CISTER
Research Centre in
Real-Time & Embedded
Computing Systems

Conference Paper

Thermal-Aware Schedulability Analysis for Fixed-Priority Non-Preemptive Real-Time Systems

Javier Pérez Rodríguez
Patrick Meumeu Yomsi

CISTER-TR-190903

2019/12/03

Thermal-Aware Schedulability Analysis for Fixed-Priority Non-Preemptive Real-Time Systems

Javier Pérez Rodríguez, Patrick Meumeu Yomsi

CISTER Research Centre

Polytechnic Institute of Porto (ISEP P.Porto)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8321159

E-mail: perez@isep.ipp.pt, pamyo@isep.ipp.pt

https://www.cister-labs.pt

Abstract

Technology advances in microprocessor design have resulted in high device density and performance during the last decades. More components are fabricated on the chip die and millions, if not billions, of instructions can now be executed within microseconds. A consequence of this advancement is heat dissipation by the microprocessors. In this context, elevated on-chip temperature issues have become an important subject for the design of future generations of microprocessors, especially in avionics and automotive industries. In this paper, we address the scheduling problem of non-preemptive periodic tasks on a single processor platform under thermal-aware design. We assume that the tasks are scheduled by following any Fixed-Task-Priority (FTP) scheduler (e.g., Rate Monotonic (RM) or Deadline Monotonic (DM)) and we propose a unique framework wherein we capture both the temporal and thermal behavior of the system. Then, we present two new thermal-aware scheduling strategies, referred to as NP-HBC and NP-CBH, to keep the system temperature within specified parameters and we derive their respective schedulability analysis. Finally, we evaluate the performance of the proposed theoretical results through intensive simulations.

Thermal-Aware Schedulability Analysis for Fixed-Priority Non-Preemptive Real-Time Systems

Javier Pérez Rodríguez and Patrick Meumeu Yomsi
CISTER Research Centre, ISEP, Polytechnic Institute of Porto, Portugal
Email: {perez; pmy}@isep.ipp.pt

***Abstract*—Technology advances in microprocessor design have resulted in high device density and performance during the last decades. More components are fabricated on the chip die and millions, if not billions, of instructions can now be executed within microseconds. A consequence of this advancement is heat dissipation by the microprocessors. In this context, elevated on-chip temperature issues have become an important subject for the design of future generations of microprocessors, especially in avionics and automotive industries.**

In this paper, we address the scheduling problem of non preemptive periodic tasks on a single processor platform under thermal-aware design. We assume that the tasks are scheduled by following any Fixed-Task-Priority (FTP) scheduler (e.g., Rate Monotonic (RM) or Deadline Monotonic (DM)) and we propose a unique framework wherein we capture both the temporal and thermal behavior of the system. Then, we present two new thermal-aware scheduling strategies, referred to as NP-HBC **and** NP-CBH**, to keep the system temperature within specified parameters and we derive their respective schedulability analysis. Finally, we evaluate the performance of the proposed theoretical results through intensive simulations.**

I. Introduction

In recent years, techniques to manage heat in order to control processor dissipation in critical real-time systems have been gaining much attention by experts from academia and industry. As a matter of fact, high temperatures due to excessive processor activity while executing heavy workloads expose the platform to transient performance degradation or even a stall of the entire system. Hence, meeting the timing requirements is not the only constraint system designers have to be content with; it is also important to ensure an efficient management of the thermal behavior of the platform.

Over the years, the periodic constrained-deadline task model has proven remarkably useful for the modeling of recurring processes that occur in critical real-time systems, where the failure to satisfy any constraint may entail disastrous consequences. The problem of scheduling such tasks upon a single processor platform so that all the deadlines are met has been widely studied in the literature. An entire body of knowledge, techniques and intuitions have been developed so far. Typically, in order to provide timing guaranteed services for critical real-time tasks, numerous schedulability analyses have been proposed and roughly speaking these analyses can be categorized in two classes, characterized as follows. The first class consists of techniques focusing on deriving a mathematical condition at design time such that, if the condition is satisfied then the system is asserted schedulable, i.e., all the task deadlines will always be met at run-time. The second class consists in simulating the execution of the tasks until a time-instant t such that, if no deadline is missed while scheduling only the tasks released within the time interval $[0, t)$ then no deadline will ever be missed during run-time. In the literature, such time intervals $[0, t)$ are often referred to as *feasibility intervals*.

Obviously, traditional schedulability analyses would need to be revisited and revised before they can apply under thermal-aware design. For example, when considering fixed priority preemptive tasks, while it is possible to interrupt the running task prior to completion to either (i) allocate the processor to incomming tasks requiring urgent service; or (ii) cool-down the processor because the temperature is critically high, the picture darkens considerably when it comes to non-preemptive tasks. Here, just because the running task cannot be interrupted whatsoever at runtime, we would need to anticipate the cooling periods at design time as a function of the processor activity. Then, we would focus only on jobs released in a so-called *busy period* (before and after that the processor is idle) as the behaviors of jobs in different busy periods do not affect each other [1]. However, under thermal-aware design, it becomes difficult to separate the execution of jobs in a busy period from the interference by the execution of jobs in an earlier busy period because new jobs may be released in a time window when the processor is cooling-down. In this contribution, we address this issue and circumvent the hurdle in a rather elegant manner.

▷ **On considering a non-preemptive scheme.** A disadvantage of a non-preemptive scheme is that it introduces additional blocking time in the execution of higher priority tasks, so reducing schedulability. However, such a scheme exhibits several advantages in comparison to a preemptive model of execution. In particular, it allows us (1) to avoid unpredictable interference among tasks at runtime; (2) to achieve a higher degree of predictability; and finally (3) the following (non-exhaustive) issues are avoided.

- In many practical situations, either preemption is impossible or prohibitively expensive.
- Preemption destroys program locality, increasing runtime overhead due to cache misses and pre-fetch mechanisms. Consequently, worst-case execution times (WCETs) are more difficult to characterize and predict.
- Mutual exclusion is trivial in non-preemptive scheduling,

which naturally guarantees the exclusive access to shared resources. On the contrary, to avoid unbounded priority inversion, preemptive scheduling requires the implementation of specific concurrency control protocols, which introduces additional overhead and complexity.

For these reasons, we opted for disabling preemption completely to nullify all the architectural related costs (e.g., cache, pipeline) and execution overheads (e.g., load and restore operations) associated to the occurrence of each preemption.

▷ **This research.** This paper proposes two thermal-aware schedulers: (1) a *reactive* scheduler (NP-HBC); and (2) a *proactive* scheduler (NP-CBH) together with their associated schedulability analysis. The basic idea is to introduce cooling periods during run-time for keeping the processor temperature within specified parameters. In a nutshell, NP-HBC cools down the processor *after* executing some workload; and NP-CBH cools down the processor *prior* to executing the corresponding workload. To the best of our knowledge, this paper is the first contribution to address the thermal-aware schedulability analysis problem of non-preemptive real-time tasks and to allow capturing in the same framework both the thermal and temporal behavior of the system. Furthermore, it provides significant insights, intuitions and techniques on the problem and we strongly believe that it opens an avenue for future research.

▷ **Paper organization.** The remainder of the paper is organized as follow. We present the adopted model of execution in Section II. Section III recapitulates the required state-of-the-art basics for the preliminary results presented in Section IV. Our main contribution is reported in Section V where we specify our thermal-aware schedulers (NP-HBC and NP-CBH). Section VII reviews existing related works and we conclude the paper in Section VIII.

II. Model of computation

In this section, we define the task, platform, scheduler and thermal models assumed throughout this paper. Also, we introduce most of the notations and parameters necessary for a good understanding of the proposed approach.

▷ **Task and platform specifications.** We consider a set of n recurring *independent*[1] tasks $\tau \stackrel{\text{def}}{=} \{\tau_1, \tau_2, \ldots, \tau_n\}$ to be executed on a single processor platform. Every τ_i is modeled by a *constrained-deadline periodic* task characterized by a 4-tuple (O_i, C_i, D_i, T_i), where O_i is the offset, C_i is the worst-case execution time (WCET), $D_i \leq T_i$ is the relative deadline and T_i is the *exact* inter-arrival time between two consecutive releases of task τ_i. These parameters are given with the interpretation that during the execution of the system, task τ_i generates a (potentially infinite) number of successive jobs $\tau_{i,k}$ (with $k = 1, \ldots, \infty$) released at time $r_{i,k}$ such that $r_{i,k+1} - r_{i,k} = T_i, \forall k$ (with $r_{i,1} \stackrel{\text{def}}{=} O_i$, the release time of the first job of τ_i). Without any loss of generality, we assume that $O_i \geq 0, \forall i \in [1, n]$, and we denote by $O_{\max}$ the maximal value among all task offsets, i.e., $O_{\max} \stackrel{\text{def}}{=} \max_{i \in [1,n]}\{O_i\}$. Each job has an execution requirement of at most C_i and must complete within the time window $[r_{i,k}, d_{i,k})$, where $d_{i,k} \stackrel{\text{def}}{=} r_{i,k} + D_i$. Job $\tau_{i,k}$ is said to be *active* at time $t > 0$ if and only if $r_{i,k} \leq t$ and it is not completed yet. More precisely, an active task is said to be *running* at time t if it is being executed by the processor. Otherwise the active task is in the ready queue of the operating system and it is said to be *ready*. We denote by $active(\tau; t)$; $run(\tau; t)$ and $ready(\tau; t)$ the subsets of active, running and ready tasks of τ at time t, respectively. It holds that $active(\tau; t) \stackrel{\text{def}}{=} run(\tau; t) \cup ready(\tau; t)$. We denote by $H \stackrel{\text{def}}{=} lcm\{T_1, T_2, \ldots, T_n\}$ the hyper-period of the task set, defined as the least common multiple of all tasks periods. If τ is synchronous (i.e., $O_i = O_j$, $\forall i, j \in [1, n]$), then it has been proven in [2] that we can consider $O_i = 0$, $\forall i$ and $[0, H)$ is a feasibility interval. Otherwise, if τ is not synchronous (i.e., it is asynchronous with the meaning that $\exists i, j \in [1, n]$ with $i \neq j$ and $O_i \neq O_j$), then $[0, O_{\max} + 2H)$ is a feasibility interval. Note that a tight feasibility interval can be derived by using the techniques presented in [3].

