Thermal-Aware Resource Management for Embedded Real-Time Systems

Youngmoon Lee, Kang G. Shin
University of Michigan
ymoonlee,kgshin@umich.edu

Hoon Sung Chwa
DGIST
chwahs@dgist.ac.kr

Shige Wang
General Motors R&D
shige.wang@gm.com

ABSTRACT

With an increasing demand for complex and powerful System-on-Chips (SoCs), modern real-time automotive systems face significant challenges in managing on-chip-temperature. We demonstrate, via real experiments, the importance of taking into account *dynamic ambient temperature* and *task-level power dissipation* in resource management so as to meet both thermal and timing constraints. To address this problem, we propose a real-time thermal-aware resource management framework, called RT-TRM. We first introduce a task-level dynamic power model that can capture different power dissipations with a simple task-level parameter called the *activity factor*. We then develop two new scheduling mechanisms, *adaptive parameter assignment* and *online idle-time scheduling*. The former adjusts voltage/frequency levels and task periods according to the varying ambient temperature while preserving feasibility. The latter generates a schedule by allocating idle times efficiently without missing any task/job deadline. By tightly coupling the solutions of these two scheduling mechanisms, we can guarantee both thermal and timing constraints under dynamic ambient temperature variations. We have implemented RT-TRM on an automotive microcontroller to demonstrate the effectiveness of our framework, achieving better resource utilization by 18.2% over other runtime approaches while satisfying both thermal and timing constraints.

1 INTRODUCTION

Thermal-aware resource management has become critical to modern embedded real-time systems like automotive controls and smartphones as they are increasingly implemented on powerful computing platforms with exponentially increasing power density. High on-chip temperatures shorten the platform lifetime and severely degrade its performance and reliability, risking safety (e.g., vehicle breakdown or smartphone explosion). Therefore, we must maintain the processor temperature below the peak temperature constraint while satisfying all application timing constraints.

There are two key thermal issues to consider for embedded real-time systems: 1) dynamically *varying ambient temperature* and 2) task-level *power dissipation*. Our experimental evaluation has shown the ambient temperature of an automotive module to vary highly and dynamically even during a single driving event, and its seasonal temperature varies widely, showing up to a 28°C difference. Moreover, the average power consumed by each application differs up to 140%, according to the results from various automotive benchmark applications (to be detailed in §3).

These dynamic thermal features pose significant challenges in meeting the application timing constraints. In particular, the maximum available computation power varies with the ambient temperature, as the processor temperature depends on its ambient temperature. According to our experimental results (§3), an increase of 14.9°C in ambient temperature results in a maximum reduction of 28.8% in the processor's computation power. In such a case, the processor's thermal constraint may be violated if it executes a task whose average power dissipation exceeds a certain limit. One may reduce the processor temperature by idling or slowing it down, but such an action may also lead to a task/job deadline miss, thus violating the app timing constraint. This calls for adaptive resource management that considers dynamically varying ambient temperature and task-level power dissipation so as to meet both the processor's thermal and the app's timing constraints.

A significant amount of work has been done on real-time thermal-aware scheduling (see [21] for a survey). Existing approaches usually employ DVFS scheduling [8, 11], idle-time scheduling [22], or task scheduling [6] to minimize the peak temperature while guaranteeing the timing constraint. Worst-case temperature analyses [23, 31] have also been proposed for offline guarantees of thermal constraints. The concept of thermal utilization [2] was introduced to capture the thermal impact of periodic real-time tasks on processors. Most of these existing solutions, however, make an assumption of either *fixed* ambient temperature or *constant task-independent* power dissipation. Although there exist real-time feedback thermal control methods to minimize the error between the current processor temperature and the desired temperature by regulating task utilization [15] or frequency [16], they are limited to guarantee the thermal constraints due to temperature overshooting in the feedback thermal controller.

In this paper, we propose a new real-time thermal-aware resource management framework, called RT-TRM, that captures not only varying computation power bounds due to the variations of ambient temperature but also different power demands by different tasks. RT-TRM adaptively makes a parameter assignment (voltage/frequency levels and a task period assignment) and builds a schedule (making the processor idle or active and priority ordering of tasks) so as to meet both thermal and timing constraints.

We first propose a *task-level* dynamic power model that uses a simple task-level parameter called the *activity factor* to capture different power dissipations by different tasks. Building on this dynamic power model, we study the effect of task-execution or processor-idling on the processor temperature. Our model is also validated experimentally with several automotive benchmarks running in various realistic environments. Second, we propose the notions of *dynamic power demand* of a task set and *dynamic power bound* at the ambient temperature and derive the feasibility conditions for a parameter assignment with respect to both thermal and timing constraints. We then develop a runtime adaptive strategy that can preserve feasibility in the presence of ambient temperature variations by adjusting the parameter assignment. Third, building on a feasible parameter assignment, we develop an online scheduling policy that determines not only the processor state (active or idle) but also the execution ordering of tasks in active state. Our scheduling algorithm can reclaim slack (spare capacity) at runtime and allocate it to tasks *in proportion to* their power demands by

considering the fact that a task with higher power dissipation requires more idle time. This way we can meet both thermal and timing constraints with much fewer preemptions. Finally, we have implemented RT-TRM on an automotive microcontroller to show the effectiveness of our framework under varying ambient temperature. Our framework is shown to improve resource utilization by 18.2% over runtime feedback thermal control approaches while guaranteeing both thermal and timing constraints.

This paper makes the following main contributions:

- Demonstration of the importance of accounting for dynamic ambient temperature and task-level power dissipation for real-time thermal-aware resource management (§3);
- Development of a dynamic power model that captures different power dissipations with a simple task-level parameter called the *activity factor* and its experimental validation (§4);
- Development of an adaptive parameter assignment framework under varying ambient temperature while preserving feasibility (§5);
- Development of an online idle-time scheduling algorithm that enables dynamic idle-time allocation with much fewer preemptions while guaranteeing both thermal and timing constraints (§6);
- Implementation and evaluation of the effectiveness of the RT-TRM on an automotive microcontroller (§7).

2 RELATED WORK

Significant efforts have been made on thermal management at both hardware (e.g., architecture design, floorplan) and OS level (e.g., thermal-aware DVFS, scheduling) [21]. In this paper, we focus on OS-level thermal-aware resource management for hard real-time systems, such as cars.

Thermal-aware real-time scheduling is an active subject of research to guarantee timing and thermal constraints under a *constant* environment. DVFS scheduling determines the voltage and frequency of a processor to minimize the peak temperature subject to timing constraints on a single-core [7, 8, 33] or multi-core platforms [11]. Multi-core task scheduling [6] determines task-to-core assignment and scheduling to minimize the peak temperature. Thermal shaping inserts idle periods during task execution at runtime to reduce the peak temperature without missing deadlines [22].

Researchers focus on different task-level power dissipation to reduce the peak temperature [3, 19] or maximize throughput [18, 34] by interleaving the execution of hot and cold tasks. By analyzing such task-level power variability and worst-case scheduling scenario, the peak temperature is derived to guarantee thermal constraint [23]. The concept of thermal utilization was introduced [1, 2] to capture the different thermal impact of periodic real-time tasks.

A few studies consider on adaptive thermal-aware resource management for real-time applications to cope with dynamic environment. Feedback control approaches regulate the processor temperature by adjusting processor utilization [15, 26] or operating frequency [16] subject to the timing constraint.

While researchers have developed task-level scheduling and processor-level thermal control techniques to deal with both thermal and timing requirements, they have not yet addressed large environmental variations and peak temperature caused by task workloads together. To this end, we first verify the significance of these factors in automotive systems. We then develop and validate a task-level thermal model that can capture individual tasks' different power dissipations. Building upon the task-level thermal model, we propose a new thermal-aware resource management scheme that (i) jointly adapts task periods and processor frequency in response to the varying ambient temperature and (ii) schedules tasks to meet both thermal and timing constraints.

