Noname manuscript No.
(will be inserted by the editor)

Energy-Efficient Thermal-Aware Multiprocessor Scheduling for Real-Time Tasks Using TCPN

L. Rubio-Anguiano [1], G. Desirena-López[1], A. Ramírez-Treviño[1] and J.L. Briz [2]

Received: date / Accepted: date

Abstract We present an energy-efficient thermal-aware real-time global scheduler for a set of hard real-time (HRT) tasks running on a multiprocessor system. This global scheduler fulfills the thermal and temporal constraints by handling two independent variables, the task allocation time and the selection of clock frequency.

To achieve its goal, the proposed scheduler is split into two stages. An off-line stage, based on a deadline partitioning scheme, computes the cycles that the HRT tasks must run per deadline interval at the minimum clock frequency to save energy while honoring the temporal and thermal constraints, and computes the maximum frequency at which the system can run below the maximum temperature. Then, an on-line, event-driven stage performs global task allocation applying a Fixed-Priority Zero-Laxity policy, reducing the overhead of quantum-based or interval-based global schedulers. The on-line stage embodies an adaptive scheduler that accepts or rejects soft RT aperiodic tasks throttling CPU frequency to the upper lowest available one to minimize power consumption while meeting time and thermal constraints. This approach leverages the best of two worlds: the off-line stage computes an ideal discrete HRT multiprocessor schedule, while the on-line stage manage soft real-time aperiodic tasks with minimum power consumption and maximum CPU utilization.

Keywords TCPN · modeling · scheduling

1

CINVESTAV. Av del Bosque 1145, Bajío, 45019 Zapopan, Jal
E-mail: {lerubio, gdesirena, art}@gdl.cinvestav.mx

2

Universidad de Zaragoza, María de Luna 3, 50009 Zaragoza, España
E-mail: briz@unizar.es

1 Introduction

Multicore microprocessors are a promising platform for embedded real-time (RT) systems. They allow the optimization of space, weight and power (SWaP) requirements, but they pose new scheduling challenges far beyond the long-established RT theory and practice for unicores. For example, CPU task allocation can lead to hot spots, reducing a microprocessor lifespan. Also, mobile devices call for a careful energy management to extend battery life. Hence, a multicore RT scheduler must consider temperature and energy besides the timing constraints. Many current MPSoCs provide power management mechanisms such as dynamic voltage and frequency scaling (DVFS), which can lower voltage and clock frequency simultaneously, thus reducing energy consumption quadratically.

Global schedulers dynamically allocate any task to any processor whereas partitioned schedulers statically divide tasks among the available processors (Sec.2.3). Only global scheduling has been proved hard RT (HRT) and soft RT (SRT) optimal for implicit deadline task sets, whereas partitioning is limited to a 50% utilization bound (Oh and Bakker [1998]). No optimal scheduler exists for constrained or arbitrary deadlines (Fisher et al. [2010]). However, context-switching and migration costs are high in global schedulers, although techniques like *deadline partitioning* (Funk et al. [2011], Moulik et al. [2017]) can lower that overhead. Recent state-of-the-art contributions like Brandenburg and Gül [2016], Casini et al. [2017] prove that *ad-hoc* approaches based on mixing semipartitioning with a number of techniques can be optimal in specific cases, always assuming unavoidable context-switch and migration costs. Meanwhile, the industry remains understandably conservative as we discuss in Sec. 2.3. Thus, there are still many open issues in the RT multiprocessor scheduling arena even when restricted to sporadic tasks with implicit deadlines without resource sharing. This become all the more complex when additional constraints like a bounded maximum temperature are considered.

