFS, DPM, job queue scheduling and task migration, are defined in CMI. Thermal management policies can access these basic mechanisms via a set of unified, pre-defined interfaces and do not need to handle the detailed implementation of the mechanisms on physical cores and the potential correlation between them. In this way, thermal management policies are isolated from the implementations of low-level mechanisms, thus can be evaluated in a fast and reliable manner.



(a) Lowering performance to reduce temperature



(b) Job scheduling to balance temperature

Figure 5.2: Examples of mechanisms to manage the temperature of multi-core processors.

This work does not concentrate on optimizing the temperature for different thermal management policies. Neither does this work claim that the temperature results of this proof-of-concept implementation on any (general purpose) operating systems are identical to those obtained from lower-level implementations. The primary goal of this work is to decouple the high-level description or principles of thermal management policies from the low-level implementations which depend on the system specification. This framework enables system designers to gain deep understanding of the thermal management policies with temperature results from real hardware platforms instead of software simulation.

5.5 Configuration Manipulation Interface

In this section, the Configuration Manipulation Interface is introduced in detail by discussing the sub-interfaces defined in it.

As shown in Section 5.4, many temperature control mechanisms are available on multi-core processors to manage the temperature or heat distribution of the cores. Based on these mechanisms, various thermal

Table 5.1: The state table in CMI. Note that S_i could be an arbitrary state among the sleep state (0) and available frequencies. L_i should be a positive real number denoting the length of the state in pre-defined time unit.

state	length (microsecond)
S_1	L_1
S_2	L_2
...	...
S_i	L_i
...	...

management policies can be proposed to optimize the temperature or performance of multi-core processors. To implement a customized thermal management policy in a fast manner with minimal effort, Configuration Manipulation Interface (CMI) is proposed as the intermediate interface between high-level algorithms of the policies and the low-level implementations of the basic temperature control mechanisms. CMI enables easy and robust control or combination of these basic mechanisms to manage temperature, hot spots and thermal gradient of multi-core processors. Next, we introduce the sub-interfaces defined in CMI for designers to access DVFS, DPM, job scheduling and task migration mechanisms.

5.5.1 Power Management

The evaluated thermal policy can control the power dissipation characteristics of a core in the processor via this interface. The dynamic and leakage power are managed through the DVFS and DPM mechanisms defined in the ACPI of the processor, respectively. The policy needs to provide a *state table* to specify how to control the power dissipation state of a core.

A state table has two columns, as shown in Tab. 5.1. The system designer can assign an individual state table to each core. The first column lists the order of demanded states of the core. A zero means to pause the execution of a job so that the core can switch to the sleep state. A positive number specifies the running frequency of that core. Since the available frequencies of a core in ACPI are defined by a set of discrete points, the given frequency will be rounded to the nearest available one if it does not equal any frequency in the set. The second column depicts

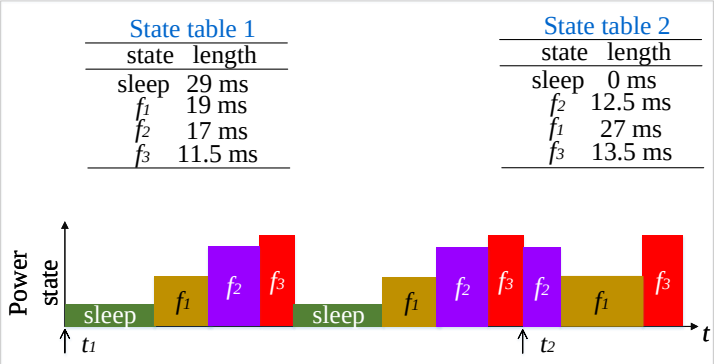


Figure 5.3: An example of McFTP controlling the power states of a core according to two state tables. We consider the core has three available frequencies, which are f_1, f_2 and f_3. State table 1 is applied at t_1 and repeated twice. Then, state table 2 is applied at t_2.

the time length of the corresponding state. The start time of each state is the end time of the previous state. Specifically, the first state is adopted immediately once the state table is given. An example of the *state table* is demonstrated in Fig. 5.3. It is worth noting that the state table will be repeated continuously to control the power dissipation of the core until a new state table is provide to replace the old one. With the *state table*, the evaluated thermal policy can control not only the length each core stays in each power dissipation state but also the order of the states.

5.5.2 Job Scheduling and Task Migration

We consider that upon arrival, the jobs of all tasks are inserted into a set of queues associated with the cores and wait to be executed on the corresponding core. In default, the queue behaves as a First-In-First-Out (FIFO) buffer. New jobs are inserted at the back of the queue and the job at the front of the queue will be executed firstly. Depending on the temperature of the cores, a thermal management policy may need to change the order of the job queue or move one job to another queue. In CMI, the following actions are defined for thermal policies to perform job queue scheduling.

- Advance. This action advances a job by a given number of job positions in the same queue.
- Recede. Similarly, this action recedes a job by a given number of job positions in the same queue.

- Move. This action moves a job in one queue to the specified position in another queue.
- Preempt. When the policy performs this action, the current running job, if exists, on the core connecting this queue will be preempted by the front job in the queue. Then, the preempted job is placed at the front of the job queue.

The above four actions can be accomplished by calling functions `advanceJobInQueue`, `recedeJobInQueue`, `moveJobToAnotherQueue` and `preemptCurrentJobOnCore`, respectively.

In addition to job queue scheduling, CMI also provides the interface for task migration. Thermal management policies can migrate the current running hot job from an overheated core to a cooler core to balance the temperature with such interface. Simply invoking the function `taskMigrate` with the source and target core indexes will make the framework perform the task migration.

5.5.3 Dynamic Information and Task Allocation

In addition to thermal-aware interfaces, CMI also provides the interface to collect dynamic information about the state of the cores as well as the job queues. For each core, such information structure contains the temperature, current power state, the on-going job, the length for which the on-going job has been executed, etc. For each job queue, its dynamic state is abstracted by a vector containing the pointers to the waiting jobs. Thermal management policies can use the dynamic information to make decisions during run-time.

A task allocation interface is also defined in CMI. When a new job arrives, this interface is called to decide where the job should be instantiated. This interface can be static, that is, defined by the designer in design phase, or dynamic, i.e., determined by the evaluated policy online according to the dynamic information of the cores. In default, CMI creates a static allocator which allocates all the jobs evenly on the cores. This default task allocator can be substituted by a customized one via the registration interface discussed in the next section.

5.5.4 Registration Interface

As discussed in Section 5.3.2, thermal management policies can be classified into two categories, namely offline and online policies. These two

types of policies work in different manners and phases. An offline policy finds the optimal resource management scheme, e.g., the state table for controlling power and/or the task mapping on the processor, in design phase and applies the scheme at the beginning of the experiment. An online policy may dynamically change the power state of the cores or schedule the jobs according to the current state of the processor. To support the evaluation of both types of policies, CMI defines a registration interface to register an offline (function `setOfflineThermalPolicy`) or online policy (function `setOnlineThermalPolicy`) in design phase. The registered policy will be invoked automatically based on its type. As aforementioned in the previous section, CMI defines a task allocator to determine on which core a new job should be executed. Designers can also set the task allocator via this interface. To define a static allocator, designers can call `setTaskRunningCore` to statically link the jobs of a task to one specific core. In the same way, a dynamic task allocator can be explicitly set by calling `setTaskAllocator`.

In this section, we have introduced five major sub-interfaces defined in CMI. With them, our framework gains high flexibility in evaluating various thermal management policies working in different manners and replying on different temperature control mechanisms. Note that it is not necessary to cover all the details of CMI in this chapter. CMI also defines a set of interfaces and functions for the convenience of implementation of the policies. They are omitted due to their simplicity.

5.6 Multi-core Fast Thermal Prototyping Framework

After introducing the Configuration Manipulation Interface, we discuss the overall structure of Multi-core Fast Thermal Prototyping Framework. Fig. 5.4 graphically demonstrates the overall structure of McFTP. As shown in the figure, McFTP can be divided into two parts by CMI, one is composed of the functional components of the thermal management policy and the other part consists of the low-level implementations for executing the decisions of the policy on the actual processor. CMI isolates two parts and thus enables a predictive behavior of thermal management policies. McFTP is composed of below components.