3 TARGET SYSTEM, CHALLENGES, AND SOLUTION OVERVIEW

This section presents our target system (§3.1) and introduces the challenges faced therein (§3.2) followed by an overview of our approach (§3.3).

3.1 Target System

We consider a prototypical embedded real-time system running a set of real-time tasks on a computing platform.

Processor and task model. We consider a uniprocessor platform that provides dynamic voltage and frequency scaling (DVFS) with a separate set of discrete frequency/voltage levels. If an operating frequency f is determined within the specified range of $[f_{min}, f_{max}]$, its corresponding voltage V is determined according to a typical implementation principle [12, 16]. We also consider a task set τ composed of implicit-deadline periodic tasks. Each task $\tau_i \in \tau$ is characterized by period p_i and its worst-case execution time (WCET) $e_i(f)$ as a function of operating frequency f. We assume that p_i is adjustable within $[p_i^{min}, p_i^{max}]$ based on typical application elasticity [10, 32]. Such τ_i is assumed to generate a sequence of jobs, once every p_i time-units, with each job needing to complete $e_i(f)$ within a relative deadline of p_i time-units.

Power and thermal model. We consider dynamic power management where the processor is in either *idle* or *active* state. The processor is said to be in active state if it is currently executing a job, or in idle state otherwise. Its power dissipation (P_{proc}) is then expressed as $P_{proc} = P_{leak} + P_{dyn}$, where P_{leak} is the leakage power for the processor to stay ready (in active or idle state) for execution of jobs, and P_{dyn} is the additional dynamic power to execute a job (in active state). The term P_{leak} is modeled as [25]: $P_{leak} = V \cdot (\beta_1 \cdot T + \beta_0)$, where β_1 and β_0 are processor-dependent constants, and T is the processor's temperature. Note that P_{dyn} depends on the task currently running on the processor, and its detailed model will be described in §4.

To translate the processor's power dissipation to its temperature, we use a well-known thermal circuit model [4]. If average processor power and ambient temperature are $P_{proc}(t)$ and $T_{amb}(t)$, respectively, over a time period t, then the processor temperature $T(t)$ at the end of this period is

$$T(t) = T(0) \cdot e^{-\frac{t}{R \cdot C}} + (T_{amb}(t) + P_{proc}(t) \cdot R) \cdot (1 - e^{-\frac{t}{R \cdot C}}), \quad (1)$$

where R and C are the thermal resistance and capacitance, respectively, and $T(0)$ is the initial temperature of the processor. We can observe from Eq. (1) that the temperature will increase/decrease towards and eventually reach $T_{amb}(t) + P_{proc}(t) \cdot R$. We define the steady temperature $T(\infty)$ of the processor as

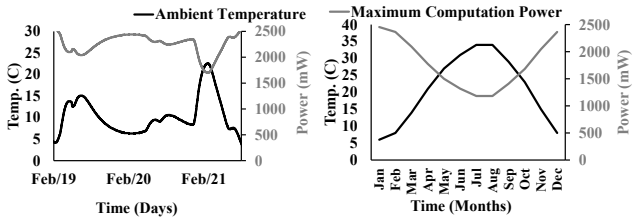


Figure 1: Ambient temperature variations over time and the corresponding available computation power.

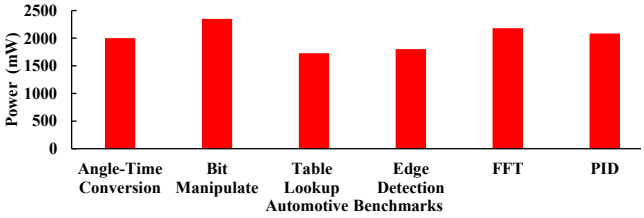


Figure 2: Average power consumptions for various automotive applications.

$$T(\infty) = T_{amb}(t) + P_{proc}(t) \cdot R. \tag{2}$$

3.2 Problem Statement and Motivation

Problem Definition We want to address the following real-time thermal-aware resource management problem.

Definition 1. *Given a task set τ running on a uniprocessor, determine (i) the voltage/frequency (V/f) level, (ii) the period $\{p_i\}$ of task $\tau_i \in \tau$ parameters, and (iii) the schedule of jobs such that (a) temperature $T(t)$ never exceeds the peak temperature constraint T_{max} (thermal constraint), and (b) all jobs of $\tau_i \in \tau$ meet their deadlines for all possible legitimate job arrival sequences (timing constraint).*

To generate a job schedule, we need to determine not only the processor state (active or idle) but also the order of executing jobs in active state. From real embedded systems (e.g., cars and smartphones), we found two key thermal characteristics: (1) dynamic changes in the ambient temperature and (2) different power dissipations by different tasks. These are the primary motivation behind our proposed approach in this paper.

Dynamic changes on ambient temperature. Unlike desktops or data-centers, embedded real-time systems experience a wide range of environmental variations (especially the ambient temperature) during their operation/life. To confirm this fact, we measured the ambient temperature of a vehicle infotainment module embedded in the dashboard over days and months, and the results are plotted in Fig. 1. When the car was driven for several days, the change in the ambient temperature was highly dynamic and fluctuating between 0°C and 23°C. During a single driving event on Feb. 21, 2018 the ambient temperature was increased by up to 180%. The seasonal variation of the ambient temperature is also very wide. A similar phenomenon was also reported in [20] for car engine and transmission control units.

Under such a varying ambient temperature, real-time thermal-aware resource management becomes much more challenging because the processor's temperature is affected by the ambient temperature. Using Eq. (2), we can calculate the maximum processor's power dissipation without exceeding T_{max} for a given ambient temperature T_{amb} as $\frac{T_{max}-T_{amb}}{R}$. The change in the maximum processor's computation power under varying ambient temperature is then plotted in Fig. 1 (see the gray line).[1] For example, on Feb. 21, 2018 as the ambient temperature increased by 14.9°C from 8.3°C, the available processor computation power decreased by 28.8%.

[1]We set T_{max} = 60°C and R = 22°C/W.

Task-level power dissipations. We also measured the processor's average power consumption to run various automotive benchmarks [17], and plotted the results in Fig. 2. Each app is shown to consume a different amount of power. For example, a table lookup task consumes 1726mW, while a bit manipulation task does 2348mW at the maximum processor frequency.[2]

In summary, the available processor's computation power varies with the ambient temperature. In addition, the execution of each task imposes a different power demand on the processor. So, we need to consider different task-level power dissipations and make adaptive parameter assignments and job schedules according to the varying ambient temperature so as to meet both thermal and timing requirements.

3.3 Overview of the Proposed Approach

To solve the real-time thermal-aware resource management problem while considering the varying ambient temperature and diverse task-level power dissipations, we address the following questions:

- **Q1.** How to model power dissipations of different tasks and analyze the impact of their execution on the processor's thermal behavior?
- **Q2.** How to make adaptive parameter assignments under dynamically changing ambient temperature while meeting both thermal and timing constraints?
- **Q3.** How to derive an actual job schedule that satisfies both thermal and timing constraints upon parameter assignment?

To answer Q1, we develop a *task-level* dynamic power model by using a simple task-level parameter called the *activity factor*. Based on this task-level dynamic power model, we analyze the effect of task execution on the processor's temperature, i.e., whether it increases or decreases the processor's temperature. Moreover, we empirically determine the activity factors for several automotive apps and verify our model in various environments, i.e., under different processor-frequency/task-utilization settings, varying ambient temperature, and executing multiple tasks.