▷ **Scheduler specifications.** We consider that the tasks are executed in a *non-preemptive* manner[2], by following a Fixed-Task-Priority (FTP) scheduler. That is, every task τ_i is assigned a constant priority at system design-time and, at run-time, every released job inherits the priority of the task it belongs to. We denote by $\mathrm{hp}(\tau_i)$ (resp., $\mathrm{lp}(\tau_i)$) the set of tasks with a higher (resp., lower) priority than τ_i and by $\mathrm{hep}(\tau_i)$ (resp., $\mathrm{lep}(\tau_i)$) the set $\mathrm{hp}(\tau_i) \cup \{\tau_i\}$ (resp., $\mathrm{lp}(\tau_i) \cup \{\tau_i\}$). At any point in time, if two jobs are ready and compete for execution, it is the job coming from the task with the highest priority that will be executed first (since the task priorities are passed on to the jobs). This implicitly assumes that all the tasks have distinct priorities and there is at most one job per task that is ready at any point in time. The latter is guaranteed since $D_i \leq T_i, \forall i$.

▷ **Thermal model specifications.** The thermal model of the platform adopted in this paper is similar to the one described in [4], where an RC circuit is used [5], [6]. Figure 1 illustrates the typical circuit representation adopted. At any time instant, we assume that the processor may be in only one of the following two possible states: (i) *active* (i.e., heating) during which tasks may execute or (ii) *inactive* (i.e., cooling) during which tasks are not allowed to execute. We denote the temperature of the processor at time $t \geq 0$ as $T(t)$. More precisely, we denote the temperature during the heating phase (resp., the cooling phase) as $T_h(t)$ (resp., $T_c(t)$). By using this model, the derivative of the system temperature with respect to time can be calculated using Fourier's law [7] by solving the classical linear differential in Equation 1, where parameters a and b are processor specific constants. Note that typical settings for these variables are $b \approx 0.228$, and $a > 1$ (See [8] for further

[1] That is, there is no communication, no precedence constraints and no shared resource (except the processor) between tasks.

[2] This means that once a job has started its execution, it cannot be interrupted or paused prior to its completion time.

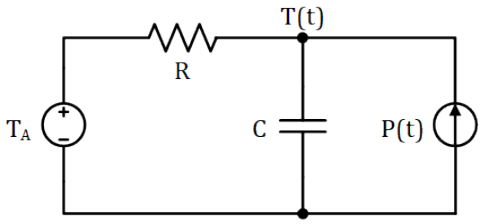


Fig. 1: Circuit representation of the thermal model.

details).

$$T'(t) + b \cdot T(t) = a \tag{1}$$

⋈ **Heating Model:** In this work, we assume that heating comes mainly from the processor activity and that the heating produced by the other components has a negligible impact on the processor global thermal behavior. The heating is modeled by the current $P(t)$ passing through the thermal resistance R and the thermal capacitance C. Also, we assume for clarity sake that the processor speed is $s_\pi = 1$. When the tasks are executing, the temperature must fluctuate between two thresholds: the maximum tolerated temperature $T_{\max}$ and $T_{\min}$, such that $T_{\min} > T_A$ (the ambient temperature)[3]. Hence, the solution to Equation 1, describing the heating function, is given by Equation 2 (see the *red curve* in Figure 2).

$$T_h(t) = \frac{a}{b} + \left(T(t_0) - \frac{a}{b}\right) \cdot e^{-b\cdot(t-t_0)} \tag{2}$$

From Figure 2, note that t_0 represents the time required by the processor to cool-down from $T_{\max}$ to $T_{\min}$ and $T(t_0) = T_{\min}$.
⋈ **Cooling Model:** During this phase, the processor is inactive and we assume that tasks execution is temporarily suspended. For simplicity sake, it is commonly assumed in the literature that this assumption results in $a = 0$ and thus Equation 3 holds to describe the cooling function[4] (see the *blue curve* in Figure 2).

$$T_c(t) = T(t_0) \cdot e^{-b\cdot(t-t_0)} \tag{3}$$

III. Prerequisite

Our thermal-aware solution builds on top of the entire body of knowledge, techniques and intuition developed for the scheduling problem of non-preemptive real-time tasks upon a single processor platform (see for example [9]–[11] among many other publications on the topic). In this section, we recall only the main results we need in the rest of the paper. Specifically, we recall the concept of "level-i busy window" for each task τ_i as well as some extra notations and basic properties.

When considering the scheduling of n tasks $\tau_1, \ldots, \tau_n$ indexed by decreasing order of priority, the level-i busy window for task τ_i is defined as the *longest* time window in which: (1) only jobs from tasks $\tau_1, \ldots, \tau_i$ are executed (except the first job as we will discuss later); and (2) there is no time

[3]The ambient temperature is usually taken from $20°C$ to $25°C$.
[4]Note that this function only depends on time and is not task specific.

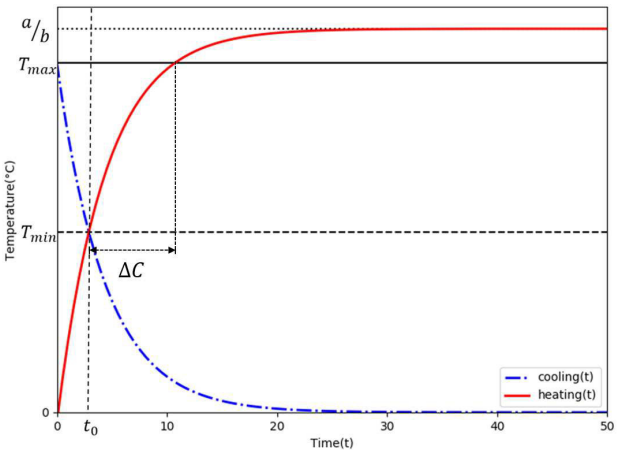


Fig. 2: Cooling and Heating functions.

instant at which a pending ready task is not executed. By using this definition, it has been proven in [9] (see Section 4.3, Lemma 6) that the job-release scenario that leads to the longest level-i busy window is the one where all the tasks $\tau_1, \tau_2, \ldots, \tau_i$ release a job at a same time (say, t) *and* the task $\tau_k \in \text{lp}(\tau_i)$ with the longest C_k releases a job at time $t - \epsilon$, where ϵ is arbitrarily small. That is, we can define a blocking term as follows.

$$B_i \stackrel{\text{def}}{=} \begin{cases} \max\limits_{\tau_k \in \text{lp}(\tau_i)} \{C_k\}; & \text{if } i \in [1, n-1] \\ 0; & \text{otherwise} \end{cases} \tag{4}$$

Starting with an initial length $L_i = B_i$, the length of the level-i busy window is traditionally computed in an iterative manner by adding to L_i the contribution of τ_i and all the higher priority tasks that release jobs during this time window. Formally, L_i is the first fixed-point solution of Equation 5 as proven in [9] (Theorem 15).

$$L_i \stackrel{\text{def}}{=} B_i + \sum_{j \in \text{hep}(\tau_i)} \left(1 + \left\lfloor \frac{L_i}{T_j} \right\rfloor\right) \cdot C_j \tag{5}$$

It is also proven in [9] that the maximum *response time*[5] (also referred to as "WCRT") of any job of τ_i is always observed in the level-i busy window. Thus, it is sufficient to verify that all the deadlines are met within this window to conclude on the schedulability. Once L_i is computed, the number n_i of jobs of τ_i released in the level-i busy window is given by Equation 6.

$$n_i \stackrel{\text{def}}{=} 1 + \left\lfloor \frac{L_i}{T_i} \right\rfloor \tag{6}$$

If we re-index these n_i jobs from 0 to $n_i - 1$, then the latest time at which job $\tau_{i,j}$ starts its execution is given by Equation 7.

$$s_{i,j} \stackrel{\text{def}}{=} B_i + j \cdot C_i + \sum_{k \in \text{hp}(\tau_i)} \left(1 + \left\lfloor \frac{s_{i,j}}{T_k} \right\rfloor\right) \cdot C_k \tag{7}$$

[5]The response time of a job is the time elapsed between its release and completion times.

Hence, the response time of job $\tau_{i,j}$ and the worst-case response time of task τ_i are given by Equation 8.

$$R_{i,j} = s_{i,j} + C_i - j \cdot T_i \text{ and } R_i = \max_{j \in [0, n_i - 1]} \{R_{i,j}\} \quad (8)$$

Although it is safe, note that the number of jobs (see Equation 6) considered in this busy period analysis can be reduced by using the approach presented by Davis et al. (see [12], Equation 10 with no jitters) in order to limit the eventual computational overhead.

IV. Observations and Preliminary Results

Before we discuss the details of our proposed solution, in this section we present a set of important properties for any set of n periodic tasks $\tau_1, \ldots, \tau_n$ (indexed in decreasing order of priority) and a platform π with thermal characteristics $T_{\max}$ and $T_{\min}$.

Property 1. *We have $T_{\max} = T_{\min} \cdot e^{bt_0}$ (from Equation 3) and $t_0 > 0$ (from Figure 2). Consequently, the following inequalities holds true: $T_A < T_{\min} < T_{\max} < a/b$.*

Property 2. *From Property 1, we derive that*

$$t_0 = \frac{1}{b} \cdot \ln\left(\frac{T_{\max}}{T_{\min}}\right) = \frac{1}{b} \cdot \ln\left(1 + \frac{(T_{\max} - T_{\min})}{T_{\min}}\right)$$

Note that variable t_0 defines the longest cooling window that can be performed at once on any processor with the same characteristics as π (see Figure 2).