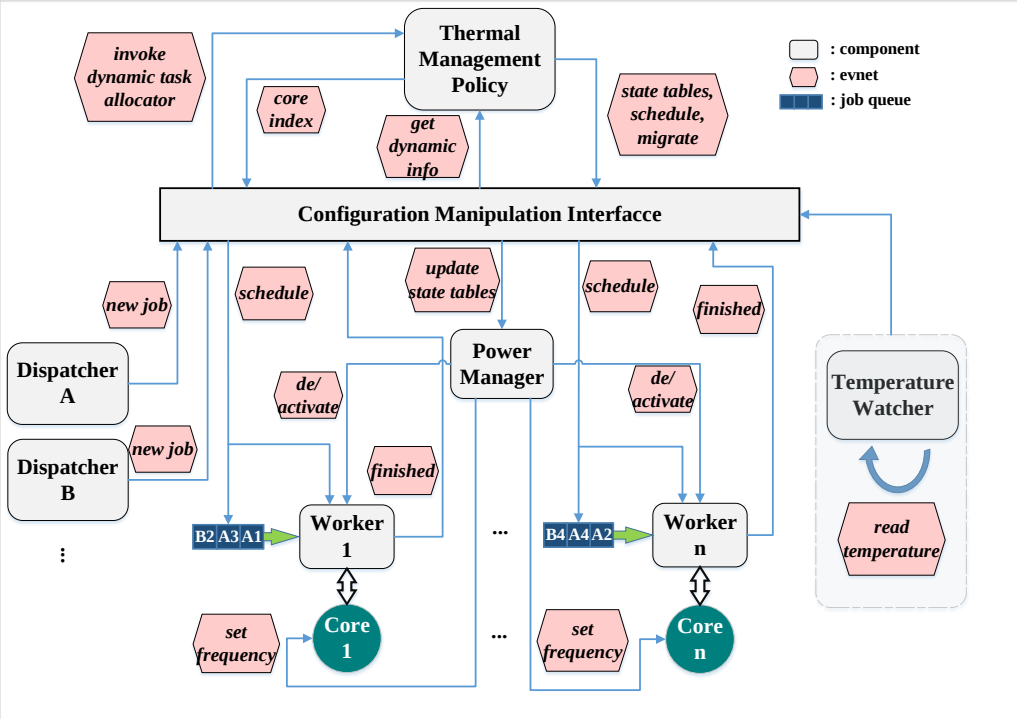


Figure 5.4: The proposed Multi-core Fast Thermal Prototyping Framework.

5.6.1 Dispatcher

A dispatcher is defined for each task. The dispatcher supports periodic, periodic with jitter and sporadic task timing models. A dispatcher creates jobs of the task based on the task timing model. When a new job is created, the dispatcher sends the job instance to CMI by calling `addJob` instead of directly appending the job to one of the job queues. Then, if the pre-defined task allocator is a static one, CMI directly gets the index of the core on which the job should be executed from the allocator. Otherwise, CMI invokes the dynamic task allocator defined in the thermal management policy to determine the core index. Finally, the new job is appended to the corresponding job queue.

5.6.2 Thermal Management Policy

This component is defined by the designer. It should contain full functional descriptions of the policy such as calculating frequencies, determining the state tables for the cores, job scheduling policies, etc. For

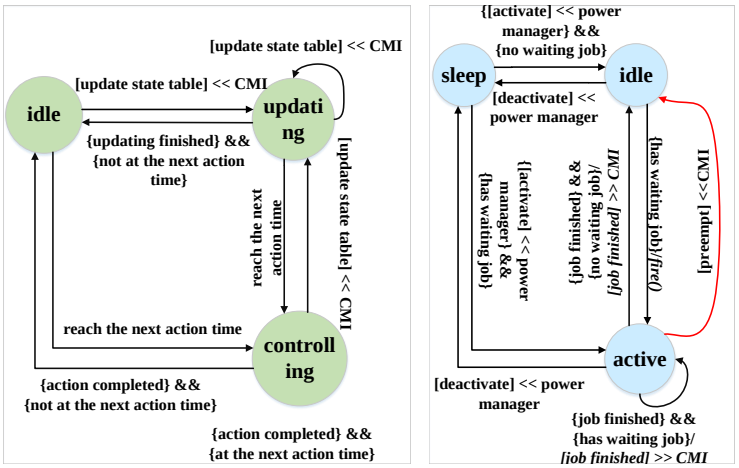
example, if the policy is online and based on DVFS, then the designer should define the online adaption period of the policy and present the routine to calculate the state table at each adaption instant. The comprehensive information about the processor can be obtained via the dynamic information interface in CMI. Then, the thermal policy can manage the resources on the processor via the power management, job scheduling and task migration sub-interfaces. Moreover, the designer may also define a dynamic task allocator in the policy to dynamically assign new jobs to proper cores, as aforementioned in Section 5.5.4.

5.6.3 Temperature Watcher

This component periodically reads and saves the temperatures of all the cores of a real processor. When the dynamic information interface in CMI is invoked, the temperature watcher provides the latest temperatures of the cores.

5.6.4 Power Manager

The power manager controls the frequencies and power states of all cores according to the *state tables* obtained from CMI. It controls the power states of the cores based on the clock frequency and C-states switching mechanisms defined in ACPI. The operation semantics of a power manager is outlined in Fig. 5.5a. After receiving one or more new stable table from CMI, the power manager is in *updating* state. It saves the stable table and then calculates the time instance when the next frequency or power state transition happens. If the next action time instant has not been reached, the power manager stays at *idle* state until the next action time. Then, the power manager is in *controlling* state and takes the corresponding action. After the action, the next action time instant is also updated. For a frequency transition, a power manager simply changes the core frequency via the interface provided by ACPI. In the case of switching one core to sleep, it sends a *deactivate* event to suspend the worker associated with that core. Then, the idle-waiting policy is adopted to stop the worker occupying CPU times so that the core can switch to sleep state, i.e., C-states. At the next action time, the power manager first sends an *activate* event to wake up the worker from sleep state and then switch the core to target state.



(a) Power Manager (b) Worker

Figure 5.5: The operation semantics for Power Manager and Worker entities. $[e]<<s$ indicates receiving an event e from sender s. $[e]>>r$ refers to sending an event e to recipient r.

5.6.5 Worker

The operation semantics of a worker is depicted in Fig. 5.5b. For a n-core processor, n workers are created to virtually represent the cores. Each worker is associated with a job queue storing the waiting jobs. After receiving a *deactivate* event from the power manager, a worker switches to *sleep*, whichever state it is current in. When the power manager sends an *activate* event to it, a worker switches to *idle* state if there is no on-going job, otherwise it switches to *active* state to execute that job. If current job is finished and the job queue is empty, a worker sends an event to CMI to inform the job completion and then goes to the *idle* state, waiting for new jobs. Moreover, to perform job preemption or task migration, a worker can also be interrupted from *active* state by CMI with a *preempt* event. After being preempted, a worker is in *idle* state to load new jobs from the queue, if they exist. If the job queue is not empty, a worker switches back from *idle* state to *active* state and executes the first job in the queue by calling function `fire`.

5.7 Portable Implementation with POSIX

In previous sections, we defined Configuration Manipulation Interface and presented the overall structure of McFTP framework. The abstract

operation semantics of the power manager and worker was also described. In this section, we discuss a specific implementation of McFTP which utilizes the API provided by POSIX standard. The POSIX standard has been widely supported by operating system including many variants of UNIX and Real-Time Operating System (RTOS).

The main goal of McFTP is to evaluate and compare the performance of different thermal management policies on actual hardware platforms in an efficient manner. The evaluation process should be fast, safe and reasonable accurate. We implement McFTP in user-space level as we argue that a user-space tool that can be accessed easily is the first choice if the designer wants to compare different thermal management policies in early design phase. The main concern is the relative thermal optimization performance of the policies among each other, not the absolute performance. Although implementations in kernel-space are more accurate in timing and power consumption controlling, it may be infeasible to prototype thermal management policies in early design phase when the actual hardware and software environment has not been specified. Moreover, the flexibility of McFTP for different operating systems is also significantly limited if it modifies the kernel. Prototyping thermal management policies in user-space is more efficient and provides greater interoperability.

5.7.1 Implementation Requirements

To implement McFTP, two basic features should be supported by the hardware environment. First, the processor must support the Advanced Configuration and Power Interface (ACPI) such that McFTP can control the power dissipation by putting the processor in different P-states and C-states. Second, the temperatures of different cores can be sampled by sensors as the comparison criteria among thermal management policies. In Linux environment, the sensors built inside processors can be read via the tool *lmsensors*.

Besides the hardware environment, the operating system must support several functionalities to realize the framework in user-space. First, McFTP should have the access to the aforementioned ACPI and thermal sensors. Second, the concurrent execution of multiple entities must be provided for running the components in McFTP. Third, preemptive priority scheduling of the concurrent execution should be supported. Finally, timers are necessary to support the time-triggered power manager.

5.7.2 Multi-thread Implementation

Given a POSIX-compliant operating system, we implement McFTP by a set of interacting threads that are assigned different priorities. The priority-based scheduler in the kernel selects the thread with higher priority to execute on the cores. The thermal policy thread has the highest priority p_0. In this thesis, p_i refers to a higher priority than p_j if $i < j$. The dispatcher is assigned priority p_1 and the power manager has priority p_2. Then, the temperature watcher gets the priority p_3. For a n-core processor, at most n worker threads can be created and each worker is assigned to one core. We utilize the `pthread_setaffinity_np` function in POSIX to assign the worker threads on specified cores. All the worker threads have the same priority p_4. Note that apart from worker threads, the aforementioned threads can be attached to arbitrary cores in the processor, which can be customized by designers. Moreover, these threads work in a time-triggered manner. They execute their tasks at the predetermined time instant. When finishing their tasks, they update the next time instant and then block themselves such that lower-priority threads on the same core can get chance to execute.

5.7.3 Power Management Implementation

McFTP controls the frequency of each core via the ACPI interface provided in the operating system. Firstly, our framework tries to set *userspace* as the Linux CPUFreq governor with the module modified from the tool *cpupower*. Then, the `scaling_governor` files in the kernel are dynamically modified by the power manager thread according to the *state tables* given by the thermal policy.