To address Q2, we define a *dynamic power demand* of a task set τ that represents the total dynamic power demand by τ at the processor's steady temperature. We also define a *dynamic power bound* function of T_{amb} that represents the maximum processor's

[2]A table lookup operation is used by an engine control module to find an output value corresponding to an input value (e.g., the ignition angle). A bit-manipulation operation is used by a display module where the pixels are moved into a display buffer until the entire buffer is displayed.

dynamic power P_{dyn} at T_{amb} without violating the thermal constraint. Based on these concepts, we derive the feasibility conditions of a task set and formulate an optimization problem that finds a feasible parameter assignment for a given T_{amb}. We also develop a runtime adaptive strategy that can preserve feasibility by adapting the parameter assignment to ambient temperature changes .

To answer Q3, building on a feasible parameter assignment derived by answering Q2, we develop an online scheduling policy. In particular, we calculate the *minimum idle-time* required for the execution of each job with respect to the thermal constraint. We then develop an idle-time scheduling algorithm that can reclaim unused resources at runtime and utilize them to allocate idle time efficiently while meeting all deadlines with the minimum idle-time for each task. As a result, our algorithm can guarantee both thermal and timing constraints with much fewer preemptions.

4 TASK-LEVEL POWER-CONSUMPTION MODEL

We present a task-level power-consumption model that captures different dynamic power dissipations by individual tasks. In particular, we use a simple task-level activity factor to characterize each task's dynamic power dissipation (§4.1) and empirically validate the model using a automotive platform and workloads (§4.2).

4.1 Task-Level Dynamic Power Model

For automotive workloads, power dissipation is found to vary significantly with the executing task (Fig. 2). Since individual tasks programmed with distinct sets of instructions generate different switching activities and dynamic power dissipations, we used a task-level activity factor α_i to capture such different dynamic power P_i consumed by each task τ_i as $P_i = V^2 \cdot f \cdot \alpha_i$. Using this task-level dynamic power model, we can analyze how the processor's temperature changes with the execution of each job/task. Let $T(t)$ ($T_i(t + e_i(f))$) be the temperature at the beginning (end) of the execution of a job of τ_i. Using Eqs. (1) and (2), $T_i(t + e_i(f))$ can be written as:

$$T_i(t + e_i(f)) = T(t) \cdot e^{-\frac{e_i(f)}{R \cdot C}} + T_i^{\infty}(T_{amb}) \cdot (1 - e^{-\frac{e_i(f)}{R \cdot C}}), \quad (3)$$

where $T_i^{\infty}(T_{amb})$ is the steady temperature associated with the execution of τ_i that would be reached if the processor executes τ_i continuously. $T_i^{\infty}(T_{amb})$ can be expressed as:

$$T_i^{\infty}(T_{amb}) = T_{amb} + (P_i + P_{leak}) \cdot R. \quad (4)$$

We observe from Eqs. (3) and (4) that (i) if $T(t) < T_i^{\infty}(T_{amb})$ then the temperature increases towards $T_i^{\infty}(T_{amb})$, and (ii) if $T(t) \geq T_i^{\infty}(T_{amb})$ then the temperature decreases towards $T_i^{\infty}(T_{amb})$. A task τ_i is said to be *hot* if $T_i^{\infty}(T_{amb}) > T_{max}$, or *cold* otherwise. Depending on the ambient temperature T_{amb}, τ_i can become hot or cold.

To consider the effect of idling the processor on its temperature, we let $T_0(t + l)$ denote the temperature at the end of an idle period of length l. Similarly to Eq. (3), $T_0(t + l)$ can be written as:

$$T_0(t + l) = T(t) \cdot e^{-\frac{l}{R \cdot C}} + T_0^{\infty}(T_{amb}) \cdot (1 - e^{-\frac{l}{R \cdot C}}), \quad (5)$$

where $T_0^{\infty}(T_{amb}) = T_{amb} + P_{leak} \cdot R$ is the processor's steady temperature in idle state.

Table 1: Identifying activity factors and maximum errors

Task	Angle	Bit	Table	Edge	FFT	PID
T_i^{∞} (°C)	66.1	73.6	59.9	61.6	69.5	67.8
α_i[3]	0.355	0.446	0.284	0.304	0.435	0.377

So far, we have discussed the thermal effect of continuous execution (idling) of a single job (a processor). Now, let's consider the impact of a schedule of periodic tasks and idle-times. Let $T(t, W(t))$ denote the temperature at the end of a schedule $W(t) = \{w_i(t)\}$, where $w_i(t)$ is the total workload scheduled in $(0, t]$. Then, $T(t, W(t))$ can be written as:

$$T(t, W(t)) = T(0) \cdot e^{-\frac{t}{R \cdot C}} + (T_{amb} + (\sum_{\tau_i} P_i \cdot \frac{w_i(t)}{t} + P_{leak}) \cdot R) \cdot (1 - e^{-\frac{t}{R \cdot C}}), \quad (6)$$

where $\sum_{\tau_i} P_i \cdot \frac{w_i(t)}{t}$ is the average dynamic power consumed by $W(t)$. Note that every task τ_i generates a sequence of jobs executing $e_i(f)$ every p_i time-units, consuming an average dynamic power of $P_i \cdot \frac{e_i(f)}{p_i}$. We can then define the steady temperature $T(\infty, \tau)$ of a task set τ as:

$$T(\infty, \tau) = T_{amb} + (\sum_{\tau_i} P_i \cdot \frac{e_i(f)}{p_i} + P_{leak}) \cdot R. \quad (7)$$

Note that the steady temperature of a task set is independent of its schedule, which serves as a basis for the feasibility condition presented in §5.

Identifying task-level activity factors. To identify the activity factor of each task, we ran automotive benchmarks, one at a time, with 100% resource utilization at the maximum frequency under the room temperature. We then measured the steady temperature and determined each task's activity factor by using Eq. (4), and presented them in Table 1.[4] The activity factor varies greatly with tasks by up to 65%. For example, a table-lookup task with a large number of conditional switches and I/O accesses shows a low activity factor, while a bit-manipulation task with a high instruction-per-cycle rate shows a high activity factor.[5]

4.2 Model Validation

To confirm that the task-level power-concumption model and its thermal effect represent real hardware behaviors, we measured the steady temperature of the processor and compared it with our model's estimation under various settings. In particular, we validated the task-level power model under (i) different processor-frequency/task-utilization settings for each task, (ii) running multiple tasks together, and (iii) different ambient temperatures.

First, as shown in Fig. 3, we varied the task period to achieve the processor utilization ranging from 10% to 90% with a 10% increment as well as the frequency level from 0.4GHz to 1GHz for each task and measured the steady temperature.[6] Fig. 3 shows the measured steady temperature as dotted points while the estimations with our

[4]The detailed experimental setup will be given in §7.
[5]The activity factor α is normalized by the maximum power, i.e., $\alpha = 1$ means the maximum power dissipation.
[6]See §7 for more details of the experimental setup.

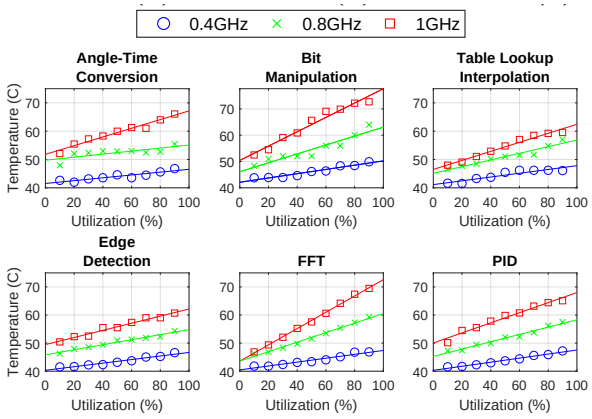


Figure 3: Model validation for each task with varying utilizations.

model are plotted as lines with an error up to 0.65°C.