Definition 1 (ΔC)**.** *Variable ΔC (see Figure 2) is defined as the longest execution time of any task that can safely be performed on π. Formally, ΔC is given by Equation 9.*

$$\Delta C \stackrel{\text{def}}{=} -\frac{1}{b} \cdot \ln\left(\frac{T_{\max} - a/b}{T_{\min} - a/b}\right) = -\frac{1}{b} \cdot \ln\left(1 - \frac{T_{\max} - T_{\min}}{a/b - T_{\min}}\right) \quad (9)$$

Definition 2 (Admissible task τ_i)**.** *Task τ_i is said to be* admissible *on π if and only if its WCET C_i can be executed in isolation without violating any thermal constraint ($T_{\min}$ and/or $T_{\max}$).*

Lemma 1 (Admissible task set)**.** *From Definition 2, task set $\tau = \{\tau_1, \ldots, \tau_n\}$ is admissible if and only if all tasks in τ are admissible, i.e., Equation 10 holds true.*

$$C_{\max} \stackrel{\text{def}}{=} \max_{\tau_i \in \tau} \{C_i\} \leq \Delta C \quad (10)$$

Proof. (By contradiction) Let us assume that there exists $\tau_i \in \tau$ such that $C_i \stackrel{\text{def}}{=} \Delta C + \epsilon$ (for any $\epsilon > 0$) and τ_i is admissible. In the best case, τ_i would execute from instant t_0, when the processor temperature is minimum, and we would have $T_h(t_0 + C_i) \leq T_{\max}$. However, we have:

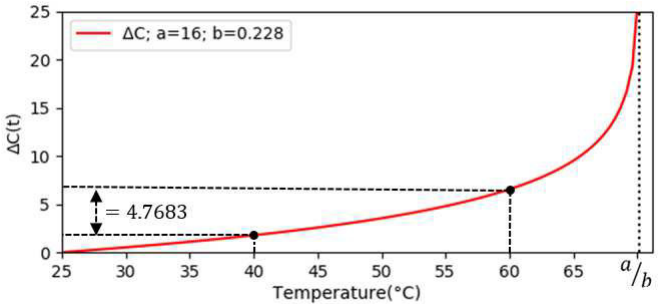


Fig. 3: Trend of admissible execution times.

$$\begin{aligned}
T_h(t_0 + C_i) &= T_h(t_0 + \Delta C + \epsilon) \\
&= \frac{a}{b} + \left(T_{\min} - \frac{a}{b}\right) \cdot e^{-b(t_0 + \Delta C + \epsilon - t_0)} \\
&= \frac{a}{b} + \left(T_{\min} - \frac{a}{b}\right) \cdot e^{-b \cdot \Delta C} \cdot e^{-b \cdot \epsilon} \\
&= \frac{a}{b} + \left(T_{\min} - \frac{a}{b}\right) \cdot e^{-b \cdot \left\{-\frac{1}{b} \cdot \ln\left(\frac{T_{\max} - a/b}{T_{\min} - a/b}\right)\right\}} \cdot e^{-b \cdot \epsilon} \\
&= \frac{a}{b} + \left(T_{\min} - \frac{a}{b}\right) \cdot \left(\frac{T_{\max} - a/b}{T_{\min} - a/b}\right) \cdot e^{-b \cdot \epsilon} \\
&= \frac{a}{b} + (T_{\max} - a/b) \cdot e^{-b \cdot \epsilon} \\
&= \frac{a}{b} + (T_{\max} - a/b) \cdot (e^{-b \cdot \epsilon} - 1) + T_{\max} - a/b \\
&= T_{\max} + (T_{\max} - a/b) \cdot (e^{-b \cdot \epsilon} - 1)
\end{aligned}$$

Now, as $T_{\max} < a/b$ and $e^{-b \cdot \epsilon} < 1$, then it holds that $(T_{\max} - a/b) \cdot (e^{-b \cdot \epsilon} - 1) > 0$. Consequently we have $T_h(t_0 + C_i) > T_{\max}$, which contradicts the claim that τ_i is admissible, i.e., $T_h(t_0 + C_i) \leq T_{\max}$. The Lemma follows. □

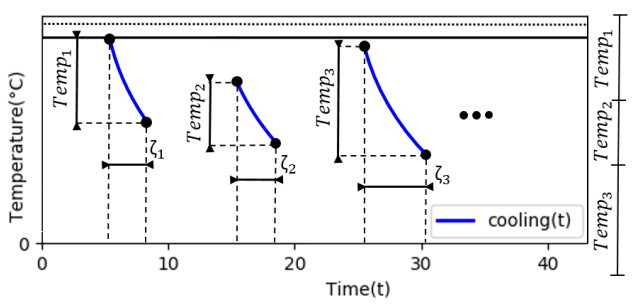
Figure 3 shows the curve of ΔC as a function of the processor temperature. The larger the length of interval $[T_{\min}, T_{\max}]$, the larger the value of admissible execution times. For example, if we consider $T_{\min} = 40°C$ and $T_{\max} = 60°C$, then $\Delta C = \Delta C(T_{\max}) - \Delta C(T_{\min}) = 6.5381 - 1.7698 = 4.7683$ units of time.

Property 3 (Borrowed from [13])**.** *In order not to violate any thermal constraint at run-time, the processor activity follows a Zig-Zag policy, i.e., the processor repeatedly swings between heating and cooling phases (see Figure 5a).*

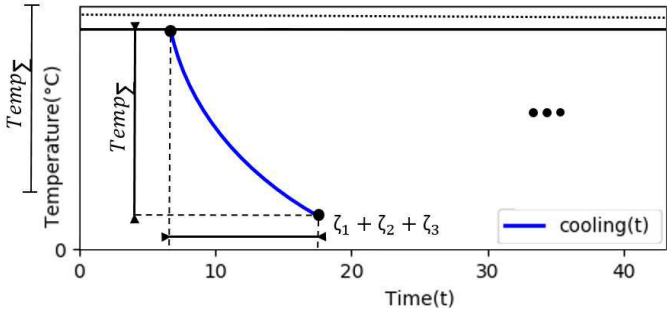
Theorem 1. *Cooling the processor for several short intervals, say ζ_j time units each (with $\zeta_j > 0$), decreases more the processor temperature than only one cooling period of length ($\sum_{j=1}^{k} \zeta_j$) time units (see Figure 4 for an illustration). Formally, for any sequence $\{\zeta_j\}_{1 \leq j \leq k}$ such that $\zeta_j \geq 0$, Equation 11 holds true.*

$$T_c\left(\sum_{j=1}^{k} \zeta_j\right) \leq \sum_{j=1}^{k} T_c(\zeta_j) \quad (11)$$

Proof. The cooling function slows down rapidly. This is because of the exponential function (see Equation 3). Consequently, to be convinced of this claim, it is sufficient to



(a) Successive cooling phases of lengths ζ_j.



(b) Single cooling phase of length $\zeta_1 + \zeta_2 + \zeta_3$.

Fig. 4: Successive cooling phases vs. Single cooling phase.

observe that $T_c\left(\sum_{j=1}^{k}\zeta_j\right) = T(t_0)\cdot e^{-b\cdot\left\{\left(\sum_{j=1}^{k}\zeta_j\right)-t_0\right\}}$ and $T_c(\zeta_j) = T(t_0)\cdot e^{-b\cdot(\zeta_j - t_0)}$. Thus, we have

$$
\begin{aligned}
& T_c\left(\sum_{j=1}^{k}\zeta_j\right) \leq \sum_{j=1}^{k} T_c(\zeta_j) \\
\Longleftarrow\ & T(t_0)\cdot e^{-b\cdot\left\{\left(\sum_{j=1}^{k}\zeta_j\right)-t_0\right\}} \leq \sum_{j=1}^{k}\left\{T(t_0)\cdot e^{-b\cdot(\zeta_j-t_0)}\right\} \\
\Longleftarrow\ & e^{-b\cdot\left\{\left(\sum_{j=1}^{k}\zeta_j\right)-t_0\right\}} \leq \sum_{j=1}^{k} e^{-b\cdot(\zeta_j-t_0)} \\
\Longleftarrow\ & e^{-b\cdot\left(\sum_{j=1}^{k}\zeta_j\right)} \leq \sum_{j=1}^{k} e^{-b\cdot\zeta_j}
\end{aligned}
$$

However, $\sum_{j=1}^{k}\zeta_j \geq \zeta_j$. Consequently, $-b\cdot\left(\sum_{j=1}^{k}\zeta_j\right) \leq -b\cdot\zeta_j$ since $b > 0$. This means:

$$
e^{-b\cdot\left(\sum_{j=1}^{k}\zeta_j\right)} \leq e^{-b\cdot\zeta_j} \leq \sum_{j=1}^{k} e^{-b\cdot\zeta_j}
$$

and thus the theorem follows. □

Property 4. *It is beneficial for the system designer from a thermal viewpoint to perform a cooling between every two consecutive job executions to (i) keep the processor temperature within specified parameters ($T_{\min}$ and $T_{\max}$); (ii) avoid longer cooling periods at once; and (iii) keep the average temperature as low as possible.*

Proof. Points (i) and (ii) follow directly from Property 3 and Theorem 1, respectively. Regarding (iii), we refer the reader to the Appendix at the end of the paper for the insights. □

The strategy defined in Property 4 provides the maximum number of cooling phases by construction. Indeed, for any fixed time window W and $\sigma \geq 2$ jobs, at most $\sigma - 1$ coolings can be interleaved between the execution of these jobs as the tasks are non-preemptive.

Lemma 2 (Shifted heating function – $T_{hs}(t)$ – and Shifted cooling function – $T_{cs}(t)$)**.** *Let $T_h(t)$ be the heating function and $T_c(t)$ the cooling function as defined in Equations 2 and 3. Assuming that τ_i (with $i \in [1, n]$) is the task to be executed from time t_0, but the execution of the blocking term B_i requires the processor to cool-down for, say $x > 0$ units of time, before C_i can be executed and the processor reaches its maximum temperature $T_{\max}$ upon the execution of τ_i (see Figure 5b). Then, the* shifted *cooling function $T_{cs}(t)$ at distance d_c from $T_c(t)$ and the* shifted *heating function $T_{hs}(t)$ at distance d_h from $T_h(t)$ are derived as follows.*

$$
\begin{aligned}
T_{cs}(t) &\stackrel{\text{def}}{=} T_{\min}\cdot e^{-b\cdot(t-t_0+d_c)} \\
&= \left(T_{\min} + \frac{a}{b}\cdot e^{b\cdot B_i} - \frac{a}{b}\right)\cdot e^{-b\cdot(t-t_0)}
\end{aligned}
\tag{12}
$$

and

$$
\begin{aligned}
T_{hs}(t) &\stackrel{\text{def}}{=} \frac{a}{b} + \left(T_{\min} - \frac{a}{b}\right)\cdot e^{-b\cdot(t-t_0+d_h)} \\
&= \frac{a}{b} + \left\{T_{\min} - \frac{a}{b}\cdot\left(e^{b\cdot(B_i+x)} - e^{b\cdot B_i} + 1\right)\right\}\cdot e^{-b\cdot(t-t_0)}
\end{aligned}
\tag{13}
$$

Proof. From Equation 3 and Property 1, we have:

$$
\begin{aligned}
T_{cs}(t) &= T_{\min}\cdot e^{-b\cdot(t-t_0+d_c)} \\
&= T_{\min}\cdot e^{-b\cdot(t-t_0)}\cdot e^{-b\cdot d_c}
\end{aligned}
\tag{14}
$$

From Figure 5a and Property 4, since the processor is required to cool-down for $x > 0$ units of time before C_i can be executed, then at time instant $t_0 + B_i$ we have:

$$
\begin{aligned}
& T_h(t_0+B_i) = T_{cs}(t_0+B_i) \\
i.e.,\ & \frac{a}{b} + \left(T_{\min} - \frac{a}{b}\right)\cdot e^{-b\cdot B_i} = T_{\min}\cdot e^{-b\cdot(B_i+d_c)} \\
i.e.,\ & \frac{\frac{a}{b} + \left(T_{\min} - \frac{a}{b}\right)\cdot e^{-b\cdot B_i}}{T_{\min}\cdot e^{-b\cdot B_i}} = e^{-b\cdot d_c}
\end{aligned}
$$