Switching a core to sleep state is accomplished by the idle-waiting policy with the POSIX semaphore library. A worker thread pauses its execution and blocks itself only corresponding to user-defined suspend checkpoints when it receives a *deactivate* event from the power manager. When reaching a suspend checkpoint, the worker thread first calls the function `sem_trywait(&suspend)`. The value of semaphore `suspend` is initialized as zero and can only be incremented by the power manager via sending a *deactivate* event, i.e., calling `sem_post(&suspend)`. The worker thread exits from the suspend checkpoint and performs normal functionalities if the return value of function `sem_trywait(&suspend)` indicates no *deactivate* event has been detected. Otherwise, the worker thread blocks itself by calling function `sem_wait(&resume)`. Similarly, the value of `resume` is also initialized as zero and can only be incremented by the power man-

ager via invoking `sem_post(&resume)`, that is, sending an *activate* event. The code of the suspend checkpoints is provided as a library to the designer.

5.7.4 Task Preemption Implementation

The job scheduling and task migration interfaces defined in CMI both require that our framework can preempt the task currently executed by the worker thread. To enable task preemption, we implement a specific class named *TaskLoad* holding all the information related to task preemption. Similar to the sleep mechanism mentioned in previous section, *TaskLoad* defines the preempt checkpoints to stop the execution when CMI wants to preempt the job. Another semaphore named `stop` is defined to check whether CMI has notified a job preempt request. If so, the thread firstly saves all the data related to the job execution and records the unique identity of the checkpoint. Then, it stops the execution of the job. The object of *TaskLoad* holding all the dynamic information is returned to CMI for job scheduling. When this job is executed by a worker again, the thread first jumps to the preempt checkpoint where the preemption happens by checking the identities of the checkpoints. Then, it continues the execution of the job with the saved data in the object.

5.8 Experimental Evaluation

In this section, the performance of proposed McFTP framework is evaluated. Firstly, we investigate the effectiveness of McFTP in managing the temperature via the configuration manipulation interface. Then, we implement three thermal managements in McFTP and compare their results. Finally, we report the running overhead of McFTP on two platforms that have different computing power abilities.

5.8.1 Temperature Experiments

In this section, we investigate the effectiveness of our framework by reporting the temperature evolution of the cores when the frequency scaling, sleep switching and task scheduling are utilized to manage the temperature. We used a Dell Optiplex 9020 desktop personal computer as the experiment hardware platform. It contains an Intel i7-4770k processor with four physical cores. Each core has 15 available running frequencies between 800MHz and 3.4GHz, when the 'acpi-cpufreq' driver

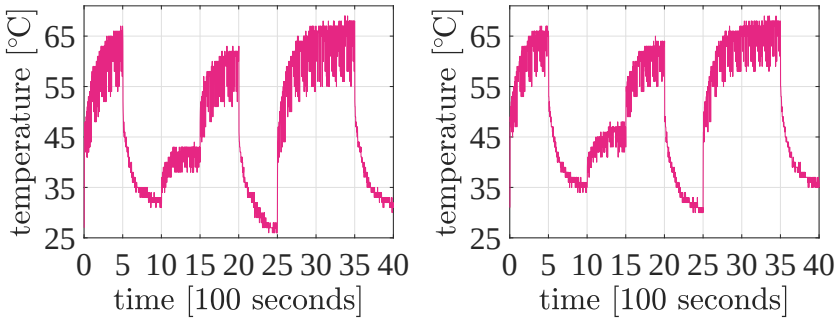
is adopted. The C-states defined for every core are C0, POLL, C1, C3, C6 and C7 if the CPUidle drive is 'intel_idle'. The cores can switch to C-state C7 when they have no workload to handle. The Hyper Thread feature of the processor is disabled in the BIOS (Basic Input/Output System) of the computer. Three air cooling fans are built inside the computer as the cooling system. The experiment ambient temperature is 20 °C . All experiments are done in the 3.16.0-53-generic Linux kernel environment. During the evaluations, the system runlevel of the operating system is set to the lowest level 1 such that only essential system services are running. Four worker threads are created and each one is attached to one core. The other threads such as dispatcher, power manager are evenly attached to all the cores. McFTP framework is implemented in C++ language and compiled by G++ 4.8.4 with optimization level O3 turned on.

In the first experiment, we test the temperature evolution of the cores when they execute jobs in different frequencies. The benchmark `SQRT-RAND` are executed on all the four cores. For clear demonstration, the four cores apply the same state-table, which is shown in Tab. 5.2. The temperature evolutions of four cores are depicted in Fig. 5.6. From the figure, we observe that (1) the temperature of the cores increases quickly from the initial idle state-steady temperature (around 25 °C) to a temperature about 65 °C . Then the temperature changes depending on the running frequency. Note that when the cores switch to sleep state (2000 to 2500 second), the temperature deceases to the initial idle state-steady value and is lower than that of 800MHz frequency state. (2) The increase in temperature is not linear to the increase in frequency. For example, the temperature is raised about 10 °C from 800 MHz to 2100 MHz while a nearly 20 °C temperature increase is resulted by a frequency increase from 2100 MHz to 3200 MHz. The reason is that the dynamic power dissipation is linear to the square of the frequency, as we discussed in Section 5.3. (3) We can also observe that although all the cores adopt the same state table, the temperature of them are not identical. For instance, when running in 800MHz and 2100MHz (500 to 1500 second), the temperatures of core 1 and core 3 are higher than that of core 0 and core 2. This is expected because the cores are on different locations in the processor floorplan. Thus, the heat removing capacity may be different for different cores.

In the second experiment, we show the effect of task scheduling policy. Two tasks, one is a hot task τ^A and the other is a cool task τ^B, are adopted. We implement a simple task scheduling policy, which assigns

Table 5.2: The state table applied in the experiment. Note the zero frequency means sleep state.

state (MHz)	length (second)
3400	500
800	500
2100	500
3200	500
0	500
3400	500



(a) core 0 (b) core 1

(c) core 2 (d) core 3

Figure 5.6: The temperature evolutions of the processor cores when state table Tab. 5.2 is applied to them.

the hot task τ^A to the cores dynamically according to the experiment time. Similarly, the cool task is assigned to the next cores in a circular manner. The temperature results are plotted in Fig. 5.7.

From this experiment, we can make some interesting observations. First, the temperature of a core strongly depends on the workload it executes. This validates that task migration and job queue scheduling can be ef-

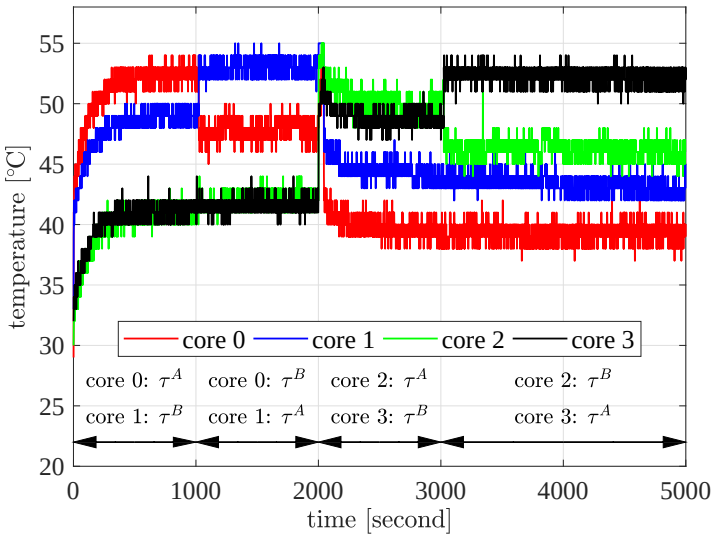


Figure 5.7: The temperatures of the cores when a hot task τ^A and a cool task τ^B are executed on different cores.

fective in controlling the temperature. Thermal management policies can select proper tasks to the cores according to their temperatures for thermal balance. Second, the temperature of a core is also influenced by other cores. In the first 2000 seconds, core 2 and core 3 have no workload but are both heated up by core 0 and core 1. Third, similar to previous experiment, the temperature of different cores can be different even for same workload. Fig. 5.7 shows that core 2 is less sensitive to the hot task τ^A, compared to other cores.

In the third experiment, we implement three thermal-aware management policies in McFTP with 33 benchmarks from the CPU stress tool `Stress-ng`. The first two managements are those proposed in Chapter 3 and Chapter 4, namely, O-PBOO and APTM. The third approach, termed as BWS [24], is an online one and switches the core to sleep state dynamically. For each policy, we run the benchmark for 100 seconds and wait for 100 seconds before next benchmark to cool the processor. The average temperatures of the four cores in different cases are shown in Fig. 5.8. We can observe that APTM outperforms BWS and O-PBOO in all cases. As seen in the figure, APTM achieves a maximal 8 K and an average 5 K temperature reductions compared to BWS, respectively. Another observation is the temperature of the processor changes when different benchmarks are executed. This further strengthens the conclusion made in the second experiment that the temperature strongly depends on the workload it executes.