Second, as shown in Fig. 4, we ran two tasks — bit manipulation and angle-time conversion — together on a single core at 1GHz by varying their utilizations. The result shows that the steady temperature of the processor is linearly increased with each task's utilization (plotted as a linear surface) as formulated in Eq. (7). We also confirmed that the same tendency is observed for different numbers of tasks (i.e., 4 and 8 tasks) with an error up to 1.2°C.

Finally, we validated our model for different ambient temperatures ranging from 20°C to 35°C. For each configuration, we repeated this 10 times with sufficient intervals, showing an error of up to 0.98°C.

5 ADAPTIVE PARAMETER ASSIGNMENT

We now present how to adjust a voltage/frequency level and task period assignment according to the varying ambient temperature, called *Adaptive Parameter Assignment Framework* (APAF). Specifically, we derive feasibility conditions for a parameter assignment, formulate a parameter optimization problem, and introduce a runtime strategy for adapting to the varying ambient temperature.

5.1 Parameter Assignment

We first consider the *feasible parameter assignment* problem for a given ambient temperature.

Definition 2 (Feasible parameter assignment). *Given a task set τ and the ambient temperature T_{amb}, determine V, f and p_i for every $\tau_i \in \tau$ such that if τ is feasible (i.e., meeting the thermal and timing constraints), it remains feasible even with the new parameter assignment.*

To solve this problem, we introduce two conditions for a parameter assignment to be feasible with respect to thermal and timing constraints for a given ambient temperature and formulate an optimization problem to find a feasible parameter assignment.

Feasibility condition. Recall that for a given task set τ, the processor temperature will eventually reach the steady temperature $T(\infty, \tau)$ of τ (defined in Eq. (7)) regardless of its schedule. Therefore, to meet the thermal constraint, the steady temperature $T(\infty, \tau)$ should be lower than or equal to the peak temperature constraint

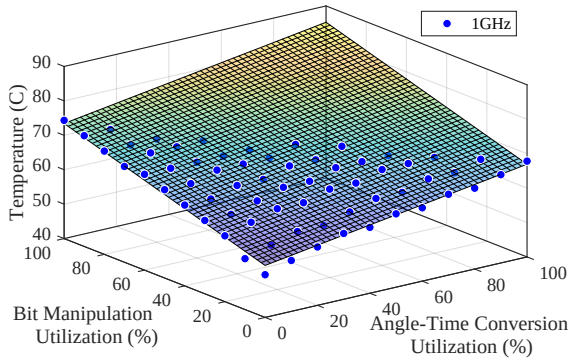


Figure 4: Model validation for running two periodic tasks with varying utilizations (*Bit manipulation, Angle-time Conversion*).

T_{max}, i.e.,

$$\text{C1: } T(\infty, \tau) \leq T_{max}. \tag{8}$$

We define a *dynamic power demand* $PD(\tau)$ of τ as the total dynamic power demand by τ at the steady temperature, which is written as:

$$PD(\tau) = \sum_{\tau_i} P_i \cdot \frac{e_i(f)}{p_i} = V^2 \cdot f \cdot \sum_{\tau_i} \alpha_i \cdot \frac{e_i(f)}{p_i}. \tag{9}$$

We also define a *dynamic power bound* $PB(T_{amb})$ of T_{amb} as the maximum processor's dynamic power at T_{amb} without exceeding T_{max}. We can derive $PB(T_{amb})$ by solving $T(\infty, \tau) = T_{max}$:

$$PB(T_{amb}) = \frac{T_{max} - T_{amb}}{R} - V \cdot (\beta_1 \cdot T_{max} + \beta_0). \tag{10}$$

Using these, the feasibility condition $C1$ with respect to the thermal constraint can be re-written as

$$\overline{\text{C1}}\text{: } PD(\tau) \leq PB(T_{amb}). \tag{11}$$

To meet the timing constraint, we use a well-known exact feasibility analysis by Liu and Layland [24]:

$$\text{C2: } \sum_{\tau_i} \frac{e_i(f)}{p_i} \leq 1. \tag{12}$$

If a parameter assignment satisfies both $\overline{\text{C1}}$ and C2, the steady temperature of τ is guaranteed not to exceed T_{max} without missing any task deadline when a task set is scheduled by an optimal scheduling algorithm. However, as can be seen from Eq. (6), a job schedule may affect a transient temperature $T(t, W(t))$, potentially violating the thermal constraint before reaching the steady temperature. To avoid this situation, we define the minimum idle-time $I_i^{min}(T_{amb})$ required for the execution of each job without violating the thermal constraint and include the term in C2 (to be detailed in §6). Then, the feasibility condition $C2$ can be extended to:

$$\overline{\text{C2}}\text{: } \sum_{\tau_i} \frac{e_i(f) + I_i^{min}(T_{amb})}{p_i} \leq 1. \tag{13}$$

So, if there exists a parameter assignment satisfying both $\overline{\text{C1}}$ and $\overline{\text{C2}}$, we can guarantee that a task set τ is feasible with respect to both thermal and timing constraints.

Parameter optimization. We formulate the parameter assignment problem as an optimization problem subject to the feasibility conditions ($\overline{C1}$ and $\overline{C2}$):

$$\underset{f, p_i}{\textbf{maximize}} \sum_{\tau_i} w_i \cdot \frac{1}{p_i} \tag{14}$$

$$\textbf{s.t.} \quad \overline{C1}\text{: } PD(\tau) = V^2 \cdot f \cdot \sum_{\tau_i} \alpha_i \cdot \frac{e_i(f)}{p_i} \leq PB(T_{amb}) \tag{15}$$

$$\overline{C2}\text{: } \sum_{\tau_i} \frac{e_i(f) + I_i^{min}(T_{amb})}{p_i} \leq 1 \tag{16}$$

$$f \in [f_{min}, ..., f_{max}]. \tag{17}$$

$$\forall \tau_i \quad p_i^{\min} \leq p_i \leq p_i^{\max} \tag{18}$$

As an optimization goal, a QoS function associated with resource usage can be used as in [10, 32]. Our objective in Eq. (14) is to maximize the weighted sum of each task-rate $\frac{1}{p_i}$.[7] Eq. (17) specifies the discrete frequency scaling levels available on the processor. Eq. (18) specifies the minimum and maximum bounds of an allowable task period within $[p_i^{\min}, p_i^{\max}]$. We use linear programming to determine a task period assignment for a given voltage/frequency level starting from the maximum level. If there is no solution, we lower the voltage/frequency level until a feasible solution is found. The computational complexity is $O(m \cdot n^{3.5})$ for n tasks and m frequency scaling levels [27].

5.2 Runtime Parameter Adaptation

We now propose a runtime parameter adaptation strategy that estimates ambient temperature variations and dynamically adjusts the voltage/frequency level and period assignment. To this end, we need to determine when and how to adjust the parameter assignment. We set fixed points of the ambient temperature threshold $\{TS_{amb}(k)\}$, which are determined by

$$TS_{amb}(k+1) = TS_{amb}(k) + \Delta T, \tag{19}$$

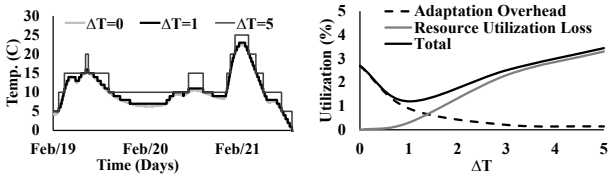
where ΔT is a tolerable ambient temperature range.

Our runtime adaptation policy periodically estimates the ambient temperature and adjusts the parameter assignment whenever the estimated ambient temperature is out of the range $(TS_{amb}(k), TS_{amb}(k+1)]$ for any k. The parameter assignment in $(TS_{amb}(k), TS_{amb}(k+1)]$ is determined by solving the optimization problem in Eq. (14) with the ambient temperature of $TS_{amb}(k+1)$.

The challenge is then how to choose ΔT and estimate the ambient temperature.