By substituting this value in Equation 14, we obtain:

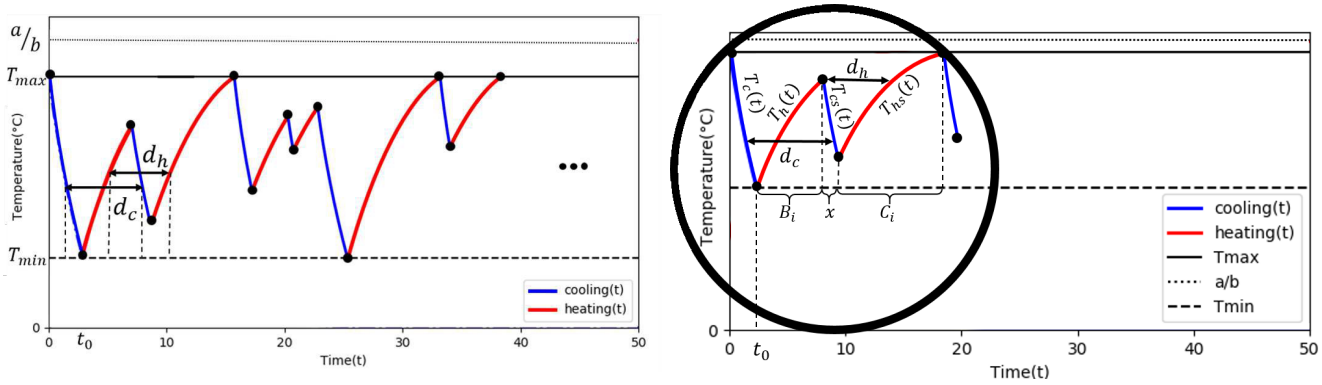
$$
T_{cs}(t) = \left(T_{\min} + \frac{a}{b}\cdot e^{b\cdot B_i} - \frac{a}{b}\right)\cdot e^{-b\cdot(t-t_0)}
$$

Similarly, given function $T_h(t)$, by definition we have:

$$
T_{hs}(t) = \frac{a}{b} + \left(T_{\min} - \frac{a}{b}\right)\cdot e^{-b\cdot(t-t_0+d_h)} \tag{15}
$$

Again, from Property 4 (every cooling phase implies a coming heating phase), at time instant $t_0 + B_i + x$ we have:

$$
\begin{aligned}
& T_{cs}(t_0+B_i+x) = T_{hs}(t_0+B_i+x) \\
i.e.,\ & \left(T_{\min} + \frac{a}{b}\cdot e^{b\cdot B_i} - \frac{a}{b}\right)\cdot e^{-b\cdot(B_i+x)} = \\
& \qquad \frac{a}{b} + \left(T_{\min} - \frac{a}{b}\right)\cdot e^{-b\cdot(B_i+x+d_h)}
\end{aligned}
$$



(a) Zig-Zag behavior.

(b) Zoom of the initial execution phases.

Fig. 5: Zig-Zag behavior and Zoom

By isolating $e^{-b \cdot d_h}$ from this last equation, we obtain:

$$\frac{T_{\min} - \frac{a}{b} \cdot \left(e^{b \cdot (B_i + x)} - e^{b \cdot B_i} + 1\right)}{T_{\min} - \frac{a}{b}} = e^{-b \cdot d_h}$$

Finally, by substituting this value in Equation 15, we obtain:

$$T_{hs}(t) = \frac{a}{b} + \left\{T_{\min} - \frac{a}{b} \cdot \left(e^{b \cdot (B_i + x)} - e^{b \cdot B_i} + 1\right)\right\} \cdot e^{-b \cdot (t - t_0)}$$

The Lemma follows. □

Property 5. *From Property 4, the length of each cooling phase can be associated to the execution of a task, i.e., each C_i can be augmented by C_i^{cool} to have:*

$$B_i^* \stackrel{\text{def}}{=} B_i + B_i^{\text{cool}} \quad \text{and} \quad C_i^* \stackrel{\text{def}}{=} C_i + C_i^{\text{cool}} \tag{16}$$

Lemma 3 (Level-i busy window)**.** *By using the previous observations, the length L_i^* of the level-i busy window ($i \in [1, n]$) considering thermal constraints is the first fixed-point solution of Equation 17.*

$$L_i^* = \left\{ B_i^* + \sum_{j \in \text{hep}(\tau_i)} \left(1 + \left\lfloor \frac{L_i^*}{T_j} \right\rfloor\right) \cdot C_j^* \right\} - C_i^{\text{cool}} \tag{17}$$

where B_i^* and C_i^* are defined by Equation 16.

Proof. The proof of this Lemma is similar to that of the traditional level-i busy window found in the literature. Here, the reader should consider the WCET of each task τ_i as defined by Equation 16. As such, we will not repeat the proof in this paper due to space limitation. Factor C_i^{cool} is subtracted from the RHS of Equation 17 because the execution of the last job of τ_i in L_i^* does not need cooling upon completion (see for example Figure 6a to get convinced). □

Lemma 4 (Number n_i^* of jobs of τ_i)**.** *Once the value of L_i^* is computed, the number n_i^* of jobs of τ_i released in the level-i busy window is given by Equation 18.*

$$n_i^* = 1 + \left\lfloor \frac{L_i^*}{T_i} \right\rfloor \tag{18}$$

Lemma 5 (Latest start time of job $\tau_{i,j}$)**.** *If we re-index the n_i^* jobs of τ_i that are released in the level-i busy window from 0 to $n_i^* - 1$, then the latest time at which job $\tau_{i,j}$ (with $j \in [0, n_i^* - 1]$) starts its execution is given by Equation 19.*

$$s_{i,j}^* = B_i^* + j \cdot C_i^* + \sum_{k \in \text{hp}(\tau_i)} \left(1 + \left\lfloor \frac{s_{i,j}^*}{T_k} \right\rfloor\right) \cdot C_k^* \tag{19}$$

Lemma 6 (Worst-case response time of task τ_i)**.** *The response time of $\tau_{i,j}$ and the worst-case response time of τ_i under thermal-aware design are given by Equation 20.*

$$R_{i,j}^* = s_{i,j}^* + C_i - j \cdot T_i \quad \text{and} \quad R_i^* = \max_{j \in [0, n_i^* - 1]} \{R_{i,j}^*\} \tag{20}$$

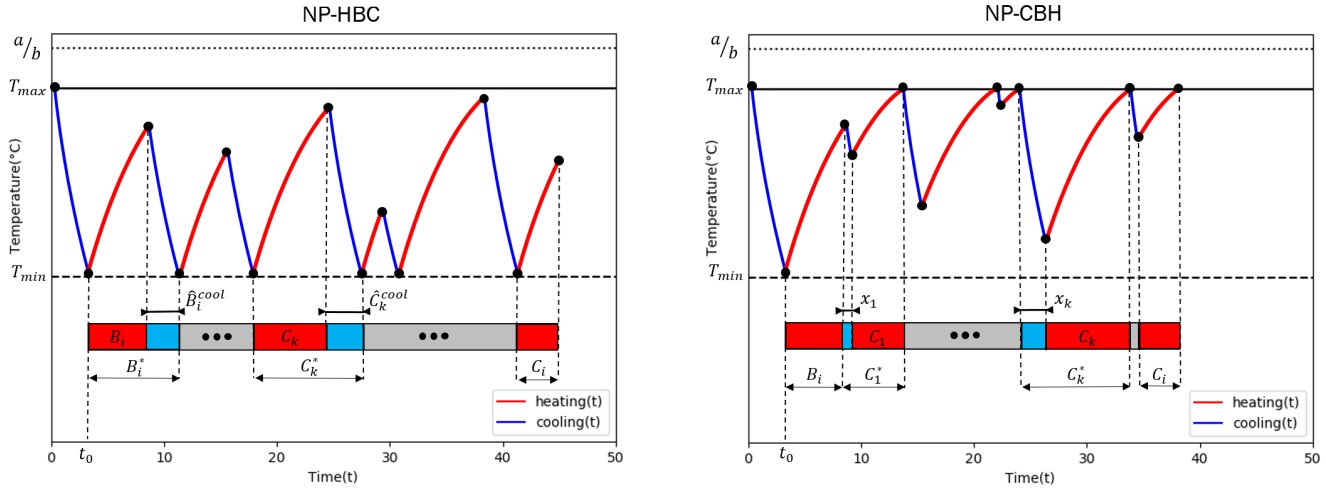
Lemma 7 (Worst-Case Scenario)**.** *The job-release scenario that leads to the longest level-i busy window is the one where:*

1) *the platform temperature is initially at the maximum level $T_{\max}$ (say, at time 0) and it has to cool down to the minimum level $T_{\min}$ (say, at time t_0), before any possible task execution;*
2) *tasks $\tau_1, \tau_2, \ldots, \tau_i$ release a job at the same time (say, at t_0);*
3) *task $\tau_k \in \text{lp}(\tau_i)$ with the longest C_k releases a job at $t_0 - \epsilon$.*

Similar to the proof of Lemma 3, the proofs for Lemmas 4 to 7 also follow the same intuitions as the corresponding ones in the literature. Due to space limitation, we will not report them in this paper. In our case, it is sufficient to mention that the processor is inactive during the cooling phases and these phases behave in a non-preemptive fashion.

V. Our Proposed Solution

With all the insights we presented in Section IV, we now have everything we need to design our reactive thermal-aware scheduler (NP-HBC) and proactive thermal-aware scheduler (NP-CBH) together with their associated schedulability analysis. We recall that the basic idea of these two schemes is to introduce cooling periods during run-time for keeping the processor temperature within specified parameters. The NP-HBC scheduler was designed with the main objective of avoiding high temperatures; whereas NP-CBH was designed with the aim of reducing the worst-case response time for each



(a) Thermal and timing behaviors under NP-HBC.

(b) Thermal and timing behaviors under NP-CBH.

Fig. 6: NP-HBC vs. NP-CBH

task as much as it is legally possible to do so, i.e., without violating any temporal and/or thermal requirement.

A. NP-HBC *scheduler*

From Property 4, the main intuition behind the design of the NP-HBC scheduler is cool-down the processor as much as possible in order to avoid high temperatures, i.e., the scheduler should cool-down the processor to its minimum temperature ($T_{\min}$), upon the execution of each job (see for example Figure 6a). To this end, Theorem 2 and Corollary 1 allow us to compute an upper-bound on the factors B_i^{cool} and C_i^{cool}, respectively.