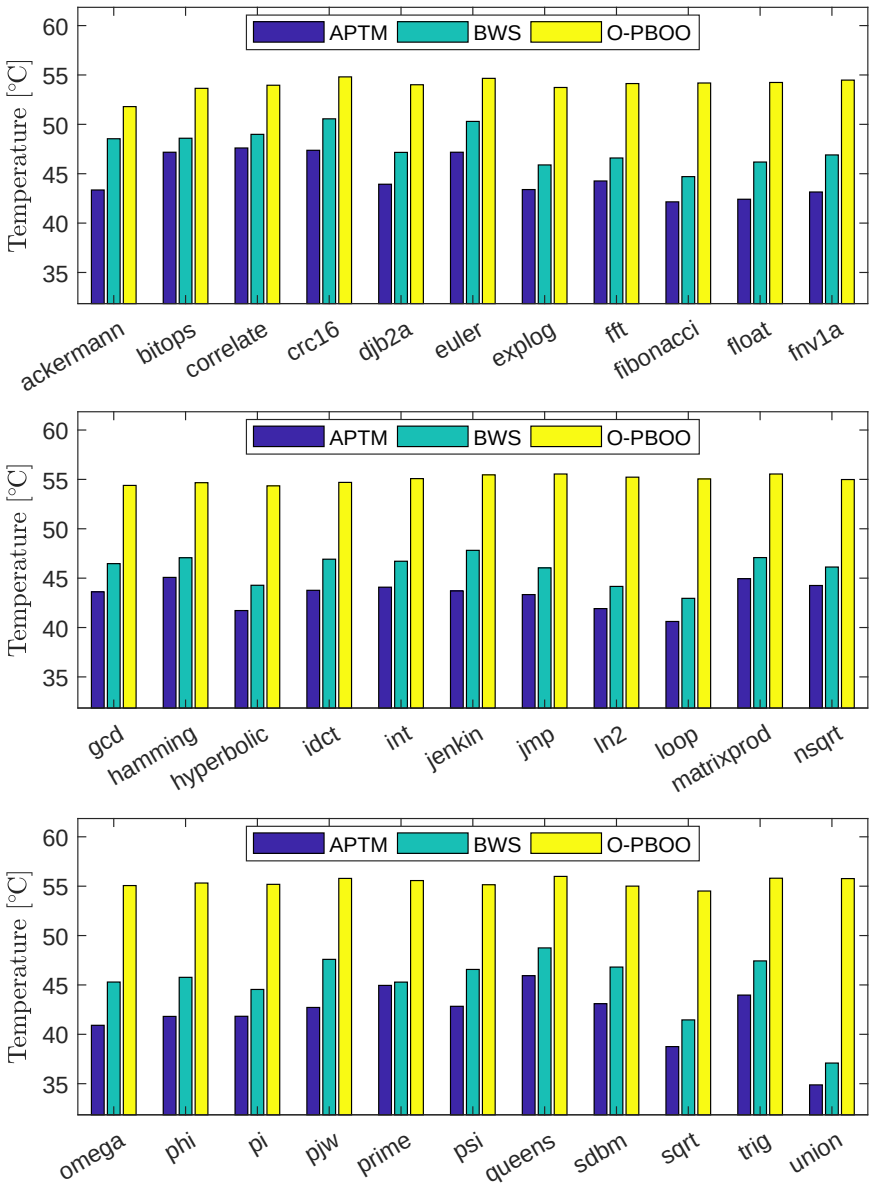


Figure 5.8: The temperatures of APTM, PBOO and BWS for the benchmark set.

5.8.2 Efficiency Experiments

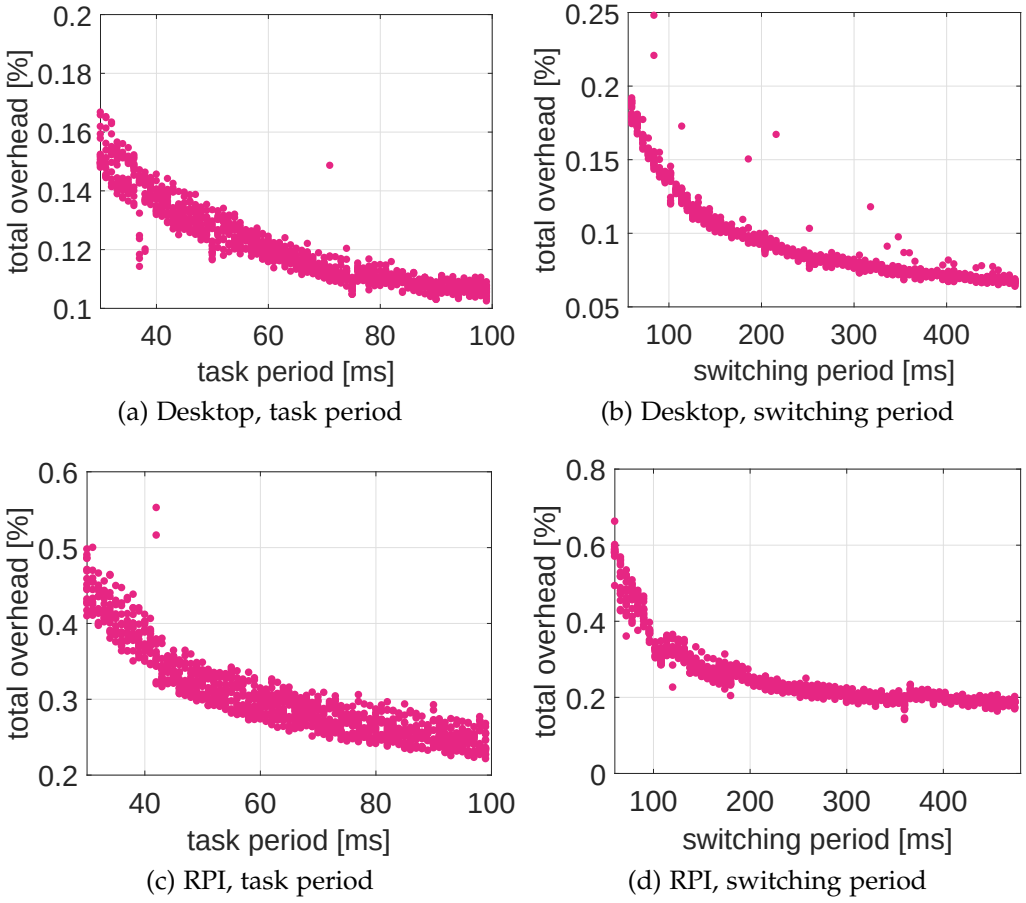
In this section, we study the efficiency of McFTP by reporting the overhead introduced by the framework in different scenarios. To study the efficiency on platforms with different computing capacities, in addition to the aforementioned Dell platform, we also use another embedded environment, a Raspberry Pi (RPI) Model B v1.2 with a 1.2GHz 64-bit

quad-core ARMv8 CPU running the Linux 4.1.19-v7 kernel. In this platform, McFTP framework is compiled by G++ 5.4.0 with optimization level O3 turned on.

McFTP has the following roles. The first one is to execute the user-defined tasks. The second one is to run the thermal management policy, if the policy works in run-time. Anything else can be considered as overhead. Specifically, we consider the sum of the CPU times spent by the power manager, temperature watcher, dispatchers and the checkpoints in workers as the overhead of McFTP. The overhead is incurred by creating and registering new jobs, reading thermal sensor interface, managing job queues, parsing state tables, sending de/activate signals, writing the frequency controlling interface and checking the state of the checkpoints. Since the overhead of checkpoints depends on how the designer programs the task code, i.e., the number of checkpoints, we first study the overhead of McFTP without checkpoints and then report the overhead of checkpoints separately.

In the first experiment, we do not consider the overhead of checkpoints. Therefore, the total overhead is the sum of the CPU times spent by the power manager, temperature watcher and dispatchers. We investigate how the overhead varies when (1) the job arriving period changes and (2) the power state switching frequency changes. For the first scenario, we vary the arriving period of jobs from 30ms to 100ms. In the second scenario, a two-state state table is used and the switching period increases from 60ms to 480ms with step 6ms. In each scenario, 750 task-sets are generated. Each task set contains five tasks with the same period. The total utilization of the task set is set as 0.5. The execution times of the tasks are randomly chosen between 1ms and the period. The experiment run-time is set as 10 seconds. The overhead is measured using the POSIX-CPU-timers for the power manager, temperature watcher and dispatcher threads, normalized over the total run-time. Fig. 5.9 shows the measured overheads plotted against the job arrival period and state switching period, for both platforms. From the figure, we can make following observations.

- *Task period-dependence*. The total overhead decreases when task period increases. This is expected since less jobs are created and managed.
- *Power state switching period-dependence*. Similar to the above observation, increasing the switching period mainly decreases the overhead incurred by the power manager thread.