Trade-off between resource efficiency and adaptation overhead. In choosing ΔT, there exists a trade-off between resource efficiency and adaptation overhead as shown in Fig. 5. A smaller ΔT can achieve efficient resource utilization with prompt response upon small ambient temperature changes at the expense of high adaptation overhead. To determine the optimal value of ΔT, we analyze the ambient temperature trace in Fig. 1a and compare the runtime overhead and resource efficiency depending on ΔT. Fig. 5a illustrates how our parameter adaptation responds to the varying ambient temperature for different values of ΔT. From the trace, we obtain the total number of parameter adaptations and the total

[7] The value of w_i can be determined according to the importance of each task.



(a) Different Adaptation Interval ΔT (b) Impact of ΔT on Overhead

Figure 5: Runtime adaptation with (a) different different adaptation intervals and (b) trade-off between adaptation overhead and resource efficiency

resource utilization for each value of ΔT. Fig. 5b shows the trade-off between resource efficiency and adaptation overhead, where the adaptation overhead (dotted line) decreases, but the resource utilization loss (grey line) increases as ΔT increases. We set the optimal value of ΔT to the point where the sum of the adaptation overhead and resource utilization loss (solid line) is minimized, which is $\Delta T = 1$°C.

Ambient temperature estimation. To estimate the ambient temperature, RT-TRM monitors the on-chip temperature every δ time-units and estimates the ambient temperature using Eq. (1) as:

$$\begin{aligned} T_{[k]} &= T_{[k-1]} \cdot e^{-\frac{\delta}{R \cdot C}} + (T_{amb} + R \cdot P_{[k]})(1 - e^{-\frac{\delta}{R \cdot C}}) \\ T_{amb} &= \lambda \cdot T_{[k]} + (1-\lambda) \cdot T_{[k-1]} - R \cdot P_{[k]}, \end{aligned} \tag{20}$$

where $T_{[k]}$ and $T_{[k-1]}$ are current and previous temperature measurements, $P_{[k]}$ is the power dissipation during the two consecutive measurements (calculated by using Eq. (6)), and $\lambda = \frac{1}{1 - e^{-\frac{\delta}{R \cdot C}}}$. We set δ to 1s by considering the chip thermal time constant of our evaluation platform [14].

6 ONLINE IDLE-TIME SCHEDULING

So far, we have discussed how to adaptively adjust the processor's voltage/frequency and the task periods under the varying ambient temperature. We now consider how to schedule task/job executions and idle-times in order to meet both thermal and timing requirements. Specifically, we want to address the following problem, which we call the *schedule-generation* problem.

Definition 3 (Schedule generation). *Given the assignment of V, f, and $\{p_i\}$ (with APAF), determine a schedule of job executions and idle-times such that the processor temperature $T(t)$ does not exceed T_{max} at any time t while all jobs of all tasks $\tau_i \in \tau$ meet their deadlines.*

To solve this problem, we must consider two key issues: 1) transient temperature $T(t)$ varies with the task running at any given time, and 2) ambient temperature T_{amb} also affects $T(t)$. Suppose that the processor has reached T_{max} (i.e., $T(t) = T_{max}$) and two tasks — a cold task τ_1 and a hot task τ_2 — are ready to run at time t. If a cold task τ_1 is scheduled, the temperature will decrease since $T_1^{\infty}(T_{amb}) \leq T_{max}$. On the other hand, if a hot task τ_2 starts to run immediately, the temperature will increase (since $T_2^{\infty}(T_{amb}) > T_{max}$), and the thermal constraint will be violated. To avoid the processor temperature exceeding T_{max}, we must idle the processor to drop its temperature to a *safe* temperature before

executing τ_2. With this safe temperature, continued execution of τ_2 will not violate the thermal constraint. The main challenge is then how to derive a safe temperature and schedule idle-times to reach the temperature before executing each hot task. Note that each task has a different power dissipation, so the safe temperature may vary with task. Moreover, the amount of idle time required to reach a safe temperature varies with the ambient temperature. Without a proper idle-time scheduling decision, it may end up with some undesirable situations, such as those where (a) the temperature exceeds T_{max} and/or (b) a task/job deadline miss occurs.

To resolve such issues, we develop a thermal-aware online idle-time scheduling policy that determines idle-times between the execution of tasks to meet both thermal and timing constraints. We assume that tasks is priority-ordered according to the earliest deadline first (EDF) policy. We calculate the *minimum idle-time* required for the execution of each task to avoid the situation (a) and take the minimum idle-time into account in our adaptive parameter assignment to avoid the situation (b). Our proposed online scheduling algorithm then makes the trade-off between the total amount of required idle-time and preemption overhead. In particular, it updates available slack at runtime and effectively utilizes it to allocate more idle-time with much fewer preemptions while guaranteeing both thermal and timing constraints.

Calculation on the minimum idle-time. Now, we describe the relation between the amount of necessary idle-time and the number of preemptions. We first consider the case of executing a hot task τ_i for $e_i(f)$ units without any preemption. We define the safe temperature of τ_i to execute for $e_i(f)$ units at T_{amb} (denoted by $T_i^{safe}(e_i(f), T_{amb})$) as the initial temperature at which the temperature reaches T_{max} after the execution of $e_i(f)$ units. Then, the safe temperature can be derived by solving the term $T(t)$ in Eq. (3) when $T_i(t + e_i(f)) = T_{max}$:

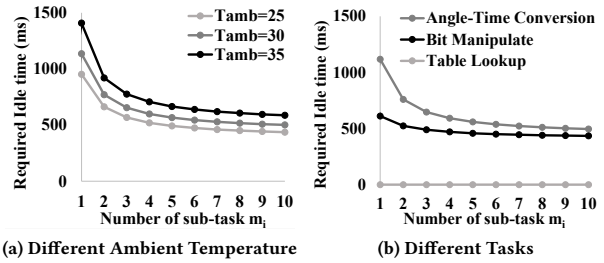
$$T_i^{safe}(e_i(f), T_{amb}) = T_i^{\infty}(T_{amb}) - \frac{T_i^{\infty}(T_{amb}) - T_{max}}{e^{-\frac{e_i(f)}{R \cdot C}}}. \quad (21)$$

Similarly, we can calculate the idle-time necessary to reach $T_i^{safe}(e_i(f), T_{amb})$ (denoted by $t_{idle}(e_i(f), T_{amb})$ by solving the term l in Eq. (5) when $T_0(t + l) = T_i^{safe}(e_i(f), T_{amb})$ and $T(t) = T_{max}$:

$$t_{idle}(e_i(f), T_{amb}) = R \cdot C \cdot ln(\frac{T_{max} - T_0^{\infty}(T_{amb})}{T_i^{safe}(e_i(f), T_{amb}) - T_0^{\infty}(T_{amb})}). \quad (22)$$

Now, let's consider the case where preemption is allowed, i.e., splitting each task τ_i into multiple — m_i ($m_i > 1$) — sub-tasks and inserting idle-time in between. Likewise, by using Eqs. (21) and (22), we can calculate the safe temperature and idle-time required for executing each sub-task for $\frac{e_i(f)}{m_i}$ units. Then, the cumulative idle-time to execute m_i sub-tasks at T_{amb} can be calculated as $m_i \cdot t_{idle}(\frac{e_i(f)}{m_i}, T_{amb})$.