Theorem 2 (Upper-bound on B_i^{cool})**.** *For any task $\tau_i \in \tau$, an upper-bound $\widehat{B}_i^{\text{cool}}$ on B_i^{cool} is given by Equation 21.*

$$\widehat{B}_i^{\text{cool}} = -\left\{ \frac{1}{b} \cdot \ln \left[\frac{T_{\min}}{\left(T_{\min} + \frac{a}{b} \cdot (e^{b \cdot B_i} - 1)\right)} \right] + B_i \right\} \quad (21)$$

Proof. Let us assume that the processor has executed for B_i units of time during the first heating phase, say from time instant t_0. Then, by following the NP-HBC scheduling scheme, $\widehat{B}_i^{\text{cool}}$ is defined by the earliest time instant at which the cooling function reaches $T_{\min}$. As such, $\widehat{B}_i^{\text{cool}}$ is obtained by solving:

$$T_{cs}(t_0 + B_i + \widehat{B}_i^{\text{cool}}) = T_{\min}$$

$$i.e., \quad \left(T_{\min} + \frac{a}{b} \cdot e^{b \cdot B_i} - \frac{a}{b}\right) \cdot e^{-b \cdot (t_0 + B_i + \widehat{B}_i^{\text{cool}} - t_0)} = T_{\min}$$

$$i.e., \quad e^{-b \cdot (B_i + \widehat{B}_i^{\text{cool}})} = \frac{T_{\min}}{\left(T_{\min} + \frac{a}{b} \cdot e^{b \cdot B_i} - \frac{a}{b}\right)}$$

$$i.e., \quad -b \cdot (B_i + \widehat{B}_i^{\text{cool}}) = \ln \left[\frac{T_{\min}}{\left(T_{\min} + \frac{a}{b} \cdot (e^{b \cdot B_i} - 1)\right)} \right]$$

From this last equation, it follows that

$$\widehat{B}_i^{\text{cool}} = -\left\{ \frac{1}{b} \cdot \ln \left[\frac{T_{\min}}{\left(T_{\min} + \frac{a}{b} \cdot (e^{b \cdot B_i} - 1)\right)} \right] + B_i \right\}$$

and the Theorem follows. □

Corollary 1 (Upper-bound on C_i^{cool})**.** *From Equation 21, it follows that an upper-bound $\widehat{C}_i^{\text{cool}}$ on C_i^{cool} is given by Equation 22.*

$$\widehat{C}_i^{\text{cool}} = -\left\{ \frac{1}{b} \cdot \ln \left[\frac{T_{\min}}{\left(T_{\min} + \frac{a}{b} \cdot (e^{b \cdot C_i} - 1)\right)} \right] + C_i \right\} \quad (22)$$

Proof. Note that $\widehat{B}_i^{\text{cool}}$ depends only on B_i and the processor characteristics, thus by following a similar reasoning as in the proof of Theorem 2, the corollary follows. □

Theorem 3 (Schedulability test under NP-HBC)**.** *A sufficient schedulability test for task $\tau_i \in \tau$ is given by Equation 23.*

$$R_i^* \leq D_i \quad (23)$$

where $B_i^* = B_i + \widehat{B}_i^{\text{cool}}$; $C_i^* = C_i + \widehat{C}_i^{\text{cool}}$; and R_i^* is given by Equation 20.

Proof. This theorem follows directly from the properties and preliminary results in Section IV, Theorem 2 and Corollary 1. □

Pessimism of the NP-HBC **analysis.** At the end of each heating phase, the NP-HBC scheduler commands to cool-down the processor to its minimum temperature level ($T_{\min}$) during each cooling phase. This scheduling strategy, while it limits the processor from continuously executing at high temperatures, it obviously introduces some pessimism in the computation of the worst-case response time of each task. The NP-CBH scheduling strategy offers an alternative solution by addressing this concern, while still meeting both the thermal and temporal constraints for each task.

B. NP-CBH *scheduler*

In contrast to the NP-HBC scheduler, the NP-CBH scheduler is proactive in the sense that it promotes the cooling of the processor prior to the execution of the corresponding task during the next heating phase. In this process, once the maximum temperature level ($T_{\max}$) is reached, the processor is cooled down for just the amount of time that is required to reach the maximum temperature level again at the end of the next heating phase. As such, the cooling phases are reduced as much as it is legally possible to do so, thus benefiting the worst-case response time for each task (see for example Figure 6b). Consequently, tighter response times are obtained under the NP-CBH scheduler, but this is achieved at the expense of requiring the processor to execute most of the tasks at high temperatures at run-time.

Theorem 4 allows us to compute the length of the first cooling phase assuming a scenario as the one considered in Lemma 2, i.e., τ_i is the task to be executed from time t_0, but the execution of the blocking term B_i requires the processor to cool-down for, say $x > 0$ units of time, before C_i can be executed and the processor reaches its maximum temperature $T_{\max}$ upon the execution of τ_i.

Theorem 4 (Computation of x)**.** *Assuming the previously mentioned scenario, the length of the first cooling phase x is obtained by Equation 24.*

$$x = \frac{1}{b} \cdot \ln \left[\frac{T_{\min} + \frac{a}{b} \cdot (e^{b \cdot B_i} - 1)}{T_{\max} + \frac{a}{b} \cdot (e^{-b \cdot C_i} - 1)} \right] - (B_i + C_i) \quad (24)$$

Proof. At the completion time t of task τ_i in the considered scenario; and from Equations 2, 3, 13, and 14, we have $t - t_0 = B_i + x + C_i$ and

$$\frac{a}{b} + \left\{ T_{\min} - \frac{a}{b} \left(e^{b \cdot (B_i + x)} - e^{b \cdot B_i} + 1 \right) \right\} e^{-b \cdot (t - t_0)} = T_{\min} \cdot e^{-b \cdot (t - t_0 - (t_0 + B_i + x + C_i))}$$

$$i.e., \ \frac{a}{b} + \left\{ T_{\min} - \frac{a}{b} \left(e^{b \cdot (B_i + x)} - e^{b \cdot B_i} + 1 \right) \right\} e^{-b \cdot (t - t_0)} = T_{\max} \cdot e^{-b \cdot (t - t_0)} \cdot e^{b \cdot (B_i + x + C_i)}$$

because $T_{\min} \cdot e^{b \cdot t_0} = T_{\max}$ (see Property 1). Thus,

$$\frac{a}{b} + \left\{ T_{\min} - \frac{a}{b} \left(e^{b \cdot (B_i + x)} - e^{b \cdot B_i} + 1 \right) \right\} \cdot e^{-b(t - t_0)} = T_{\max} \cdot e^{-b(t - t_0)} \cdot e^{b \cdot (B_i + x + C_i)}$$

$$i.e., \ \frac{a}{b} \cdot e^{b(t - t_0)} + \left\{ T_{\min} - \frac{a}{b} \left(e^{b \cdot (B_i + x)} - e^{b \cdot B_i} + 1 \right) \right\} = T_{\max} \cdot e^{b \cdot (B_i + x + C_i)}$$

$$i.e., \ \frac{a}{b} \cdot e^{b \cdot (B_i + x + C_i)} + T_{\min} - \frac{a}{b} \left(e^{b \cdot (B_i + x)} - e^{b \cdot B_i} + 1 \right) = T_{\max} \cdot e^{b \cdot (B_i + x + C_i)}$$

Since $e^{b \cdot (B_i + x)} = e^{b \cdot (B_i + x + C_i)} \cdot e^{-b \cdot C_i}$, we can isolate $e^{b \cdot (B_i + x + C_i)}$ to get:

$$e^{b \cdot (B_i + x + C_i)} = \frac{T_{\min} + \frac{a}{b} \cdot (e^{b \cdot B_i} - 1)}{T_{\max} + \frac{a}{b} \cdot (e^{-b \cdot C_i} - 1)}$$

$$i.e., \ x = \frac{1}{b} \cdot \ln \left[\frac{T_{\min} + \frac{a}{b} \cdot (e^{b \cdot B_i} - 1)}{T_{\max} + \frac{a}{b} \cdot (e^{-b \cdot C_i} - 1)} \right] - (B_i + C_i)$$

and the theorem follows. □

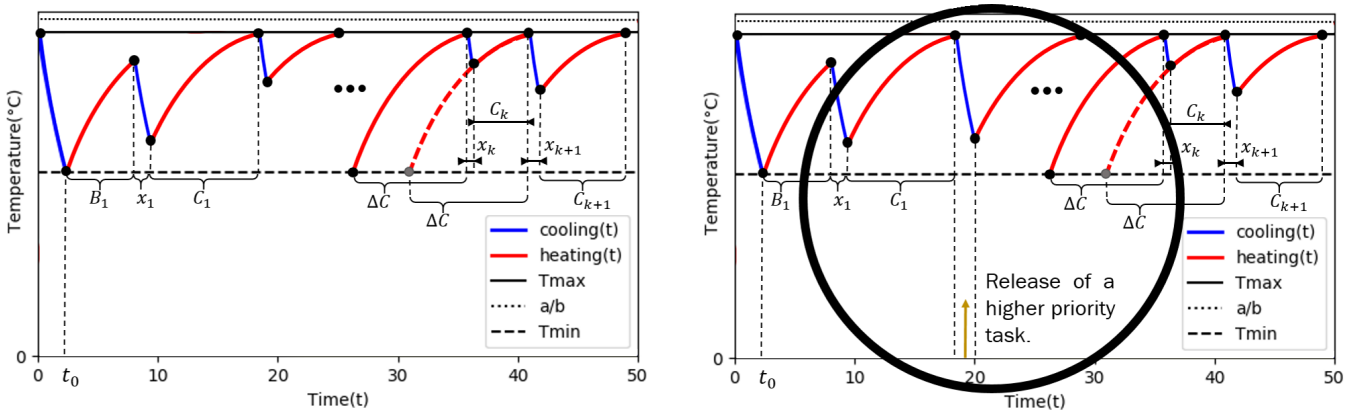
From Equation 24 and Property 4, it is important to remark that the computation of x depends on the processor characteristics and only two other factors: (1) the length of the *previous* heating phase (on the left) *before* τ_i is executed; and (2) the length of the *next* heating phase (on the right), during which τ_i is executed. Corollary 2 provides us with a generic expression for the computation of the length of successive cooling periods x_k (with $k \geq 1$) when the tasks are scheduled by following the NP-CBH scheduler.