(a) Desktop, task period

(b) Desktop, switching period

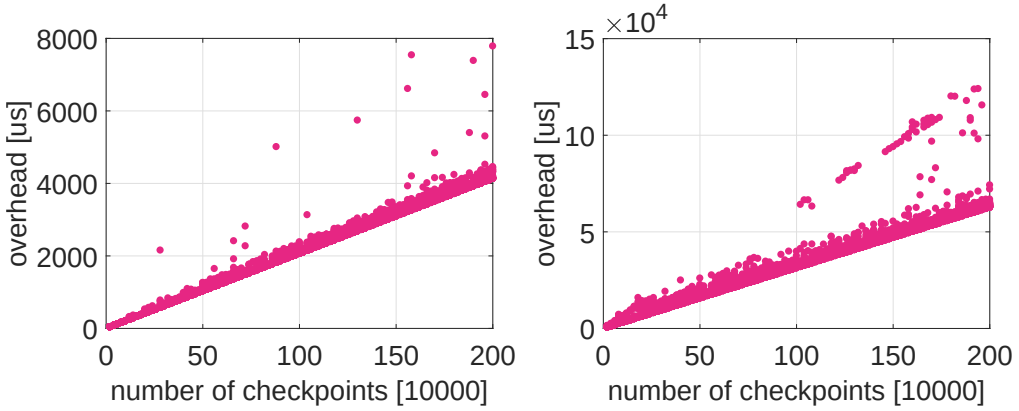
(c) RPI, task period

(d) RPI, switching period

Figure 5.9: McFTP overhead in different scenarios on two platforms having different computing capabilities.

- *Platform-dependence*. The overhead is higher (around 3×) on the Raspberry Pi platform than the overhead on Dell desktop platform. However, the overhead is still below 1% of the total run-time.

The second experiment investigates the overhead incurred by the checkpoints in our framework. We study how the overhead changes when the total number of checkpoints increases. The two types of checkpoints, i.e., the suspend and preempt checkpoints, have same number in the experiment. We vary the total number from 20000 to two million with step 20000 and repeat the experiment 50 times for each scenario. Again, we adopt the POSIX-CPU-timers to measure the time spent by checkpoints. Fig. 5.10 plots the overheads of different numbers of checkpoints. Following observations can be made from the figure.



(a) Desktop (b) RPI

Figure 5.10: Checkpoints overhead for different platforms.

- *number-dependence*. The overhead increases (approximately) linearly with respect to the total number of checkpoints, which is straightforward as the framework has to check more states.
- *Platform-dependence*. On the Raspberry Pi platform, the overhead is higher (around 10×) than that on Dell desktop platform.

5.9 Summary

In this chapter, we present the Multi-core Fast Thermal Prototyping (McFTP) framework, a new thermal framework for evaluating general thermal management policies. The variety of thermal management policies is supported by pre-implementing a set of widely used temperature control mechanisms and combining them freely. An intermediate interface named configuration manipulation interface is defined to separate the thermal management policies from low-level implementations. McFTP is designed in user-space and has little interaction to the kernel-space, thus supporting a large variety of target platforms. We implement McFTP with four basic temperature control mechanisms on top of POSIX-compliant operating systems. Its effectiveness is demonstrated by implementing two existing two works on a Dell desktop platform with a four-core Inter-I7 processor. We also investigate the efficiency of McFTP by reporting its overheads on two platforms, i.e., the Dell platform and a Raspberry Pi.

Chapter 6

Conclusion

This chapter summarizes the main results of this thesis and discusses potential directions of future research.

6.1 Main Results

The aim of this thesis is to address the emerging thermal challenges for hard real-time systems with the periodic thermal management. The challenges are categorized into a few topics and the corresponding solutions are provided. In summary, the main contributions are listed in the following:

- We present the periodic thermal management (PTM) to minimize the peak temperature for single-core real-time systems. We adopt the well-known Fourier's law to model the temperature evolution and offer a closed-form solution of the peak temperature. To find the optimal periodic thermal management scheme which can minimize the peak temperature under real-time constraints, two algorithms with different accuracy levels are proposed. One searches the optimal solution brutally in the exploration space. The other one adopts the bounded delay function to calculate an approximated solution. The proposed thermal management can be implemented easily on real-time systems having little computing power by simply using a hardware timer.
- We investigate the offline implementation of the periodic thermal management on pipelined hard real-time systems. To avoid accounting the burst in the incoming stream more than once, the

principle pay-burst-only-once is adopted to obtain a tight bound of the aggregate service service. We analyze and present theoretical results for calculating the peak temperature of multi-core systems under periodic thermal managements. Two algorithms are proposed to derive the peak temperature with different levels of efficiency and accuracy. For each peak temperature algorithm, we present an algorithm for determining PTM schemes, targeting reduction of the peak temperature under hard real-time constraints.

- We also study adopting online thermal managements to dynamically adjust the periodic on/off patterns for pipelined multi-core systems. Based on the extended pay-burst-only-once principle, we present a sufficient real-time condition for online periodic thermal managements to guarantee deadline constraints of current and future events. With this condition, the online thermal management is formalized as an optimization problem which is solved at each online adaption instant. Under the guidance of the unique thermal properties of the system, an heuristic scheme is proposed to solve the problem online with negligible overhead.
- Finally, we investigate how to evaluate multi-core thermal managements that are based on different temperature control mechanisms on real hardware platforms in an efficient manner. We present a user-space thermal framework named Multi-core Fast Thermal Prototyping, which can implement customized thermal management with minimal effort. A set of common temperature control mechanisms is implemented as a thermal library for users. To isolate the thermal management policies and the low-level implementation on the target platform, we also defined an intermediate interface termed as configuration manipulation interface.

6.2 Future Perspectives

The contributions presented in this thesis provide partial solutions for a few thermal management challenges for hard real-time systems. Nevertheless, there still exists potential for further extensions and improvements. Several possible directions for future research are identified in the following list.

- *Multi-objective thermal management*
  There is a strong correlation between temperature and power because heat is the primary dissipation form of the consumed energy.

System performance can also be influenced indirectly by thermal managements as the temperature is controlled via adjusting the execution of workloads. In other words, energy, performance and temperature are correlated with each other. Therefore, an promising research direction is the design of new thermal managements which can optimize energy, performance and temperature simultaneously. Because of the overall complexity of such problem, evolutionary algorithm would be a viable candidate as a starting point.

- *Refining the timing model*
  The arrival and service curves are adopted to model workloads and system resources in this thesis. They provide upper and lower bounds on the non-determinism in the stream in any time interval with specified length. While presenting high level of abstraction, they also have an clear drawback: they cannot capture any timing correlation between two streams. For instance, the timing offsets and the PTM phase shifts between different cores in a multi-core processor are completely ignored in the service curve. This severe abstraction loss leads to conservative decisions when searching the optimal PTM schemes under hard real-time constraints. Therefore, refining the timing model such that the correlation between different cores in the time domain are utilized would be high beneficial to further lowering the temperature and improving system performance.

- *Coupling with other mechanisms*
  In this thesis, we focus on adopting dynamic power management to reduce the power dissipation and thus control the temperature of hard real-time systems. This coupling is intuitive since temperature strongly depends on the power. However, there also exist other temperature control mechanisms, e.g., job scheduling, task migration, etc. One interesting topic would be combining two or more of these mechanisms to optimize the temperature under real-time constraints.

- *Temperature estimation*
  In chapter 5, to reflect the real effectiveness of thermal managements, we utilize the temperatures obtained from the temperature sensors built for each physical core as the criterion to evaluate different thermal managements. However, due to the limits of die size, costs, etc., there are still processors that do not install temperature sensor for all the cores. For these processors, soft tem-

perature estimators are a promising solution. Moreover, they can also provide finer spatial and temporal granularities than physical temperature sensors. Developing an efficient and accurate temperature estimator for multi-core processors is a potential research direction.

Bibliography

[1] Approximation of curves. http://www.mpa.ethz.ch/static/tutorial.html.

[2] Masud Ahmed, Nathan Fisher, Shengquan Wang, and Pradeep Hettiarachchi. Minimizing peak temperature in embedded real-time systems via thermal-aware periodic resources. *Sustainable Computing: Informatics and Systems*, 1(3):226–240, 2011.

[3] Andrea Alimonda, Andrea Acquaviva, and Salvatore Carta. Temperature and leakage aware power control for embedded streaming applications. In *11th EUROMICRO Conference on Digital System Design Architectures, Methods and Tools*, pages 107–114. IEEE, 2008.

[4] EA Amerasekera and Farid N. Najm. *Failure mechanisms in semiconductor devices*. Wiley, 1997.

[5] David Atienza, Pablo Garcia Del Valle, Giacomo Paci, Francesco Poletti, Luca Benini, Giovanni De Micheli, and Jose Manuel Mendias. A fast hw/sw fpga-based thermal emulation framework for multi-processor system-on-chip. In *43rd ACM/IEEE Design Automation Conference*, pages 618–623. IEEE, 2006.

[6] Nikhil Bansal, Tracy Kimbrel, and Kirk Pruhs. Speed scaling to manage energy and temperature. *Journal of the ACM*, 54(1):3, 2007.

[7] Nikhil Bansal and Kirk Pruhs. Speed scaling to manage temperature. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 460–471. Springer, 2005.

[8] Min Bao, Alexandru Andrei, Petru Eles, and Zebo Peng. Temperature-aware voltage selection for energy optimization. In *Proceedings of the conference on Design, automation and test in Europe*, pages 1083–1086. ACM, 2008.

[9] Andrea Bartolini, Matteo Cacciari, Andrea Tilli, Luca Benini, and Matthias Gries. A virtual platform environment for exploring power, thermal and reliability management control strategies in high-performance multicores. In *Proceedings of the 20th symposium on Great lakes symposium on VLSI*, pages 311–316. ACM, 2010.