Fig. 6 shows the cumulative idle-time as m_i increases. It is important to observe that the more sub-tasks, the less cumulative idle-time required, as was also observed in [18]. In Fig. 6(a), we can see that the amount of required idle-time also depends the ambient temperature. As shown in Fig. 6(b), each task requires a different



(a) Different Ambient Temperature (b) Different Tasks

Figure 6: Cumulative idle time for (a) different ambient temperature and (b) different tasks decreases with the number of subtasks m_i

amount of idle-time. Note that cold tasks — e.g., a table-lookup task — does not require idle-time. The results shown in Fig. 6 imply that the cumulative idle-time can be reduced by splitting each task into more sub-tasks with frequent idling of the processor. However, the benefit of frequent idling becomes saturated as m_i increases, and the preemption overhead can no longer be ignored. Considering this, we derive the minimum idle-time for a task τ_i (denoted by $I_i^{min}(T_{amb})$) as follows. We calculate a decreasing amount of cumulative idle-time by taking derivative $\frac{\partial}{\partial m_i}(m_i \cdot t_{idle}(\frac{e_i(f)}{m_i}, T_{amb}))$, and find the value of m_i (denoted by m_i^{max}) where the value of the derivative becomes closest to the preemption cost for switching between active and idle states. Then, the minimum idle-time of τ_i can be calculated as

$$I_i^{min}(T_{amb}) = m_i^{max} \cdot t_{idle}(\frac{e_i(f)}{m_i^{max}}, T_{amb}). \quad (23)$$

Guarantee of thermal and timing constraints. For every invocation of a task τ_i, if the minimum idle-time is correctly scheduled before finishing the execution of τ_i, we can guarantee that the thermal constraint is never violated. The question then becomes how to guarantee the timing constraint when all tasks are scheduled together with their minimum idle-time. To address this, we derive a new feasibility condition by incorporating the minimum idle-time for each task. In order for a task set τ to be feasible under both thermal and timing constraints, every job of each task τ_i should have its minimum idle-time (for at least $I_i^{min}(T_{amb})$) and finish its execution (for at most its WCET $e_i(f)$) before its deadline. Then, a new feasibility condition can be derived by extending the utilization-based exact feasibility analysis by Liu and Layland [24]:

$$\sum_{\tau_i} \frac{e_i(f) + I_i^{min}(T_{amb})}{p_i} \leq 1. \quad (24)$$

We include the feasibility condition (Eq. (24)) in the optimization formulation for the parameter assignment presented in §5.1. This way RT-TRM can guarantee both thermal and timing constraints.

Online idle-time scheduling. Building upon the parameter assignment obtained by APAF, if we divide each task τ_i into $m_i(I_i^{min}(T_{amb}))$ sub-tasks and evenly distribute the idle-time

Algorithm 1 Slack calculation

1: $U = \sum_{\tau_i} \frac{e_i(f)+I_i^{min}(T_{amb})}{p_i}$
2: $p = 0$
3: **for** $i = n$ to 1, $\tau_i \in \{\tau_1, ..., \tau_n | d_1(t_{cur}) \leq \cdots \leq d_n(t_{cur})\}$ **do**
4: ▷ In reverse EDF order of tasks
5: $U = U - \frac{e_i(f)+I_i^{min}(T_{amb})}{p_i}$
6: $q_i = \max\left(0, e\_left_i(f) + I_i^{min}(T_{amb}) - (1-U)\cdot(d_i(t_{cur}) - d_1(t_{cur}))\right)$
7: $U = \min\left(1.0, U + \frac{e\_left_i(f)+I_i^{min}(T_{amb})-q_i}{d_i(t_{cur})-d_1(t_{cur})}\right)$
8: $p = p + q_i$
9: **end for**
10: $S(t_{cur}, d_1(t_{cur})) = d_1(t_{cur}) - t_{cur} - p$

$I_i^{min}(T_{amb})$ between the execution of each sub-task, we can schedule all tasks without violating thermal and timing constraints. However, such *static* idle time allocation under pessimistic assumptions cannot efficiently utilize all available slack resources at runtime, which may, in turn, incur unnecessary preemption overheads. Therefore, we develop an online idle-time scheduling algorithm that reclaims unused resources and utilize them to allocate *dynamic* idle-time for each task in an efficient way. As a result, our algorithm can fulfill both thermal and timing requirements with much fewer preemptions.

We here present our online idle-time scheduling algorithm. The scheduler is invoked upon (i) release of a new job (`JOB_RELEASE`), (ii) completion of a job (`JOB_COMPLETION`), or (iii) update of frequency by APAF (`FREQ_UPDATE`). The scheduler keeps track of the worst-case remaining execution time, $e\_left_i(f)$ for the active job of τ_i. This is set to $e_i(f)$ on `JOB_RELEASE`, decremented as the job executes, updated according to frequency change on `FREQ_UPDATE`, and set to 0 on `JOB_COMPLETION`. Upon each invocation (either `JOB_RELEASE`, `JOB_COMPLETION`, or `FREQ_UPDATE`), the scheduler updates available slack $S(t_{cur}, d_1(t_{cur}))$ for the interval of $[t_{cur}, d_1(t_{cur}))$, where t_{cur} is the current time instant and $d_1(t_{cur})$ is the earliest absolute deadline among all released jobs whose deadline is after t_{cur}. Then, the scheduler assigns slack $S(t_{cur}, d_1(t_{cur}))$ to tasks *in proportion to* their average power dissipation (i.e., $P_i \cdot \frac{e_i(f)}{p_i}$). The rationale for such a proportional slack distribution is that a task with higher power dissipation requires more idle-time. In this way, each task is assigned an amount of idle-time equal to

$$I_i(t_{cur}) = I_i^{min}(T_{amb}) + S(t_{cur}, d_1(t_{cur})) \cdot \frac{P_i \cdot \frac{e_i(f)}{p_i}}{\sum_{\tau_i} P_i \cdot \frac{e_i(f)}{p_i}}. \quad (25)$$

Based on the assigned idle time $I_i(t_{cur})$ and the remaining execution time $e\_left_i(f)$, the scheduler splits τ_i into $m_i(I_i(t_{cur}))$ sub-tasks and alternates the processor to be idle for $\frac{I_i(t_{cur})}{m_i(I_i(t_{cur}))}$ units and task execution for $\frac{e\_left_i(f)}{m_i(I_i(t_{cur}))}$ units.

Let's consider how to calculate slack $S(t_{cur}, d_1(t_{cur}))$. Our goal is to find the maximum amount of slack time, which may be available during the interval $[t_{cur}, d_1(t_{cur}))$, while guaranteeing 1) at least the minimum idle-time for all future jobs and 2) all future deadlines ($\geq t_{cur}$) to be met. Algorithm 1 presents our slack calculation method. At time t_{cur}, we look at the interval until the earliest absolute deadline $d_1(t_{cur})$ among all tasks and examine all tasks in reverse EDF order, i.e., latest deadline first (Line 4). Note that tasks are indexed in EDF order (i.e., for τ_i and τ_k where $i < k$, $d_i(t_{cur}) \leq d_k(t_{cur})$). We assume that future task invocations require the worst-case execution and minimum idle- times, and thus their utilization is $\sum_{\tau_i} \frac{e_i(f)+I_i^{min}(T_{amb})}{p_i}$ (Line 1). We try to defer as much execution/idling as possible beyond $d_1(t_{cur})$ and compute the minimum amount of execution/idling p that must execute before $d_1(t_{cur})$ in order to meet all future deadlines (Lines 5–8). This step is repeated for all tasks. To calculate p, we use the similar approach as in [9, 30]. Then, the slack is set to the remaining time slots except for p over the interval $[t_{cur}, d_1(t_{cur}))$ (Line 10). The underlying principle behind our slack calculation is that EDF will determine a feasible schedule if the utilization in Eq. (24) is $\leq$1.0 at any time [5].

Runtime complexity. At each invocation (either `JOB_RELEASE`, `JOB_COMPLETION`, or `FREQ_UPDATE`), our scheduling algorithm updates the slack by Algorithm 1 with the complexity of $O(n)$, where n is the number of tasks. Then, our algorithm allocates the slack to a job with the earliest deadline according to Eq. (25) with the complexity of $O(1)$. Thus, the complexity of our online scheduling algorithm is $O(n)$.