Corollary 2 (Computation of x_k with $k \geq 1$)**.** *Assuming that tasks are re-indexed in the ready queue w.r.t. their priority and τ_1 is the first task in the queue, then we have:*

$$x_k = \begin{cases} \frac{1}{b} \cdot \ln \left[\frac{T_{\min} + \frac{a}{b} \cdot (e^{b \cdot B_1} - 1)}{T_{\max} + \frac{a}{b} \cdot (e^{-b \cdot C_1} - 1)} \right] - (B_1 + C_1), & \text{if } \ k = 1 \\ \\ \frac{1}{b} \cdot \ln \left[\frac{T_{\min} + \frac{a}{b} \cdot (e^{b \cdot \Delta C} - 1)}{T_{\max} + \frac{a}{b} \cdot (e^{-b \cdot C_k} - 1)} \right] - (\Delta C + C_k), & \text{if } \ k > 1 \end{cases} \quad (25)$$

Proof. The computation of x_1 is provided by Theorem 4 and the computation of x_k (with $k > 1$) follows directly from the NP-CBH scheduling strategy. It can be proven by induction on k (Figure 7a provides the support for such a proof). Due to space limitation, we will not formalize the proof in this paper. However, the intuition is as follows. Under NP-CBH, we start a heating phase at t_0 (i.e., at the processor minimum temperature level ($T_{\min}$)) and this phase completes at time $t_0 + B_1$. Then, we start a cooling phase for x_1 units of time in order to make the execution of τ_1 possible. Note that x_1 is a function of B_1 (on the left) and C_1 (on the right). Upon the completion of τ_1, we are exactly at the processor maximum temperature level ($T_{\max}$), i.e., at time $t_0 + B_1 + x_1 + C_1$. At this time instant, we start another cooling phase, but just for the amount of time required, say x_j, to hit the processor maximum temperature level ($T_{\max}$) upon the completion of the next task, say τ_j, during the next heating phase. The key insight for the computation of x_j is to consider a blocking equal to ΔC (on the left) since we are at the maximum temperature at time $t_0 + B_1 + x_1 + C_1$; and C_j (on the right). We recall that ΔC defines an upper-bound on the WCET of admissible tasks (see Definition 1 and Lemma 1). □

With everything we have presented so far about the design of the NP-CBH scheduler, we would now be able to "easily" derive a closed-form schedulability test for all tasks. Unfortunately, a piece of the puzzle is still missing. The challenge here stems from the fact NP-CBH, in contrast to NP-HBC, is a proactive scheduler. It computes the length of each cooling phase, say $x_\star$, prior to the execution of the corresponding task, say $\tau_\star$, during the next heating phase. In this cooling phase window, the processor is inactive, but it could perfectly be the case that a task, say $\tau_{\star\star} \in \tau$, with a higher priority than $\tau_\star$, releases a new job (see Figure 7b). In this situation, task $\tau_{\star\star}$ is the one that should take over the processor next and be executed during the next heating phase (instead of $\tau_\star$).



(a) Support for the computation of x_k.

(b) HP task released during a cooling phase.

Fig. 7: Challenge for deriving a closed-form schedulability test under NP-CBH.

Consequently, the ready queue has to be updated after the computation of the length of each cooling phase.

Because of the aforementioned hurdle, we derive the schedulability analysis for each task under the NP-CBH scheduler by using a pseudo-polynomial algorithm. We simulate the system in an interval $[0, t_\eta)$ such that if all the jobs released in this interval meet their deadlines, then we have the guarantee that this will always be the case. For asynchronous periodic tasks, $[0, O_{\max} + 2H)$ is a well-known feasibility interval for the adopted model of execution. However, we recall that tighter feasibility intervals can be derived by using techniques such as the ones presented in [3].

Again, we assume that tasks are re-indexed in the ready queue w.r.t. their priority and τ_1 is initially the first task in that queue. At any time instant t, we refer to the pending ready task with the highest priority as τ_{first} and refer to its WCET as C_{first}. Algorithm 1 provides the pseudo-code of the NP-CBH schedulability test for each task in the feasibility interval $[0, O_{\max} + 2H)$ for sake of simplicity and readability. We recall that $H \stackrel{\text{def}}{=} lcm_{\tau_i \in \tau}\{T_i\}$.

In Algorithm 1, after the initializations (lines 1 to 5), we sort the ready queue w.r.t. task priorities (line 6). For each task in this sorted queue (from the first task), we check if it is admissible (line 8). If this is not the case, we jump to line 40 as the processor thermal constraint is violated (see Lemma 1). Otherwise, if the task is admissible, then we execute as many tasks as possible within the first heating phase window (lines 9 to 20). At the end of the last possible execution, we compute the length of the eventual cooling phase (line 25). We check whether other tasks would release new jobs within this window (line 26) and update the ready queue (line 27). Once the ready queue is stable (unchanged)[6], we compute the length of the actual cooling phase (line 29); we execute τ_{first} (line 30) and update the ready queue (line 31). We repeat this procedure until the ready queue is empty (i.e., the first idle point is reached in the schedule – thus defining the level-i busy window L_i^{**}), or $O_{\max} + 2H$ is exceeded (lines 22 to 32). In the latter case, the system will never stabilize and we have the guarantee of a deadline miss [2] – the system is stamped as not schedulable. Finally, we compute the worst-case response time (WCRT) R_i^{**} for each task τ_i and assess its schedulability (lines 33 to 38).

[6]This will be the case after a number of iterations as the number of tasks is finite.

VI. Experimental results

This section reports on the experiments conducted to validate the presented theoretical results. Our simulations are carried out by generating $20,000$ synthetic periodic implicit-deadline[7] task-sets.

▷ **Task-set generation.** The inputs to the task-set generator are as follows. We consider variables $a = 16$, $b = 0.228$, computed from Skadron et al. [14] for a silicon chip. Then, we use the thermal characteristics of a typical ARM-Cortex-A9 core: $T_{\max} = 65°C$ and $T_{\min} = 30°C$. From these parameters, the maximum admissible execution time ΔC and the value of t_0 are computed by using Definition 1 and Property 2: $\Delta C = 8.9882$ and $t_0 = 3.3911$. Finally, a maximum system utilization $U \stackrel{\text{def}}{=} \sum_{i=1}^{n} \frac{C_i}{T_i} \in [0.1, 1]$ is set. To capture the actual system behavior at run-time, we process the data by generating $1,000$ task-sets per system utilization, from 0.1 to 1, with a step of 0.05.

Given these inputs, the execution time values C_i are uniformly generated within $[\Delta C/m, \Delta C]$, where $m \in \mathbb{N}^*$ is used to dimension the length of the interval. Here, we choose $m = 2$. The tasks' periods are generated in such a way that the hyper-period is kept "reasonably small". Typically, period T_i is computed as $2^{\alpha_i} \cdot 3^{\beta_i} \cdot 5^{\gamma_i}$, where α_i, β_i, and γ_i are randomly chosen in $\{0, 1, 2\}$. In order to accurately evaluate the effect of temperature on the temporal behavior of the system, we choose the periods such that $T_i \geq 3 \cdot \Delta C$, $\forall i$. This is to allow enough slack for the execution of each task. The utilization u_i of τ_i is computed as $u_i = C_i/T_i$. New tasks are generated until the sum of the tasks' utilizations exceeds the threshold

[7]For each task τ_i, we consider $D_i = T_i$.

Algorithm 1: NP-CBH schedulability test.

Data: Task-set $\tau = \{\tau_1, \ldots, \tau_n\}$; a; b; $T_{\max}$; $T_{\min}$.
Result: NP-CBH Schedulability test.

```
ReadyQueue ← ∅;
current ← 0;
Compute t_0 (see Property 2);
Compute ΔC (see Definition 1);
Record all the releases at current;
Update ReadyQueue w.r.t. tasks priorities;
foreach τ_i ∈ ReadyQueue do
    if C_i ≤ ΔC then
        Compute B_i (see Equation 4);
        exec ← B_i;
        avail ← ΔC − exec;
        Check releases during heating phase;
        Update ReadyQueue;
        while C_j ≤ avail and (j < i) do
            exec ← exec + C_j;
            avail ← avail − C_j;
            Remove τ_j from ReadyQueue;
            Check releases during heating phase;
            Update ReadyQueue;
        end
        current ← exec;
        while (current ≤ O_max + 2 · lcm) or
          ReadyQueue == ∅ do
            do
                previousReadyQueue ←
                  ReadyQueue;
                Compute eventual x (see Equation 25);
                Check releases during cooling phase;
                Update ReadyQueue;
            while
              (ReadyQueue ≠ previousReadyQueue);
            Compute actual x (see Equation 25);
            current ← current + x + C_first;
            Remove τ_first from ReadyQueue;
        end
        Compute the WCRT R_i^** for τ_i;
        if R_i^** ≤ D_i then
            τ_i is schedulable;
            break;
        end
        τ_i is not schedulable;
    else
        τ_i is not schedulable;
    end
end
```

U and the last generated task is discarded. Finally, for every generated task-set we assume that priorities are assigned by following the RM policy. That is, the shorter the period of a task the higher its priority.

▷ **Considered metrics.** We evaluated the fraction of schedulable task-sets by following RM[8], NP-HBC and NP-CBH (see Figure 8). On another front, we measured the *minimum*, *average* and *maximum* worst-case response time of the lowest priority task for each generated task-set by using the following ratios, where R_n^* and R_n^{**} are defined by Equation 20 and Algorithm 1 (see Figure 9).

$$\text{dev}^*_{\tau_n} = \frac{|R_n^* - R_n|}{R_n^*} \quad \text{and} \quad \text{dev}^{**}_{\tau_n} = \frac{|R_n^{**} - R_n|}{R_n^{**}}$$

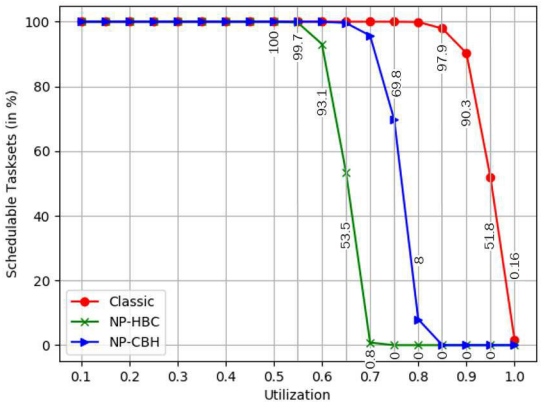
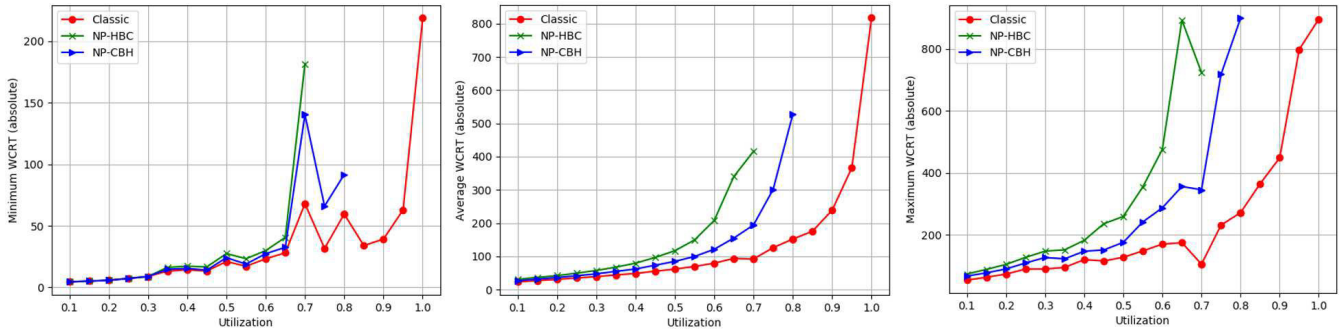


Fig. 8: Schedulability ratio.