[10] Sanjoy K Baruah, Alan Burns, and Robert I Davis. Response-time analysis for mixed criticality systems. In *32nd Real-Time Systems Symposium*, pages 34–43. IEEE, 2011.

[11] Mark Benson. *The Art of Software Thermal Management for Embedded Systems*. Springer, 2014.

[12] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R Hower, Tushar Krishna, Somayeh Sardashti, et al. The gem5 simulator. *SIGARCH Computer Architecture News*, 39(2):1–7, 2011.

[13] James R Black. Electromigration failure modes in aluminum metallization for semiconductor devices. *Proceedings of the IEEE*, 57(9):1587–1594, 1969.

[14] Shekhar Borkar. Design challenges of technology scaling. *IEEE micro*, 19(4):23–29, 1999.

[15] David Brooks and Margaret Martonosi. Dynamic thermal management for high-performance microprocessors. In *The Seventh International Symposium on High-Performance Computer Architecture*, pages 171–182. IEEE, 2001.

[16] David Brooks, Vivek Tiwari, and Margaret Martonosi. *Wattch: A framework for architectural-level power analysis and optimizations*, volume 28. ACM, 2000.

[17] J Adam Butts and Gurindar S Sohi. A static power model for architects. In *33rd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 191–201. IEEE, 2000.

[18] Salvatore Carta, Andrea Acquaviva, Pablo G Del Valle, David Atienza, Giovanni De Micheli, Fernando Rincon, Luca Benini, and Jose M Mendias. Multi-processor operating system emulation framework with thermal feedback for systems-on-chip. In *Proceedings of the 17th Great Lakes symposium on VLSI*, pages 311–316. ACM, 2007.

[19] Salvatore Carta, Andrea Alimonda, Alessandro Pisano, Andrea Acquaviva, and Luca Benini. A control theoretic approach to energy-efficient pipelined computation in mpsocs. *Transactions on Embedded Computing Systems*, 6(4):27, 2007.

[20] Thidapat Chantem, X Sharon Hu, and Robert P Dick. Online work maximization under a peak temperature constraint. In *Proceedings of the ACM/IEEE international symposium on Low power electronics and design*, pages 105–110. ACM, 2009.

[21] Thidapat Chantem, X Sharon Hu, and Robert P Dick. Temperature-aware scheduling and assignment for hard real-time applications on mpsocs. *Transactions on Very Large Scale Integration Systems*, 19(10):1884–1897, 2011.

[22] Gang Chen, Kai Huang, Christian Buckl, and Alois Knoll. Energy optimization with worst-case deadline guarantee for pipelined multiprocessor systems. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 45–50. EDA Consortium, 2013.

[23] Gang Chen, Kai Huang, Christian Buckl, and Alois Knoll. Applying pay-burst-only-once principle for periodic power management in hard real-time pipelined multiprocessor systems. *Transactions on Design Automation of Electronic Systems*, 20(2):26, 2015.

[24] Gang Chen, Kai Huang, and Alois Knoll. Adaptive dynamic power management for hard real-time pipelined multiprocessor systems. In *20th International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 1–10. IEEE, 2014.

[25] Jian-Jia Chen, Shengquan Wang, and Lothar Thiele. Proactive speed scheduling for real-time tasks under thermal constraints. In *15th Real-Time and Embedded Technology and Applications Symposium*, pages 141–150. IEEE, 2009.

[26] Renzhi Chen, Peter R Lewis, and Xin Yao. Temperature management for heterogeneous multi-core fpgas using adaptive evolutionary multi-objective approaches. In *International Conference on Evolvable Systems*, pages 101–108. IEEE, 2014.

[27] Long Cheng, Kai Huang, Gang Chen, Biao Hu, and Alois Knoll. Periodic thermal management for hard real-time systems. In *10th International Symposium on Industrial Embedded Systems*, pages 1–10. IEEE, 2015.

[28] Long Cheng, Kai Huang, Gang Chen, Biao Hu, and Alois Knoll. Minimizing peak temperature for pipelined hard real-time systems. In *Design, Automation Test in Europe Conference Exhibition*. European Design and Automation Association, 2016.

[29] Simone Corbetta, Davide Zoni, and William Fornaciari. A temperature and reliability oriented simulation framework for multi-core architectures. In *Computer Society Annual Symposium on VLSI*, pages 51–56. IEEE, 2012.

[30] Hewlett-Packard Corporation, Intel Corporation, Microsoft Corporation, Phoenix Technologies Ltd, and Toshiba Corporation. Advanced configuration and power interface specification. revision 5.0 errata a. http://www.acpi.info/DOWNLOADS/ACPIspec50.pdf, 2013.

[31] Sony Semiconductor Solutions Corporation. Sony semiconductor quality and reliability handbook. http://www.sony-semicon.co.jp/products_en/quality/pdf/Handbook_e_201604.pdf, 2016.

[32] Marco Cox, Amit Kumar Singh, Ajit Kumar, and Henk Corporaal. Thermal-aware mapping of streaming applications on 3d multi-processor systems. In *11th Symposium on Embedded Systems for Real-time Multimedia*, pages 11–20. IEEE, 2013.

[33] Matthew Curtis-Maury, Karan Singh, Sally A McKee, Filip Blagojevic, Dimitrios S Nikolopoulos, Bronis R De Supinski, and Martin Schulz. Identifying energy-efficient concurrency levels using machine learning. In *International Conference on Cluster Computing*, pages 488–495. IEEE, 2007.

[34] Anup Das, Akash Kumar, and Bharadwaj Veeravalli. Temperature aware energy-reliability trade-offs for mapping of throughput-constrained applications on multimedia mpsocs. In *Proceedings of the conference on Design, Automation & Test in Europe*, page 102. European Design and Automation Association, 2014.

[35] Robert H Dennard, Fritz H Gaensslen, V Leo Rideout, Ernest Bassous, and Andre R LeBlanc. Design of ion-implanted mosfet's with very small physical dimensions. *Journal of Solid-State Circuits*, 9(5):256–268, 1974.

[36] Ahad Mozafari Fard, Mehdi Ghasemi, and Mehdi Kargahi. Response-time minimization in soft real-time systems with temperature-affected reliability constraint. In *CSI Symposium on Real-Time and Embedded Systems and Technologies*, pages 1–8. IEEE, 2015.

[37] Markus Fidler. Extending the network calculus pay bursts only once principle to aggregate scheduling. In *International Workshop on Quality of Service in Multiservice IP Networks*, pages 19–34. Springer, 2003.

[38] Nathan Fisher, Jian-Jia Chen, Shengquan Wang, and Lothar Thiele. Thermal-aware global real-time scheduling on multicore systems. In *15th Real-Time and Embedded Technology and Applications Symposium*, pages 131–140. IEEE, 2009.

[39] Yong Fu, Nicholas Kottenstette, Chenyang Lu, and Xenofon D Koutsoukos. Feedback thermal control of real-time systems on multicore processors. In *Proceedings of the tenth international conference on Embedded software*, pages 113–122. ACM, 2012.

[40] Dayan Adionel Guimaraes. *Digital transmission: a simulation-aided introduction with VisSim/Comm*. Springer Science & Business Media, 2010.

[41] Stephen Gunther, Frank Binns, Douglas M Carmean, and Jonathan C Hall. Managing the impact of increasing microprocessor power consumption. *Intel Technology Journal*, 5(1):1–9, 2001.

[42] Vinay Hanumaiah and Sarma Vrudhula. Temperature-aware dvfs for hard real-time applications on multicore processors. *Transactions on Computers*, 61(10):1484–1494, 2012.

[43] Pradeep M Hettiarachchi, Nathan Fisher, Masud Ahmed, Le Yi Wang, Shinan Wang, and Weisong Shi. The design and analysis of thermal-resilient hard-real-time systems. In *18th Real-Time and Embedded Technology and Applications Symposium*, pages 67–76. IEEE, 2012.

[44] Pradeep M Hettiarachchi, Nathan Fisher, Masud Ahmed, Le Yi Wang, Shinan Wang, and Weisong Shi. A design and analysis framework for thermal-resilient hard real-time systems. *Transactions on Embedded Computing Systems*, 13(5s):146, 2014.

[45] Pradeep M Hettiarachchi, Nathan Fisher, and Le Yi Wang. Achieving thermal-resiliency for multicore hard-real-time systems. In *25th Euromicro Conference on Real-Time Systems*, pages 37–46. IEEE, 2013.

[46] John Howard, Saurabh Dighe, Yatin Hoskote, Sriram Vangal, David Finan, Gregory Ruhl, Devon Jenkins, Howard Wilson, Nitin Borkar, Gerhard Schrom, et al. A 48-core ia-32 message-passing processor with dvfs in 45nm cmos. In *International Solid-State Circuits Conference Digest of Technical Papers*, pages 108–109. IEEE, 2010.

[47] Ming-yu Hsieh, Arun Rodrigues, Rolf Riesen, Kevin Thompson, and William Song. A framework for architecture-level power, area, and thermal simulation and its application to network-on-chip design exploration. *SIGMETRICS Performance Evaluation Review*, 38(4):63–68, 2011.