We propose in this paper a thermal-aware, energy-efficient RT global scheduler based on an off-line stage and an on-line stage. The off-line stage takes two steps. The first step computes the minimum frequency (F^*) required to correctly run the HRT task set at maximum CPU utilization, since F^* is minimum, then the power consumption is minimized. The second step computes the maximum frequency (F^+) subject to the 100% CPU utilization and thermal constraints. In other words, any frequency increase above F^+ would imply a thermal bound violation. Then, the off-line stage leverages a deadline partitioning approach to compute the CPU cycles that each task must run per deadline interval. We get around the NP-completeness of the workload problem by splitting the hyperperiod into deadline intervals. An on-line stage performs the task allocation leveraging a simple Fixed-Priority Zero-Laxity policy (FPZL, Davis and Burns [2011a]). It includes an Adaptive Scheduler (AS) which adds robustness by throttling CPU frequency and accepting or rejecting aperiodic tasks, thus ensuring the correct execution of the HRT task set minimizing energy consumption and keeping a controlled temperature.

The main contribution of this paper is to show the applicability of TCPNs to HRT multiprocessor scheduling. We include thermal constraints and SRT aperiodic task management to pose a scenario which is considered non-trivial in today's HRT multiprocessor scheduling. However, there are implicit meaningful contributions to the state-of-the-art in the field as well. First, the unimodularity of the HRT and thermal constraint matrices reduces the complexity of the off-line stage to a linear programming problem (LPP), which yields task cycles per deadline interval at minimum CPU frequency (minimum power). Second, leveraging a FPZL policy makes the on-line allocation stage event-driven instead of quantum- or interval-driven, dramatically reducing the number of scheduling points (i.e. the chances of a context-switch and migration) to three minimal events (zero-laxity, job completion and aperiodic arrival). Last, the combination of the off-line stage with the AS manages aperiodic task arrival while guaranteeing HRT and thermal constraints with optimized power consumption.

This article is a journal extension of a conference paper (Rubio-Anguiano et al. [2018]). We now provide complete proofs and obtain context-switch and migration bounds. Also, we present further experiments assuming a more realistic DVFS mechanism based on the XScale multicore processor (Chen and Kuo [2007]), and we include a comparison with a global EDF (*g-EDF*) scheduler and a *pfair /deadline partitioning* hybrid scheduler.

The work is organized as follows. Section 2 presents basic concepts related with Petri nets and Timed Continuous Petri nets, and the forced TCPN equation as we use it. Section 3 states the scheduling problem herein addressed. Section 4 describes the modeling methodology. Section 5 describes the off-line stage of the scheduler. Section 6 extends on the on-line stage, providing context-switch and migration bounds, detailing the aperiodic scheduler, and stating algorithm complexity. Section 7 compares the proposal with two thermal-aware schedulers based on global EDF and *pfair*, discussing simulation results. Last, Section 8 draw some conclusions and points out future work.

2 Background

The following two subsections introduce basic necessary definitions concerning Petri nets and Timed Continuous Petri nets. An interested reader may also consult Mahulea et al. [2008], Desel and Esparza [1995], Silva and Recalde [2007] to get a deeper insight in the field. A last subsection provides insight on related research on multiprocessor real-time scheduling, highlighting the novelty of our proposal.

2.1 Discrete Petri Nets

Definition 1 A (discrete) Petri net is the 4-tuple $N = (P, T, \boldsymbol{Pre}, \boldsymbol{Post})$ where P and T are finite disjoint sets of places and transitions, respectively. $\boldsymbol{Pre}$ and $\boldsymbol{Post}$ are $|P| \times |T|$ $\boldsymbol{Pre}-$ and $\boldsymbol{Post}-$ incidence matrices, where $\boldsymbol{Pre}(i, j) > 0$ (resp. $\boldsymbol{Post}(i, j) > 0$) if there is an arc going from t_j to p_i (resp. going from p_i to t_j), otherwise $\boldsymbol{Pre}(i, j) = 0$ (resp. $\boldsymbol{Post}(i, j) = 0$).

Definition 2 A (discrete) Petri net system is the pair $Q = (N, M)$ where N is a Petri net and $M : P \rightarrow \mathbb{N} \cup \{0\}$ is the marking function assigning zero or a natural number to each place. The marking is also represented as a column vector $\boldsymbol{M}$, such that its $i - th$ element is equal to $M(p_i)$, named the tokens residing into p_i. $\boldsymbol{M}_0$ denotes the initial marking distribution.