Table 2: Thermal parameters of i.MX6 processor [12, 13]

R (°C/W)	C (J/°C)	β_1 (mA/°C)	β_0 (mA)	P_{max}(mW)
22	0.0454	0.435	611	3860

Table 3: Task execution time and min/maximum periods

(s)	Angle	Bit	Table	Edge	FFT	PID
e_i	2.51	1.03	0.919	0.872	0.456	0.151
p_i^{min}, p_i^{max}	15, 30	6, 12	6, 12	5, 10	2.5, 5	1, 2

7 EVALUATION

We have implemented and evaluated RT-TRM on an automotive and infotainment app processor. Our evaluation focuses on how it guarantees thermal and real-time constraints.

Experimental setup. Our evaluation platform is i.MX6 [13] with ARM A9 supporting 3 discrete frequency levels (1GHz, 0.8GHz, 0.4GHz) and the corresponding voltage levels (1.25V, 1.15V, 0.95V). The chip is equipped with an on-chip thermal sensor with precision of 0.4°C. Table 2 specifies the power and thermal parameters of our target platform. We set the peak temperature constraint T_{max} to 60°C. For the purpose of demonstration, we used realistic automotive workloads obtained from MiBench [17], including *Angle-time Conversion, Bit Manipulation, Table Lookup, Edge Detection, FFT, PID*. The configuration of each workload are shown in Table 3.

We implemented RT-TRM on Linux using Resource Kernel [29] for real-time scheduling and period adaptation. We also leverage GNU LPsolver [28] for linear optimization. For idle-time scheduling, we generate a kernel idle thread to preempt task execution.

Handling ambient temperature variation. To illustrate how RT-TRM adapts to various environmental conditions to meet the

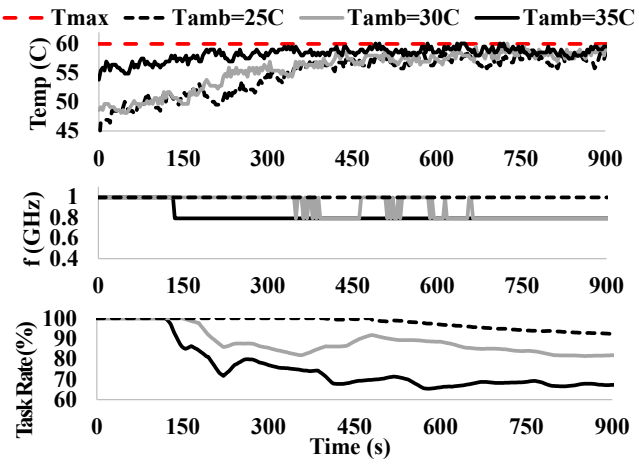


Figure 7: Experimental results of RT-TRM showing processor temperature, frequency, task-rate traces under different ambient temperatures (25/30/35°C).

thermal constraint, we conducted a set of experiments under ambient temperature 25, 30 and 35°C. Fig. 7 plots the real-time traces of processor temperature, frequency and task-rate. The task-rate is defined in Eq. (14) and normalized by the maximum rate. The results show that RT-TRM effectively regulates the processor temperature below T_{max}. Under 25°C (dotted line) RT-TRM is shown to be able to maintain the 1GHz processor frequency and 91.5% of the maximum task-rate. Under 35°C (solid line) , the processor frequency had to be reduced at time around 150s to meet the thermal constraint, resulting in 65.6% of the maximum task-rate. At ambient temperature 30°C (grey line), the processor frequency is switched between 1 and 0.8GHz dynamically adjusting task-rate to achieve 82.6% of the maximum task-rate. RT-TRM used a combination of adaptive parameter assignment and online idle-time scheduling to maintain the processor temperature below the specified limit.

Effect of online slack usage. After demonstrating RT-TRM's ability to meet the thermal constraint under the dynamically changing ambient temperature, we analyze the effect of slack usage on online idle-time scheduling. During the above-mentioned experiment, we measure the total idle-time and the number of preemptions per job, and they are shown in Table 4. Our online idle-time scheduling algorithm can assign more idle-time by 0.054s by efficiently utilizing runtime slack and, as a result, reduce the number of preemptions by 7.4x, compared to the static minimum idle-time allocation method. We observe that a small amount of additional idle-time can dramatically reduce the number of preemptions. By reclaiming the available slack at runtime, RT-TRM uses 24.4% more idle-time to reduce 86.5% of preemptions without violating both thermal and timing constraints.

Performance evaluation. We have thus far demonstrated how RT-TRM to handle the dynamically changing ambient temperature and use runtime slack to reduce the number of preemptions while satisfying thermal and timing constraints. We now focus on resource-efficiency and compare RT-TRM with two baseline approaches:

Table 4: Average number of preemptions and idle-time per job

	Preemption	Used idle-time (s)
Static minimum idle-time	8.77	0.221
Online idle-time scheduling	1.18	0.275

- EDF: static parameter assignment under EDF
- RT-MTC : dynamic processor frequency assignment using feedback control under EDF [16]
- RT-TRM: adaptive parameter assignment (§5) and online idle-time scheduling (§6)

Under EDF, we consider two static parameter assignments[8]: one assumes the average ambient temperature of 25°C (EDF-A), and the other assumes the worst-case ambient temperature of 35°C (EDF-W). Under RT-MTC, if the processor utilization exceeds the schedulable utilization by lowering the processor frequency, task periods are scaled to meet job/task deadlines. We use two metrics: (1) the percentage of time during the thermal constraint is violated and (2) the task-rate. Higher task-rate indicates higher resource-efficiency.

Fig. 8 compares the processor temperature, frequency and task-rate for three different thermal management schemes. EDF-A assigned the processor frequency of 1GHz and the task-rate of 100% whereas EDF-W assigned the frequency of 0.8GHz and the task-rate of 63.5% (Fig. 8a). Under EDF-A, the maximum temperature was 71.5°C, and thus the thermal constraint was violated for 76.7% of the time. Under EDF-W, on the other hand, the thermal constraint was satisfied for all the time with the maximum temperature of 59.1°C, but resources are severely under-utilized.

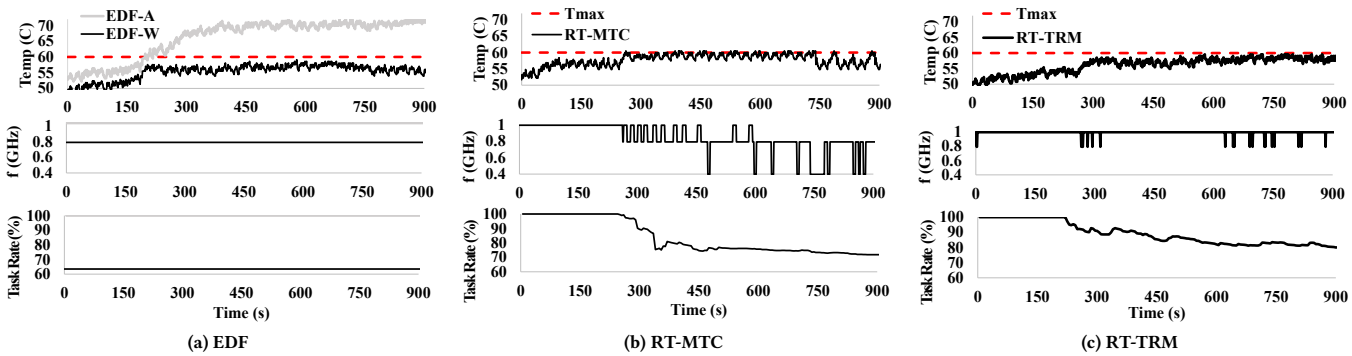
Under RT-MTC in Fig. 8b, when the processor temperature hit the threshold at time 250s, the processor frequency was lowered to 0.8GHz. The temperature still exceeded the limit, so the frequency was lowered again to 0.4GHz at time 750s. Due to the reduced frequency to the lowest level, the task-rate for RT-MTC is reduced to 67.2%. While the feedback control regulates the temperature close to the set point, it violates the thermal constraint for 3% of the time with the maximum temperature of 60.5 °C.