▷ **Interpretation of the results.** From Figure 8, the effects of temperature starts to be visible when $U = 0.5$. Before this threshold, all the task-sets are schedulable. The picture changes when $U > 0.5$. We notice significant discrepancies among the three scheduling strategies and NP-CBH consistently dominates NP-HBC in terms of schedulability ratio and task responsiveness (see Figure 9). For example at $U = 0.7$, only 0.8% of the task-sets are schedulable by NP-HBC, whereas more than 85% are schedulable by NP-CBH. This represents a gap of more than 80%. Above $U = 0.75$, both NP-HBC and NP-CBH perform poorly. This can be explained by the effect of temperature on the processor activity. For example, at $U = 0.8$, no task-set is schedulable by NP-HBC, whereas only 8% are by NP-CBH. At $U = 1$, 0.16% are schedulable by RM, but none is by NP-HBC or NP-CBH. Here, the number of schedulable task-sets by RM can be explained by the manner in which tasks' periods are generated. The periods cannot exceed $T = 2^2 \cdot 3^2 \cdot 5^2 = 900$, which also corresponds to the maximum possible least common multiple (lcm). The domination of NP-CBH over NP-HBC stems from the reduction of the length of the cooling phases as much as possible.

From Figure 9, the deviation in terms of worst-case response time of the lowest priority task increases as the system utilization increases and NP-CBH clearly dominates NP-HBC. From a system perspective, the greatest system utilization at which NP-HBC could find a valid schedule is $U = 0.7$, whereas some task-sets can still make it under NP-CBH, up until $U = 0.8$. Above this value, neither NP-CBH nor

[8]Recall that the thermal constraints are not considered at all for RM.



(a) Minimum. (b) Average. (c) Maximum.

Fig. 9: [Minimum - Average - Maximum] Worst-case response time of the lowest priority task.

NP-HBC could find a valid schedule. This is clearly due to the introduction of cooling phases. Note that any small change in the trend of RM is replicated in that of NP-CBH and NP-HBC. However, NP-CBH absorbs the deviations better. This can be explained by the fact that NP-CBH allows the execution of a new task as soon as possible, whereas NP-HBC always requires the processor to cool-down to its minimum temperature level.

VII. Related Work

In this section we discuss the existing works dealing with thermal-aware scheduling techniques upon single processor platforms. The presented contributions are not exhaustive, but representative.

Most of the contributions in this context consider preemptive task sets and rely on scaling down the processor speed (through a combination of voltage and frequency scaling (DVFS)) to reduce power consumption and thereby temperature [13]–[17]. Here, actions are taken only when temperature gets above a certain threshold and/or below through requests issued by task/scheduler. By using such an approach, Quan et al. [17] derived thermal feasibility checks and formulated constructive speed scheduling algorithms for periodic tasks. Chen et al. [16] explored thermal-constrained speed scheduling, but for a set of frame-based real-time tasks with the same period. They developed a proactive speed scheduling by using dynamic voltage/speed scaling (DVS). In general, these techniques are proposed to avoid hotspots and/or minimize peak temperature and energy consumption [18]–[20]. On another front, Zhang et al. [21] presented a fully polynomial time approximation schemes (FPTAS) for the latency minimization of a set of periodic tasks executing on a processor subject to thermal constraints and proved the problem to be NP-hard [22]. Unfortunately, all these approaches cannot be applied to systems which are not capable of Dynamic Frequency Scaling. Recently, Ahmed et al. [23] investigated the schedulability analysis for thermal-aware real-time periodic tasks on single cores. They established a necessary and sufficient condition and later extended it to multi-cores. Then, by considering non-concrete sporadic tasks, Chandarli et al. [4] proposed an alternative solution. Specifically, they adapted energy harvesting techniques [24], [25] and proposed a response-time based analysis. To date, we are not aware of any work considering non-preemptive policies in the context of thermal-aware scheduling. As discussed in the previous sections, this problem may look simple at first glance, but it is far from being the case. The only fact that the temperature has to be kept permanently within specified boundaries adds an orthogonal and tremendous level of difficulty, unfortunately. This paper fills this gap and we strongly believe that it paves the way towards deriving sound and efficient solutions for the multi-core settings.

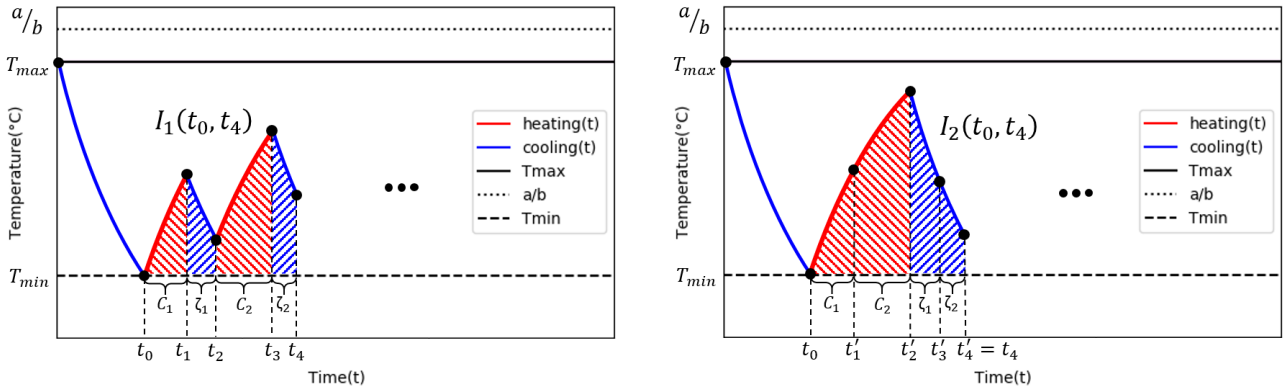
VIII. Conclusion and Future Work

This paper considered the thermal-aware schedulability analysis of non-preemptive real-time tasks on a single processor platform. We captured both the thermal and timing behaviors of the system in the same framework by proposing two new schedulers (a reactive scheduler NP-HBC and a proactive scheduler NP-CBH), together with their schedulability analysis, to maintain the processor temperature within predefined boundaries, while guaranteeing that all the timing constraints are met. We validated the run-time behavior of our solutions through intensive simulations by using the typical thermal specifications of an ARM-Cortex-A9 processor. As future work, we plan to (i) rigorously address the time complexity of the proposed approach; (ii) explore various priority assignment strategies when no restriction is imposed on the priority of tasks, before we can address the multi-core problem; and finally (iii) perform experiments on real platforms to expose the differences due to model abstraction.

Appendix: Insights for the Proof of Property 4–Point (iii)

The complete proof is performed by induction on the number of tasks. It cannot be faithfully reported in this paper due to space limitation. However, the key steps are as follows. The average temperature, denoted by T_{avg}, over a time window, say $[u, v]$, is defined by:

$$T_{\text{avg}}(u, v) \stackrel{\text{def}}{=} \frac{1}{v-u} \int_u^v T(t)dt$$



(a) Cooling between every two job executions.

(b) Cooling at once.

Fig. 10: Cooling between every two job executions vs. Cooling at once.

To be convinced of the correctness of the claim, it is sufficient to consider a set of two tasks $\{\tau_1, \tau_2\}$ with WCETs C_1 and C_2; which require cooling periods of lengths $\zeta_1 \geq 0$ and $\zeta_2 \geq 0$, respectively. Because the tasks are executed in a non-preemptive manner, we assume that $T_{\max}$ is high enough to accommodate the execution of τ_1 and τ_2 during a single heating phase. Also, without any loss of generality, we assume that τ_1 starts executing at time instant t_0 (i.e., when the processor is at its minimum temperature ($T_{\min}$)), followed by task τ_2 (see Figure 10). In this figure, we distinguish Scenario $\mathcal{S}_1$, where a cooling is performed between every two job executions (see Figure 10a); and Scenario $\mathcal{S}_2$, where a cooling is performed at once after the execution of the tasks (see Figure 10b). In Scenario $\mathcal{S}_1$, let $t_1 \stackrel{\text{def}}{=} t_0 + C_1$; $t_2 \stackrel{\text{def}}{=} t_1 + \zeta_1$; $t_3 \stackrel{\text{def}}{=} t_2 + C_2$; and finally $t_4 \stackrel{\text{def}}{=} t_3 + \zeta_2$. In the same vein, in Scenario $\mathcal{S}_2$, let $t'_1 \stackrel{\text{def}}{=} t_0 + C_1$; $t'_2 \stackrel{\text{def}}{=} t'_1 + C_2$; $t'_3 \stackrel{\text{def}}{=} t'_2 + \zeta_1$; and finally $t'_4 \stackrel{\text{def}}{=} t'_3 + \zeta_2$. Remark that $t_4 = t'_4$. Furthermore, let $I_1(t_0, t_4)$ and $I_2(t_0, t_4)$ be the shaded regions defined by the execution of the tasks and cooling periods in Scenarios $\mathcal{S}_1$ and $\mathcal{S}_2$, respectively. We have to show that the average temperature over $[t_0, t_4]$ in Scenario $\mathcal{S}_1$ is less than or equal to the average temperature over $[t_0, t_4]$ in Scenario $\mathcal{S}_2$. Formally, we have to show that Equation 26 holds true.

$$\frac{1}{t_4 - t_0} \cdot I_1(t_0, t_4) \leq \frac{1}{t_4 - t_0} \cdot I_2(t_0, t_4) \tag{26}$$

i.e., $I_2(t_0, t_4) - I_1(t_0, t_4) \geq 0$.