A transition $t \in T$ is said enabled at the marking $\boldsymbol{M} \in \mathbb{N}^{|P|}$ *iff* $\boldsymbol{M} \geq \boldsymbol{Pre}[p, t]$, the occurrence or firing of an enabled transition leads to a new marking distribution $\boldsymbol{M'} \in \mathbb{N}^{|P|}$ that can be computed by using $\boldsymbol{M' = M + C}[P, t] = \boldsymbol{M + C} \cdot \boldsymbol{e_t}$, where $\boldsymbol{C = Post - Pre}$ is named the incidence matrix, and $\boldsymbol{e_t}$ denotes the $t - th$ elementary vector ($\boldsymbol{e_t}(k) = 1$ if $k = t$, otherwise $\boldsymbol{e_t}(k) = 0$).

2.2 Continuous and Timed continuous Petri nets

Definition 3 A continuous Petri Net (*ContPN*) is a pair $ContPN = (N, \boldsymbol{m}_0)$ where $N = (P, T, \boldsymbol{Pre}, \boldsymbol{Post})$ is a Petri net (PN) and $\boldsymbol{m}_0 \in \{\mathbb{R}^+ \cup 0\}^{|P|}$ is the initial marking.

The evolution rule is different from the discrete PN case. In continuous PN's the firing is not restricted to be integer. A transition t_i in a *ContPN* is *enabled* at $\boldsymbol{m}$ if $\forall\ p_j \in^{\bullet} t_i, \boldsymbol{m}[p_i] > 0$; and its *enabling degree* is defined as $enab(t_i, \boldsymbol{m}) = min_{p_j \in^{\bullet} t_i} \frac{\boldsymbol{m}[p_j]}{\boldsymbol{Pre}[p_j, t_i]}$. The firing of t_i in a certain positive amount $\alpha \leq enab(t_i, \boldsymbol{m})$ leads to a new marking $\boldsymbol{m' = m} + \alpha \boldsymbol{C}[P, t_i]$, where $\boldsymbol{C = Post - Pre}$ is computed as in the discrete case.

If $\boldsymbol{m}$ is reachable from $\boldsymbol{m}_0$ by firing the finite sequence $\boldsymbol{\sigma}$ of enabled transitions, then $\boldsymbol{m = m_0 + C \overrightarrow{\sigma}}$ is named the fundamental Eq. where $\overrightarrow{\boldsymbol{\sigma}} \in \{\mathbb{R}^+ \cup 0\}^{|T|}$ is the firing count vector, i.e $\overrightarrow{\boldsymbol{\sigma}}[t_j]$ is the cumulative amount of firings of t_j in the sequence σ.

Definition 4 A timed continuous Petri net (TCPN) is a time-driven continuous state system described by the tuple $(N, \boldsymbol{\lambda}, \boldsymbol{m}_0)$ where $(N, \boldsymbol{m}_0)$ is a continuous PN and the vector $\boldsymbol{\lambda} \in \{\mathbb{R}^+ \cup 0\}^{|T|}$ represents the transitions rates determining the temporal evolution of the system.

Transitions fire according to certain speed, which generally is a function of the transition rates and the current marking. Such function depends on the semantics associated to the transitions. Under the infinite server semantics Silva et al. [2011] the flow through a transition t_i (the transition firing speed) is defined as the product of its rate, λ_i, and $enab(t_i, \boldsymbol{m})$, the instantaneous enabling of the transition, i.e., $f_i(\boldsymbol{m}) = \lambda_i enab(t_i, \boldsymbol{m}) = \boldsymbol{\lambda}_i \min_{p_j \in {}^{\bullet}t_i} \frac{\boldsymbol{m}[p_j]}{\boldsymbol{Pre}[p_j,t_i]}$ (through the rest of this paper, for the sake of simplicity the flow through a transition t_i is denoted as f_i).