Fig. 8c shows that RT-TRM maintained the maximum processor frequency for most of the time by adaptively adjusting the task periods, achieving the task-rate of 79.4% — an 18.2% improvement over RT-MTC. By efficiently scheduling idle-time, RT-TRM could meet the thermal constraint for all the time with the maximum temperature of 59.6 °C.

8 CONCLUSION

Emerging embedded real-time systems, such as connected cars and smartphones, pose new challenges in meeting the timing constraints under the processors' thermal constraints. Such a system should consider a new *dynamic computation power bound* in addition to the conventional schedulable utilization bound. To address this problem, we have developed a new thermal model that captures individual tasks' heat generations as their *activity factors*. We

[8]Parameters are assigned by solving the optimization problem Eq. (14)



(a) EDF (b) RT-MTC (c) RT-TRM

Figure 8: Experimental results of different schemes showing the processor temperature, frequency, and task-rate traces

then develop two new scheduling mechanisms, *adaptive parameter assignment* and *online idle-time scheduling*. By tightly coupling the solutions of these two scheduling mechanisms, we can guarantee both thermal and timing constraints under dynamic ambient temperature variations. Our evaluation of RT-TRM on a realistic microcontroller using automotive benchmarks has demonstrated the validity of the proposed thermal model and effectiveness of RT-TRM in meeting both real-time and thermal constraints.

REFERENCES

[1] Rehan Ahmed, Parameswaran Ramanathan, and Kewal K Saluja. 2013. On thermal utilization of periodic task sets in uni-processor systems. In *RTCSA*.

[2] Rehan Ahmed, Parameswaran Ramanathan, and Kewal K Saluja. 2014. Necessary and sufficient conditions for thermal schedulability of periodic real-time tasks. In *ECRTS*.

[3] Rehan Ahmed, Parameswaran Ramanathan, and Kewal K Saluja. 2014. Temperature minimization using power redistribution in embedded systems. In *VLSI Design*.

[4] Theodore L Bergman. 2011. *Introduction to heat transfer*. John Wiley & Sons.

[5] S. A. Brandt, S. Banachowski, C. Lin, and T. Bisson. 2003. Dynamic integrated scheduling of hard real-time, soft real-time and non-real-time processes. In *RTSS*.

[6] Thidapat Chantem, X Sharon Hu, and Robert P Dick. 2011. Temperature-aware scheduling and assignment for hard real-time applications on MPSoCs. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* (2011).

[7] Jian-Jia Chen, Chia-Mei Hung, and Tei-Wei Kuo. 2007. On the minimization fo the instantaneous temperature for periodic real-time tasks. In *RTAS*.

[8] Jian-Jia Chen, Shengquan Wang, and Lothar Thiele. 2009. Proactive speed scheduling for real-time tasks under thermal constraints. In *RTAS*.

[9] Hoon Sung Chwa, Kang G. Shin, Hyeongboo Baek, and Jinkyu Lee. 2018. Physical-state-aware dynamic slack management for mixed-criticality systems. In *RTAS*.

[10] Tommaso Cucinotta, Luigi Palopoli, Luca Abeni, Dario Faggioli, and Giuseppe Lipari. 2010. On the integration of application level and resource level QoS control for real-time applications. *IEEE Transactions on Industrial Informatics* (2010).

[11] Nathan Fisher, Jian-Jia Jia Chen, Shengquan Wang, and Lothar Thiele. 2009. Thermal-Aware Global Real-Time Scheduling on Multicore Systems. In *RTAS*.

[12] Freescale. i.MX 6Dual/6Quad Power Consumption Measurement: Table 1. VDDARM, VDDSOC, VDDPU Voltage Levels. https://cache.freescale.com/files/32bit/doc/app_note/AN4509.pdf

[13] Freescale. Technical Data Applications Processors: Table 5. FCPBGA Package Thermal Resistance.

[14] Freescale. Thermal Analysis of Semiconductor Systems: Figure 9. Transient thermal response curve. https://cache.freescale.com/files/analog/doc/white_paper/BasicThermalWP.pdf

[15] Yong Fu, Nicholas Kottenstette, Yingming Chen, Chenyang Lu, Xenofon D. Koutsoukos, and Hongan Wang. 2010. Feedback Thermal Control for Real-time Systems. In *RTAS*.

[16] Yong Fu, Nicholas Kottenstette, Chenyang Lu, and Xenofon D Koutsoukos. 2012. Feedback thermal control of real-time systems on multicore processors. In *EMSOFT*.

[17] Matthew R Guthaus, Jeffrey S Ringenberg, Dan Ernst, Todd M Austin, Trevor Mudge, and Richard B Brown. 2001. MiBench : A free, commercially representative embedded benchmark suite. In *Workshop on Workload Characterization*.

[18] Huang Huang, Gang Quan, Jeffery Fan, and Meikang Qiu. 2011. Throughput Maximization for Periodic Real-Time Systems Under the Maximal Temperature Constraint. In *DAC*.

[19] Ramkumar Jayaseelan and Tulika Mitra. 2008. Temperature aware task sequencing and voltage scaling. In *ICCAD*.

[20] R. Wayne Johnson, John L. Evans, Peter Jacobsen, James R. Thompson, and Mark Christopher. 2004. The changing automotive environment: High-temperature electronics. *IEEE Transactions on Electronics Packaging Manufacturing* 27 (2004).

[21] Joonho Kong, Sung Woo Chung, and Kevin Skadron. 2012. Recent thermal management techniques for microprocessors. *ACM Computing Surveys (CSUR)* 44, 3 (2012), 13.

[22] Pratyush Kumar and Lothar Thiele. 2011. Cool shapers: Shaping real-time tasks for improved thermal guarantees. In *DAC*.

[23] Pratyush Kumar and Lothar Thiele. 2011. System-level power and timing variability characterization to compute thermal guarantees. In *CODES+ISSS*.

[24] C. L. Liu and J. W. Layland. 1973. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the Association for Computing Machinery* 20, 1 (1973), 46–61.

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

[26] Yue Ma, Thidapat Chantem, X Sharon Hu, Robert P Dick, and Notre Dame. 2015. Improving Lifetime of Multicore Soft Real-Time Systems through Global Utilization Control. In *GLVLSI*.

[27] Nimrod Megiddo. 1984. Linear Programming in Linear Time When the Dimension Is Fixed. *J. ACM* (1984).

[28] Peter Notebaert Michel Berkelaar, Kjell Eikland. Open source (Mixed-Integer) Linear Programming sys. http://web.mit.edu/lpsolve/doc/

[29] Shuichi Oikawa and Raj Rajkumar. 1998. Linux RK: A Portable Resource Kernel in Linux. In *RTSS*.

[30] Padmanabhan Pillai and Kang Shin. 2001. Real-time dynamic voltage scaling for low-power embedded operating systems. In *SOSP*.

[31] Lars Schor, Iuliana Bacivarov, Hoeseok Yang, and Lothar Thiele. 2012. Worst-case temperature guarantees for real-time applications on multi-core systems. In *RTAS*.

[32] Danbing Seto, John P Lehoczky, Lui Sha, and Kang G Shin. 2001. Trade-off analysis of real-time control performance and schedulability. *Real-Time Systems* 21, 3 (2001), 199–217.

[33] Shengquan Wang and Riccardo Bettati. 2006. Delay analysis in temperature-constrained hard real-time systems with general task arrivals. In *RTSS*.

[34] Sushu Zhang and KS Chatha. 2010. Thermal aware task sequencing on embedded processors. In *DAC*.