[48] Kai Huang, Gang Chen, Christian Buckl, and Alois Knoll. Conforming the runtime inputs for hard real-time embedded systems. In *Proceedings of the 49th Annual Design Automation Conference*, pages 430–436. ACM, 2012.

[49] Kai Huang, Luca Santinelli, Jian-Jia Chen, Lothar Thiele, and Giorgio C Buttazzo. Adaptive dynamic power management for hard real-time systems. In *30th Real-Time Systems Symposium*, pages 23–32. IEEE, 2009.

[50] Kai Huang, Luca Santinelli, Jian-Jia Chen, Lothar Thiele, and Giorgio C Buttazzo. Periodic power management schemes for real-time event streams. In *Proceedings of the 48th Conference on Decision and*

Control, held jointly with the 28th Chinese Control Conference, pages 6224–6231. IEEE, 2009.

[51] Kai Huang, Luca Santinelli, Jian-Jia Chen, Lothar Thiele, and Giorgio C Buttazzo. Applying real-time interface and calculus for dynamic power management in hard real-time systems. *Real-Time Systems*, 47(2):163–193, 2011.

[52] Wei Huang, Shougata Ghosh, Siva Velusamy, Karthik Sankaranarayanan, Kevin Skadron, and Mircea R Stan. Hotspot: A compact thermal modeling methodology for early-stage vlsi design. *Transactions on Very Large Scale Integration Systems*, 14(5):501–513, 2006.

[53] Haris Javaid, Muhammad Shafique, Sri Parameswaran, and Jörg Henkel. Low-power adaptive pipelined mpsocs for multimedia: an h. 264 video encoder case study. In *Proceedings of the 48th Design Automation Conference*, pages 1032–1037. ACM, 2011.

[54] Joonho Kong, Sung Woo Chung, and Kevin Skadron. Recent thermal management techniques for microprocessors. *ACM Computing Surveys*, 44(3):13, 2012.

[55] Pratyush Kumar and Lothar Thiele. Cool shapers: shaping real-time tasks for improved thermal guarantees. In *Proceedings of the 48th Design Automation Conference*, pages 468–473. IEEE, 2011.

[56] Pratyush Kumar and Lothar Thiele. Thermally optimal stop-go scheduling of task graphs with real-time constraints. In *Asia and South Pacific Design Automation Conference*, pages 123–128. IEEE Press, 2011.

[57] V Lakshminarayanan and N Sriraam. The effect of temperature on the reliability of electronic components. In *International Conference on Electronics, Computing and Communication Technologies*, pages 1–6. IEEE, 2014.

[58] Kai Lampka et al. Keep it slow and in time: Online dvfs with hard real-time workloads. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 385–390. IEEE, 2016.

[59] Kai Lampka, Kai Huang, and Jian-Jia Chen. Dynamic counters and the efficient and effective online power management of embedded real-time systems. In *Proceedings of the seventh IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pages 267–276, 2011.

[60] Jean-Yves Le Boudec and Patrick Thiran. *Network calculus: a theory of deterministic queuing systems for the internet*. Springer, 2001.

[61] Edward A Lee and Sanjit A Seshia. Introduction to embedded systems. *A Cyber-Physical*, 2014.

[62] Jong Sung Lee, Kevin Skadron, and Sung Woo Chung. Predictive temperature-aware dvfs. *Transactions on Computers*, 59(1):127–133, 2010.

[63] Jungseob Lee and Nam Sung Kim. Optimizing throughput of power-and thermal-constrained multicore processors using dvfs and per-core power-gating. In *46th ACM/IEEE Design Automation Conference*, pages 47–50. IEEE, 2009.

[64] Jungseob Lee and Nam Sung Kim. Analyzing potential throughput improvement of power-and thermal-constrained multicore processors by exploiting dvfs and pcpg. *Transactions on Very Large Scale Integration Systems*, 20(2):225–235, 2012.

[65] Sheng Li, Jung Ho Ahn, Richard D Strong, Jay B Brockman, Dean M Tullsen, and Norman P Jouppi. Mcpat: an integrated power, area, and timing modeling framework for multicore and manycore architectures. In *42nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 469–480. IEEE, 2009.

[66] Yingmin Li, K Skadron, D Brooks, and Zhigang Hu. Performance, energy, and thermal considerations for smt and cmp architectures. In *11th International Symposium on High-Performance Computer Architecture*, pages 71–82. IEEE, 2005.

[67] Chien-Hui Liao and Charles H-P Wen. Thermal-constrained task scheduling on 3-d multicore processors for throughput-and-energy optimization. *Transactions on Very Large Scale Integration Systems,*, 23(11):2719–2723, 2015.

[68] Y. Liu, R. P. Dick, L. Shang, and H. Yang. Accurate temperature-dependent integrated circuit leakage power estimation is easy. In *Design, Automation Test in Europe Conference Exhibition*, pages 1–6, April 2007.

[69] Morteza Mohaqeqi, Mehdi Kargahi, and Kazim Fouladi. Stochastic thermal control of a multicore real-time system. In *24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, pages 208–215. IEEE, 2016.

[70] Rajarshi Mukherjee and Seda Ogrenci Memik. Physical aware frequency selection for dynamic thermal management in multi-core systems. In *Proceedings of the IEEE/ACM international conference on Computer-aided design*, pages 547–552. ACM, 2006.

[71] Rajarshi Mukherjee, Seda Ogrenci Memik, and Gokhan Memik. Peak temperature control and leakage reduction during binding in high level synthesis. In *Proceedings of the International Symposium on Low power electronics and design*, pages 251–256. ACM, 2005.

[72] Fabrizio Mulas, David Atienza, Andrea Acquaviva, Salvatore Carta, Luca Benini, and Giovanni De Micheli. Thermal balancing policy for multiprocessor stream computing platforms. *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(12):1870–1882, 2009.

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

[74] Gergely Nagy and András Poppe. Simulation framework for multilevel power estimation and timing analysis of digital systems allowing the consideration of thermal effects. In *13th Latin American Test Workshop*, pages 1–5. IEEE, 2012.

[75] Hyunok Oh and Soonhoi Ha. Hardware-software cosynthesis of multi-mode multi-task embedded systems with real-time constraints. In *Proceedings of the tenth international symposium on Hardware/software codesign*, pages 133–138. ACM, 2002.

[76] Santiago Pagani, Heba Khdr, Waqaas Munawar, Jian-Jia Chen, Muhammad Shafique, Minming Li, and Jörg Henkel. Tsp: thermal safe power: efficient power budgeting for many-core systems in dark silicon. In *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis*, page 10. ACM, 2014.

[77] Michael Pecht. The influence of temperature on microelectronic device failure mechanisms. phase 2. Technical report, DTIC Document, 1993.

[78] Simon Perathoner, Kai Lampka, Nikolay Stoimenov, Lothar Thiele, and Jian-Jia Chen. Combining optimistic and pessimistic dvs scheduling: An adaptive scheme and analysis. In *Proceedings of the International Conference on Computer-Aided Design*, pages 131–138. IEEE Press, 2010.

[79] Xiaoke Qin and Prabhat Mishra. Tecs: Temperature-and energy-constrained scheduling for multicore systems. In *27th International Conference on VLSI Design and 13th International Conference on Embedded Systems*, pages 216–221. IEEE, 2014.

[80] Devendra Rai, Hoeseok Yang, Iuliana Bacivarov, Jian-Jia Chen, and Lothar Thiele. Worst-case temperature analysis for real-time systems. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 1–6. IEEE, 2011.

[81] Ravishankar Rao and Sarma Vrudhula. Fast and accurate prediction of the steady-state throughput of multicore processors under thermal constraints. *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(10):1559–1572, 2009.

[82] Jose Renau, Basilio Fraguela, James Tuck, Wei Liu, Milos Prvulovic, Luis Ceze, Smruti Sarangi, Paul Sack, Karin Strauss, and Pablo Montesinos. Sesc simulator, 2005.

[83] MohammadSadegh Sadri, Andrea Bartolini, and Luca Benini. Single-chip cloud computer thermal model. In *17th International Workshop on Thermal Investigations of ICs and Systems*, pages 1–6. IEEE, 2011.

[84] Karthik Sankaranarayanan. *Thermal modeling and management of microprocessors*. PhD thesis, University of Virginia, 2009.

[85] Robert Schöne, Daniel Molka, and Michael Werner. Wake-up latencies for processor idle states on current x86 processors. *Computer Science-Research and Development*, 30(2):219–227, 2015.

[86] Lars Schor, Iuliana Bacivarov, Hoeseok Yang, and Lothar Thiele. Worst-case temperature guarantees for real-time applications on multi-core systems. In *Real-Time and Embedded Technology and Applications Symposium*, pages 87–96. IEEE, 2012.

[87] Lui Sha, Marco Caccamo, Renato Mancuso, Jung-Eun Kim, Man-Ki Yoon, Rodolfo Pellizzoni, Heechul Yun, Russel Kegley, Dennis Perlman, Greg Arundale, et al. Single core equivalent virtual machines for hard real—time computing on multicore processors. Technical report, 2014.

[88] Hafiz Fahad Sheikh, Ishfaq Ahmad, and Dongrui Fan. An evolutionary technique for performance-energy-temperature optimized scheduling of parallel tasks on multi-core processors. *Transactions on Parallel and Distributed Systems*, 27(3):668–681, 2016.