The firing rate matrix is denoted by $\boldsymbol{\Lambda} = diag(\lambda_1, ..., \lambda_{|T|})$. For the flow to be well defined, every continuous transition must have at least one input place, hence in the following we will assume $\forall t \in T, |{}^{\bullet}t| \geq 1$. The "min" in the above definition leads to the concept of configuration. A configuration of a *TCPN* at $\boldsymbol{m}$ is a set of arcs (p_i, t_j) such that p_i provides the minimum ratio $\boldsymbol{m}[p]/\boldsymbol{Pre}[p, t_i]$ among the places $p \in^{\bullet} t_i$ at the given marking $\boldsymbol{m}$. We say that p_i constrains t_j for each arc (p_i, t_j) in the configuration. A configuration matrix is defined for each configuration as follows:

$$\boldsymbol{\Pi(m)} = \begin{cases} \frac{1}{\boldsymbol{Pre}[i,j]} & \text{if } p_i \text{ is constraining } t_j \\ 0 & \textit{otherwise.} \end{cases} \tag{1}$$

The flow through the transitions of the net can be written in a vectorial form as $\boldsymbol{f(m) = \Lambda \Pi(m) m}$. The dynamical behaviour of a *PN* system is described by its fundamental equation:

$$\dot{\boldsymbol{m}} = \boldsymbol{C\Lambda\Pi(m)m} \tag{2}$$

In order to apply a control action to (2), a term u such that $0 \leq u_i \leq f_i(m)$ is added to every transition t_i to indicate that its flow can be reduced. Thus the *controlled flow* of transition t_i becomes $w_i = f_i - u_i$. Then, the forced state equation is

$$\begin{gathered} \dot{\boldsymbol{m}} = \boldsymbol{C}[\boldsymbol{f(m)} - \boldsymbol{u}] = \boldsymbol{Cw} \\ 0 \leq u_i \leq f_i(m) \end{gathered} \tag{3}$$

2.3 Related work

RT scheduling in multiprocessors has been traditionally tackled through two different approaches: *partitioned* and *global* scheduling (Baker [2005], Davis and Burns [2011b]) and mixed solutions (Calandrino et al. [2007]). Partitioned schedulers allocate tasks statically to CPUs, and tasks are not allowed to migrate. Under this scheme, HRT schedulability analysis can be derived from uniprocessor scheduling, ensuring a maximum utilization bound of 50%, which severely hampers the SWaP compromises (Oh and Bakker [1998]).

In contrast, global schedulers can allocate tasks to any CPU and allow task migration. Global Earliest Deadline First (*gEDF*) (Bertogna and Cirinei [2007]) is a global preemptive scheduling algorithm where all ready tasks are enqueued in a single ready queue, and the m highest-priority tasks are executed on the m processors. Job priorities are inversely proportional to their associated deadlines, with a smaller absolute deadline corresponding to a higher priority. This algorithm guarantees soft real-time (SRT) schedulability for implicit-deadline task sets, managing dynamic priorities at task level while fixing priorities at job level, but is not optimal under a HRT scheme (Goossens and Funk [2003], Bertogna et al. [2005]). Global scheduling can benefit from the concept of *fluid scheduling*, which consists in instantaneously sharing CPUs among all active jobs. Practical implementations approach this theoretical behavior by interleaving jobs, keeping a *fair* CPU share within time periods, and honoring time constraints in the case of RT tasks sets. Upon this principles, global schedulers based on *proportionate fairness* like *pfair* (Baruah et al. [1996]), PD (Baruah et al. [1995]) and PD^2 (Anderson and Srinivasan [2001]) achieve HRT optimal schedulability for implicit-deadline tasks sets. In *pfair* algorithms, the time is discretized and the tasks can only be executed during an integer number of quanta Q. The time quanta are then fairly distributed between the tasks so that ζ, the difference between the execution time of every task τ_i and the fluid schedule is smaller than 1 quantum at any time.