By using the additive property of integrals and the same reasoning as in Lemma 2, we have:

$$\begin{aligned} I_1(t_0, t_4) = &\int_{t_0}^{t_1} \left\{ \frac{a}{b} + \left(T_{\min} - \frac{a}{b} \right) \cdot e^{-b(t-t_0)} \right\} dt \\ &+ \int_{t_1}^{t_2} \left\{ T_{\min} + \frac{a}{b} e^{bC_1} - \frac{a}{b} \right\} \cdot e^{-b(t-t_0)} \, dt \\ &+ \int_{t_2}^{t_3} \left\{ \frac{a}{b} + \left(T_{\min} - \frac{a}{b} \left(e^{b(C_1+\zeta_1)} - e^{bC_1} + 1 \right) \right) \cdot e^{-b(t-t_0)} \right\} dt \\ &+ \int_{t_3}^{t_4} \left\{ T_{\min} + \frac{a}{b} e^{b(C_1+\zeta_1+C_2)} - \frac{a}{b} \left(e^{b(C_1+\zeta_1)} - e^{bC_1} + 1 \right) \right\} \cdot e^{-b(t-t_0)} \, dt \end{aligned}$$

By omitting all intermediate steps, a careful algebraic computation of the right-hand-side of this expression leads us to:

$$\begin{aligned} I_1(t_0, t_4) = &\frac{a}{b} \cdot (C_1 + C_2) + \frac{1}{b} \cdot T_{\min} - \frac{1}{b} \cdot T_{\min} \cdot e^{-b(C_1+C_2+\zeta_1+\zeta_2)} \\ &- \frac{a}{b^2} \cdot e^{-b\zeta_2} + \frac{a}{b^2} \cdot e^{-b(C_2+\zeta_2)} - \frac{a}{b^2} \cdot e^{-b(C_2+\zeta_1+\zeta_2)} \\ &+ \frac{a}{b^2} \cdot e^{-b(C_1+C_2+\zeta_1+\zeta_2)} \end{aligned}$$

Similarly, we have:

$$\begin{aligned} I_2(t_0, t_4) = &\int_{t_0}^{t'_1} \left\{ \frac{a}{b} + \left(T_{\min} - \frac{a}{b} \right) \cdot e^{-b(t-t_0)} \right\} dt \\ &+ \int_{t'_1}^{t'_2} \left\{ \frac{a}{b} + \left(T_{\min} - \frac{a}{b} \right) \cdot e^{-b(t-t_0)} \right\} dt \\ &+ \int_{t'_2}^{t'_3} \left\{ T_{\min} + \frac{a}{b} e^{b(C_1+C_2)} - \frac{a}{b} \right\} \cdot e^{-b(t-t_0)} \, dt \\ &+ \int_{t'_3}^{t_4} \left\{ T_{\min} + \frac{a}{b} e^{b(C_1+C_2)} - \frac{a}{b} \right\} \cdot e^{-b(t-t_0)} \, dt \end{aligned}$$

Again, by omitting all the intermediate steps, a careful algebraic computation of the right-hand-side leads us to:

$$\begin{aligned} I_2(t_0, t_4) = &\frac{a}{b} \cdot (C_1 + C_2) + \frac{1}{b} \cdot T_{\min} - \frac{1}{b} \cdot T_{\min} \cdot e^{-b(C_1+C_2+\zeta_1+\zeta_2)} \\ &- \frac{a}{b^2} \cdot e^{-b(\zeta_1+\zeta_2)} + \frac{a}{b^2} \cdot e^{-b(C_1+C_2+\zeta_1+\zeta_2)} \end{aligned}$$

Consequently

$$\begin{aligned} I_2(t_0, t_4) - I_1(t_0, t_4) \geq 0 &\Longleftarrow \frac{a}{b^2} \cdot e^{-b\zeta_2} + \frac{a}{b^2} \cdot e^{-b(C_2+\zeta_1+\zeta_2)} \\ &\quad - \frac{a}{b^2} \cdot e^{-b(\zeta_1+\zeta_2)} - \frac{a}{b^2} \cdot e^{-b(C_2+\zeta_2)} \geq 0 \\ &\Longleftarrow \frac{a}{b^2} e^{-b\zeta_2} \left\{ 1 + e^{-b(C_2+\zeta_1)} - e^{-b\zeta_1} - e^{-bC_2} \right\} \geq 0 \\ &\Longleftarrow (1 - e^{-b\zeta_1}) - e^{-bC_2}(1 - e^{-b\zeta_1}) \geq 0 \\ &\Longleftarrow (1 - e^{-b\zeta_1})(1 - e^{-bC_2}) \geq 0 \end{aligned}$$

Since $b\zeta_1 \geq 0$ and $bC_2 \geq 0$, it follows that $e^{-b\zeta_1} \leq 1$ and $e^{-bC_2} \leq 1$. This means that $(1 - e^{-b\zeta_1})(1 - e^{-bC_2}) \geq 0$. Therefore, Equation 26 holds true and the property follows.

ACKNOWLEDGMENT

This work was partially supported by National Funds through FCT/MCTES (Portuguese Foundation for Science and Technology), within the CISTER Research Unit (UID/CEC/04234); also by the Operational Competitiveness Programme and Internationalization (COMPETE 2020) under the PT2020 Partnership Agreement, through the European Regional Development Fund (ERDF), and by national funds through the FCT, within project POCI-01-0145-FEDER-029119 (PREFECT); and by the European Union through the Clean Sky 2 Joint Undertaking, under the H2020 Framework Programme (H2020-CS2-CFP08-2018-01), grant agreement nr. 832011 (THERMAC).

Finally, we are grateful to the anonymous reviewers for their constructive input. Their comments and suggestions have greatly improved this manuscript.

REFERENCES

[1] J. Liu, *Real-time systems*. Upper S. River: Prentice Hall, 2000.

[2] L. Cucu-Grosjean and J. Goossens, "Exact schedulability tests for real-time scheduling of periodic tasks on unrelated multiprocessor platforms," *J. Syst. Archit.*, vol. 57, no. 5, pp. 561–569, May 2011.

[3] V. Nelis, P. M. Yomsi, and J. Goossens, "Feasibility intervals for homogeneous multicores, asynchronous periodic tasks, and FJP schedulers," in *21st Int. Conference on Real-Time Networks and Systems*. NY, USA: ACM, 2013, pp. 277–286.

[4] Y. Chandarli, N. Fisher, and D. Masson, "Response time analysis for thermal-aware real-time systems under fixed-priority scheduling," in *IEEE 18th International Symposium on Real-Time Distributed Computing*, April 2015, pp. 84–93.

[5] K. Skadron, M. R. Stan, K. S., W. Huang, S. Velusamy, and D. Tarjan, "Temperature-aware microarchitecture: Modeling and implementation," *ACM Trans. Archit. Code Optim.*, vol. 1, no. 1, pp. 94–125, Mar. 2004.

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

[7] S. Wang, Y. Ahn, and R. Bettati, "Schedulability analysis in hard real-time systems under thermal constraints," *Real-Time Syst.*, vol. 46, no. 2, pp. 160–188, Oct 2010.

[8] M. Ahmed, N. Fisher, S. Wang, and P. Hettiarachchi, "Minimizing peak temperature in embedded real-time systems via thermal-aware periodic resources," *Sustainable Computing: Informatics and Systems*, vol. 1, no. 3, pp. 226 – 240, 2011.

[9] L. George, N. Rivierre, and M. Spuri, "Preemptive and Non-Preemptive Real-Time UniProcessor Scheduling," INRIA, Tech. Rep. RR-2966, 1996, rEFLECS.

[10] R. J. Bril, J. J. Lukkien, and W. F. Verhaegh, "Worst-case response time analysis of real-time tasks under fixed-priority scheduling with deferred preemption," *Real-Time Syst.*, vol. 42, pp. 63–119, 2009.

[11] G. C. Buttazzo, M. Bertogna, and G. Yao, "Limited preemptive scheduling for real-time systems. a survey," *IEEE Transactions on Industrial Informatics*, vol. 9, no. 1, pp. 3–15, Feb 2013.

[12] R. I. Davis, A. Burns, R. J. Bril, and J. J. Lukkien, "Controller area network (CAN) schedulability analysis: Refuted, revisited and revised," *Real-Time Systems*, vol. 35, no. 3, pp. 239–272, April 2007.

[13] D. Rajan and P. S. Yu, "On temperature-aware scheduling for single-processor systems," in *High Performance Computing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 342–355.

[14] K. Skadron, M. R. Stan, W. Huang, S. Velusamy, K. S., and D. Tarjan, "Temperature-aware microarchitecture," in *30th Int. Symp. on Comp. Arch.*, June 2003, pp. 2–13.

[15] S. Wang and R. Bettati, "Reactive speed control in temperature-constrained real-time systems," in *18th Euromicro Conference on Real-Time Systems*, July 2006, pp. 10 pp.–170.

[16] J. Chen, S. Wang, and L. Thiele, "Proactive speed scheduling for real-time tasks under thermal constraints," in *15th IEEE Real-Time and Embedded Technology and App. Symp.*, April 2009, pp. 141–150.

[17] G. Quan and Y. Zhang, "Leakage aware feasibility analysis for temperature-constrained hard real-time periodic tasks," in *21st Euromicro Conference on Real-Time Systems*, July 2009, pp. 207–216.

[18] Y. Liu, H. Yang, R. P. Dick, H. Wang, and L. Shang, "Thermal vs Energy optimization for DVFS-enabled processors in embedded systems," in *8th Int. Symp. on Quality Electronic Design*, March 2007, pp. 204–209.

[19] M. Bao, A. Andrei, P. Eles, and Z. Peng, "On-line thermal aware dynamic voltage scaling for energy optimization with frequency/temperature dependency consideration," in *46th ACM/IEEE Design Automation Conference*, July 2009, pp. 490–495.

[20] S. Kiamehr, M. Ebrahimi, M. S. Golanbari, and M. B. Tahoori, "Temperature-aware dynamic voltage scaling to improve energy efficiency of near-threshold computing," *IEEE Trans. on Very Large Scale Int. Systems*, vol. 25, no. 7, pp. 2017–2026, 2017.

[21] S. Zhang and K. S. Chatha, "Approximation algorithm for the temperature-aware scheduling problem," in *IEEE Int. Conf. on Comp-Aided Design*, Nov 2007, pp. 281–288.

[22] S. Arora and B. Barak, "Computational complexity - a modern approach," 2009.

[23] R. Ahmed, P. Ramanathan, and K. K. Saluja, "Necessary and sufficient conditions for thermal schedulability of periodic real-time tasks," in *26th Euromicro Conference on Real-Time Systems*, July 2014, pp. 243–252.

[24] C. Moser, D. Brunelli, L. Thiele, and L. Benini, "Real-time scheduling with regenerative energy," in *18th Euromicro Conf. on Real-Time Systems*, July 2006, pp. 10 pp.–270.

[25] Y. Abdeddaïm, Y. Chandarli, and D. Masson, "The optimality of PFPasap algorithm for fixed-priority energy-harvesting real-time systems," in *25th Euromicro Conf. on Real-Time Systems*, July 2013, pp. 47–56.