[89] Insik Shin and Insup Lee. Compositional real-time scheduling framework. In *25th International Real-Time Systems Symposium*, pages 57–67. IEEE, 2004.

[90] SIA. International technology roadmap for semiconductors (itrs). *Semiconductor Industry Association*, 2015.

[91] Fridtjof Siebert. Multicore systems–challenges for the real-time software developer.

[92] SIXSIGMA. Ionic cleanliness testing. http://www.sixsigmaservices.com/ioniccleanliness.asp, 2016.

[93] Kevin Skadron, Mircea R Stan, Wei Huang, Sivakumar Velusamy, Karthik Sankaranarayanan, and David Tarjan. Temperature-aware microarchitecture. *SIGARCH Computer Architecture News*, 31(2):2–13, 2003.

[94] Greg Taylor. Energy efficient circuit design and the future of power delivery. *Electrical Performance of Electronic Packaging and Systems*, 2009.

[95] ARS TECHNICA. Nvidia denies rumors of faulty chips, mass gpu failures. http://arstechnica.com/hardware/news/2008/07/nvidia-denies-rumors-of-mass-gpu-failures.ars, 2008.

[96] Lothar Thiele, Samarjit Chakraborty, and Martin Naedele. Real-time Calculus for Scheduling Hard Real-time Systems. *International Symposium on Circuits and Systems*, 4:101–104, 2000.

[97] Lothar Thiele, Lars Schor, Iuliana Bacivarov, and Hoeseok Yang. Predictability for timing and temperature in multiprocessor system-on-chip platforms. *Transactions on Embedded Computing Systems*, 12(1s):48, 2013.

[98] Lothar Thiele, Ernesto Wandeler, and Nikolay Stoimenov. Real-time interfaces for composing real-time systems. In *Proceedings of the 6th ACM & IEEE International conference on Embedded software*, pages 34–43, 2006.

[99] EE TIMES. The truth about last year's xbox 360 recall. http://www.eetimes.com/electronicsnews/4077187/The-truth-about-last-year-s-Xbox-360-recall., 2008.

[100] Ivan Ukhov, Min Bao, Petru Eles, and Zebo Peng. Steady-state dynamic temperature analysis and reliability optimization for embedded multiprocessor systems. In *Proceedings of the 49th Annual Design Automation Conference*, pages 197–204. ACM, 2012.

[101] Ernesto Wandeler and Lothar Thiele. Optimal tdma time slot and cycle length allocation for hard real-time systems. In *Asia and South Pacific Design Automation Conference*. IEEE, 2006.

[102] Shengquan Wang and Riccardo Bettati. Reactive speed control in temperature-constrained real-time systems. *Real-Time Systems*, 39(1-3):73–95, 2008.

[103] Tianyi Wang, Ming Fan, Gang Quan, and Shangping Ren. Heterogeneity exploration for peak temperature reduction on multi-core platforms. In *Fifteenth International Symposium on Quality Electronic Design*, pages 107–114. IEEE, 2014.

[104] Jonathan A Winter and David H Albonesi. Addressing thermal nonuniformity in smt workloads. *Transactions on Architecture and Code Optimization*, 5(1):4, 2008.

[105] Yuan Xie and Wei-Lun Hung. Temperature-aware task allocation and scheduling for embedded multiprocessor systems-on-chip (mpsoc) design. *The Journal of VLSI Signal Processing*, 45(3):177–189, 2006.

[106] Hoeseok Yang and Soonhoi Ha. Pipelined data parallel task mapping/scheduling technique for mpsoc. In *Proceedings of the conference on Design, automation and test in Europe*, pages 69–74. European Design and Automation Association, 2009.

[107] Jianxun Yang and Shan Cao. An accurate power and temperature simulation framework for network-on-chip. In *International Conference on Integrated Circuits and Microsystems*, pages 166–171. IEEE, 2016.

[108] Jun Yang, Xiuyi Zhou, Marek Chrobak, Youtao Zhang, and Lingling Jin. Dynamic thermal management through task scheduling. In *International Symposium on Performance Analysis of Systems and software*, pages 191–201. IEEE, 2008.

[109] Dante C Youla. Two observations regarding first-quadrant causal bibo-stable digital filters. *Proceedings of the IEEE*, 78(4):598–603, 1990.

[110] Buyoung Yun, Kang G Shin, and Shige Wang. Thermal-aware scheduling of critical applications using job migration and power-gating on multi-core chips. In *10th International Conference on Trust, Security and Privacy in Computing and Communications*, pages 1083–1090. IEEE, 2011.

[111] Sushu Zhang and Karam S Chatha. Approximation algorithm for the temperature-aware scheduling problem. In *Proceedings of the international conference on Computer-Aided Design*, pages 281–288. IEEE, 2007.

[112] Changyun Zhu, Zhenyu Gu, Li Shang, Robert P Dick, and Russ Joseph. Three-dimensional chip-multiprocessor run-time thermal management. *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(8):1479–1492, 2008.

List of Publications

[1] Long Cheng, Kai Huang, Gang Chen, Biao Hu, and Alois Knoll. Periodic thermal management for hard real-time systems. In *10th IEEE International Symposium on Industrial Embedded Systems*, pages 1–10. IEEE, 2015.

[2] Long Cheng, Kai Huang, Gang Chen, Biao Hu, and Alois Knoll. Minimizing peak temperature for pipelined hard real-time systems. In *Design, Automation Test in Europe Conference Exhibition*. European Design and Automation Association, 2016.

[3] Long Cheng, Kai Huang, Gang Chen, Biao Hu, and Alois Knoll. Mixed-criticality control system with performance and robustness guarantees. In *The 14th IEEE International Conference On Embedded Software And Systems*. IEEE, 2017.

[4] Long Cheng, Zhihao Zhao, Kai Huang, Gang Chen, and Alois Knoll. Mcftp: A framework to explore and prototype multi-core thermal managements on real processors. In *The 14th IEEE International Conference On Embedded Software And Systems*. IEEE, 2017.

[5] Long Cheng, Zhihao Zhao, Kai Huang, and Alois Knoll. Hyper-periodic thermal management for hard real-time systems. In *12th IEEE International Symposium on Industrial Embedded Systems*. IEEE, 2017.

[6] Long Cheng, Zhenshan Bing, Alois Knoll, and Kai Huang. Biologically inspired spiking neural network for autonomous locomotion control of snake-like robots. In *International Journal of Biosensors & Bioelectronics*. MedCrave, 2017.

[7] Zhenshan Bing, Long Cheng, Kai Huang, Mingchuan Zhou, and Alois Knoll. Smooth gait transition of body shape and locomotion speed based on cpg control for snake-like robot. In *IEEE International Conference on Robotics and Automation (ICRA)*, June 2017.

[8] Zhenshan Bing, Long Cheng, Guang Chen, Florian Röhrbein, Kai Huang, and Alois Knoll. Towards autonomous locomotion: Cpg-based control of smooth 3d slithering gait transition of a snake-like robot. *Bioinspiration & Biomimetics*, 12(3):035001, 2017.

[9] Zhenshan Bing, Long Cheng, Anyang Zhong, Feihu Zhang, Kai Huang, and Alois Knoll. Slope angle estimation based on multi-sensor fusion for a snake-like robot. In *20th International Conference on Information Fusion*, Xi'an, P.R. China, July 2017.

[10] Guang Chen, Zhenshan Bing, Florian Rohrbein, Jorg Conradt, Kai Huang, Long Cheng, Zhuangyi Jiang, and Alois Knoll. Towards brain-inspired learning with the neuromorphic snake-like robot and the neurorobotic platform. In *IEEE Transactions on Cognitive and Developmental Systems*, June 2017. accepted.

[11] Biao Hu, Kai Huang, Gang Chen, Long Cheng, and Alois Knoll. Online workload monitoring with the feedback of actual execution time for real-time systems. In *Design, Automation & Test in Europe Conference & Exhibition*, pages 764–769. IEEE, 2017.

[12] Biao Hu, Kai Huang, Gang Chen, Long Cheng, Dongkun Han, and Alois Knoll. Schedulability analysis towards arbitrarily activated tasks in mixed-criticality systems. *Journal of Circuits, Systems and Computers*, 26(10):1750159, 2017.

[13] Gang Chen, Kai Huang, Long Cheng, Biao Hu, and Alois Knoll. Dynamic partitioned cache memory for real-time mpsocs with mixed criticality. *Journal of Circuits, Systems and Computers*, 25(06):1650062, 2016.

[14] Biao Hu, Kai Huang, Gang Chen, Long Cheng, and Alois Knoll. Adaptive workload management in mixed-criticality systems. In *ACM Transactions on Embedded Computing Systems*, May 2016.

[15] Biao Hu, Kai Huang, Gang Chen, Long Cheng, and Alois Knoll. Adaptive runtime shaping for mixed-criticality systems. In *International Conference on Embedded Software*, October 2015.

[16] Biao Hu, Kai Huang, Gang Chen, Long Cheng, and Alois Knoll. Evaluation and improvements of runtime monitoring methods for real-time event streams. In *ACM Transactions on Embedded Computing Systems*, Feb 2016.