The downside of *pfair* algorithms is that they incur in an unfeasible number of context switches and migrations, since scheduling actions are taken on a quantum basis. *Deadline partitioning* schedulers such as *Dpfair* (Funk et al. [2011]) alleviate that overhead by limiting the scheduling points to the set of all task deadlines, i.e scheduling actions are now at variable time intervals instead of at a fixed quantum. *RT-TCPN* (Desirena-Lopez et al. [2016]) leverages a TCPN system model (tasks and CPUs) to track the fluid schedule as closely as possible by resorting to a sliding mode feedback controller. Every task is continuously executed at a rate equal to its utilization factor, formulated as a per-task continuous (fluid) function. The time is divided in intervals bounded by the successive deadlines in the ordered set of all task deadlines as in *deadline partitioning.* The fluid functions are calculated for every deadline interval. At any instant ζ, any task τ_i must have been executed $u_i \times \zeta$ units of time from the start of the schedule. This ensures fairness at every deadline interval. The discretization is performed on a quantum basis, with Q being the greater common divisor (GCD) of all deadlines in the set. When Q tends to 1, this approach tends to the *pfair* behaviour.

Thermal-aware scheduling has been widely studied for single core systems, exploiting DVFS to reduce power consumption and temperature (Kong et al. [2014], Hettiarachchi et al. [2014]). Chen et al. [2007] study the temperature problem in uniprocessors and in homogeneous multiprocessor systems. They leverage a partitioned EDF-based algorithm that minimizes energy, and derive an approximation bound for the maximum temperature. Schor et al. [2012] perform a worst-case temperature analysis for RT tasks with non-deterministic workloads running on multiprocessors systems. Ahmed et al. [2016] tackle the problem of thermal constrained scheduling of periodic tasks. Chantem et al. [2011] use an equivalent circuit model to estimate the temperature for a given set of HRT tasks on a multicore system, also referred to a partitioned scheme.

Feedback methods from control theory have been often used to cope with a dynamic environment for RT scheduling. The feedback control algorithm in Fu et al. [2010] enforces both thermal and RT constraints but is restricted to single-core processors, as they do not consider inter-core thermal coupling in multicore processors. A general framework of dynamic thermal management for multicore processors is proposed in Donald and Martonosi [2006]. It consists of a hierarchical feedback control loop with PI controllers but does not guarantee RT performance. The thermal problem is defined in Zanini et al. [2009] as a control theory problem with a state space representation, and proposes an optimum solution to the frequency assignment problem for thermal balancing of MPSoC, but it does not consider RT constraints nor the scheduling problem. Other contributions based on control theory are limited to SRT systems (Fu et al. [2009], Fu et al. [2012]), allowing for a certain percentage of missed deadlines.

Most former references leverage partitioned approaches. In recent years, context-switch and migration overhead has tipped the scale in favor of semipartitioned and empirical designs with few contributions left leveraging global scheduling techniques (Moulik et al. [2017]). In semipartitioned schedulers, some tasks are statically allocated to the processors whereas others are split across multiple processors as in global scheduling. This allow Casini et al. [2017] to deal with dynamic task sets, for example. Empirical designs resort to a combination of techniques, but can only solve specific problems (Brandenburg and Gül [2016]). However, the automotive, aeronautics and space industry, understandably conservative but already embracing multicores, keeps clinging to the traditional cyclic executive upon static partitioning (ARINC [1997], Diniz and Rufino [2005], AUTOSAR [2017]) suffering from unbalanced SWaP and facing tough problems such as the fair Worst Case Delay calculation, which lead to complex ILP solutions (Fernandez et al. [2017], Cardona et al. [2018]). Moreover, the problem of including additional constraints such as a maximum temperature or shared resources, still leaves many open avenues in the HRT multiprocessor scheduling arena, which gives global scheduling a chance because of its capability to achieve optimal CPU utilization.

In this vein, quantum-based schedulers such as *pfair* are good candidates to control temperature, at the cost of a high overhead. Deadline partitioning approaches limit context switching and migration, but the variable intervals between deadlines can be too long to cope with temperature variations. In Desirena-Lopez et al. [2019] we extended and improved the aforementioned approach taken in Desirena-Lopez et al. [2016] to include thermal constraints. The fluid time share that each task must be granted at each processor during the hyperperiod to meet HRT and thermal constraints is computed off-line in polynomial time by solving a LPP. This time share is a per-CPU (CPU_j), per-task (τ_i) fluid schedule function ${}_jFSC_{\tau_i}(\zeta)$, ζ being the current time. This functions implicitly provide a partition, and are calculated at each deadline interval over the ordered set of all task deadlines (i.e. at each $\zeta = sd_i$) following a deadline partitioning scheme. The fluid scheduler is discretized on-line, on a quantum basis, with $Q = GCD(sd_i)$. At each quantum, a sliding mode feedback controller makes a TCPN model of tasks and CPUs to evolve so that the error between the ${}_jFSC_{\tau_i}(\zeta)$ and the actual fluid execution time of τ_i on each CPU_j becomes minimum, accounting for disturbances. The evolution of the TCPN barely takes a simple linear computation. Then, a per-CPU task priority queue is build upon the difference between the fluid and the actual discrete execution time, and jobs are accordingly dispatched. The feedback controller allows the system to recover from disturbances such as CPU detentions due to environmental hazards causing energy interruptions or thermal peaks. The advantage of this approach is the implementation of an on-off control law, which leads to a low-overhead scheduler, capable of handling perturbations in underloaded systems without rescheduling a job.

As in Desirena-Lopez et al. [2019], we follow in this paper a scheme in two stages, off-line (LPP) and on-line, but there are substantial differences. In broad terms, we add frequency control to minimize power consumption and deal with aperiodic tasks, but differences go further when going into detail. First, the LPP solved during the off-line stage yields CPU cycles per task and deadline interval (over the set of all task deadlines, according to a deadline partitioning approach), instead of providing a fluid time share per task and CPU that must be discretized later on-line, as in Desirena-Lopez et al. [2019]. That is, there is no implicit partition or CPU allocation now, and the time share comes out already in cycles per deadline interval, also meeting the time and thermal constraints of the HRT task set, at the minimum available CPU frequency (F^*), in order to minimize power consumption. F^* is always below the maximum frequency (F^+) at which the system can run the task set without violating the thermal constraint. Second, the on-line stage is now event-based (zero-laxity, task completion and aperiodic task arrival events) instead of quantum-based, which dramatically plummets the overhead. Also, calculations in this stage are much lighter than in Desirena-Lopez et al. [2019], since the feedback controller can be obviated, and the only pending tasks left are task allocation and aperiodic task management, a capability unavailable in Desirena-Lopez et al. [2019]. Task allocation is performed by applying a Fixed-Priority Zero-Laxity algorithm (FPZL, Davis and Burns [2011a]). Aperiodic task arrival is managed by an Adaptive Scheduler (AS). which accepts or rejects arriving aperiodic tasks. The AS adds robustness by throttling CPU frequency, accepting or rejecting aperiodic tasks upon system limits. This warrants the correct execution of the HRT tasks, minimizes energy consumption,

Table 1: System notation

Symbol	**Description**
n	Number of tasks
m	Number of processors
$\mathcal{T}$	A task set
$\mathcal{P}$	a Processor set
τ_i	The i^{th} task
cc_i	The worst-case execution in CPU cycles of τ_i
c_i	The worst-case execution time of τ_i
d_i	The relative deadline of τ_i
ω_i	The period of τ_i
u_i	The utilization of task τ_i
U	System utilization
H	Hyperperiod
$\mathcal{F}$	Set of discrete frequencies
$\mathcal{F}^s$	Set of discrete operating frequencies $\mathcal{F}^s \subseteq \mathcal{F}$
$\Phi*$	The normalized minimum frequency
$F*$	The minimum frequency
F_c	The solution of Eq.(11)
F^+	The maximum thermal frequency
F_n	The operating frequency
sd_i^k	The $k-th$ deadline of task τ_i
SD	The set of ordered deadlines sd_i^q
I_{SD}^k	The $k-th$ scheduling interval
x_i^k	The cycles of task τ_i to be executed during I_{SD}^k
$x_{\tau_i^a}^k$	The cycles of task τ_i^a to be executed during I_{SD}^k
X^k	Set of all active tasks in I_{SD}^k
k_q	Thermal conductivity of component q
V_q	Volume of component q
V_{CPU_j}	Volume of CPU j
ρ_q	Density of component q
cp_q	Specific heat capacity of component q
h	Convection coefficient
T_q	Temperature of component q

and meets thermal constraints, much improving the scheduling scheme presented in Desirena-Lopez et al. [2019].

3 Problem definition

This section introduces the scheduling problem herein addressed. It starts setting the working scenario of tasks and then, it states the Minimum Energy Thermal-Aware RT Scheduler.

The set of independent periodic tasks is denoted by $\mathcal{T} = \{\tau_1, ..., \tau_n\}$. Each task is identified by the *3-tuple* $\tau_i = (cc_i, d_i, \omega_i)$, where cc_i is the worst-case execution time (WCET) in CPU cycles, ω_i is the task period, and d_i is the relative implicit deadline ($d_i = \omega_i$) (Baruah et al. [2015]). $\mathcal{P} = \{CPU_1, \ldots, CPU_m\}$ is a set of m identical processors with an homogeneous clock frequency $F \in \mathcal{F} = \{F_1, \ldots, F_{max}\}$.

We assume that all task parameters, including task period and CPU cycles are integers and that any task instance or *job* can be preempted at any time. The hyperperiod is defined as the period equal to the least common multiple of periods $H = lcm(\omega_1, \omega_2, \ldots, \omega_n)$ of the n periodic tasks. A task τ_i executed on a processor at its maximum frequency F_{max}, requires $c_i = \frac{cc_i}{F_{max}}$ processor time at every ω_i interval. The system utilization is defined as the fraction of time during which the processor is busy running the task set i.e., $U = \sum\limits_{i=1}^{n} \frac{c_i}{\omega_i}$. The CPU utilization must be computed and should be less or equal to the number of processors i.e., $U \leq m$ (Baruah et al. [1996]).

We also consider the arrival of asynchronous, aperiodic tasks. Each aperiodic task τ_i^a is defined as a *3-tuple* (cc_i^a, d_i^a, r_i^a) in which cc_i^a (required CPU cycles) and d_i^a (deadline) are known at task arrival time, and the arrival time r_i^a is unknown.

Formally, the problem addressed in this work is stated as follows.

Problem 31 *Minimum Energy Thermal Aware RT Scheduler (METARTS). Given the sets $\mathcal{T}$ of tasks and $\mathcal{P}$ of CPUs, the METARTS problem consists in designing an algorithm to allocate within the hyperperiod H the tasks in $\mathcal{T}$ to the m identical CPUs in such a way that the deadlines for $\mathcal{T}$ are always satisfied, the CPU temperatures are always kept below a given bound $\boldsymbol{T_{max}}$ and the consumed energy is minimum. Additionally, the scheduler must execute aperiodic tasks upon arrival subject to the temporal and thermal constraints, or reject the aperiodic task otherwise.*

4 System modeling Methodology

This section describes how we model tasks, CPU allocation, thermal behavior and energy consumption with a TCPN. The TCPN model is outlined in Fig. 1. Table 1 summarizes system parameters and symbols as used in the paper, and Table 2 gathers the notation related to our TCPN model.

4.1 Task model

A task τ_i arrives every ω_i time units to the system. This is modeled in the *TCPN module for tasks* of Fig. 1a (one dotted box per task) as follows. If $\lambda_i^{\omega} = \frac{1}{\omega_i}$, then one token is generated every ω_i. Since the arc weight from t_i^{ω} to p_i^{cc} is equal to cc_i, then cc_i tokens are added to p_i^{cc} every ω_i, and the marking of p_i^{cc} represents the WCET of task τ_i in CPU cycles (cc_i). The relative task

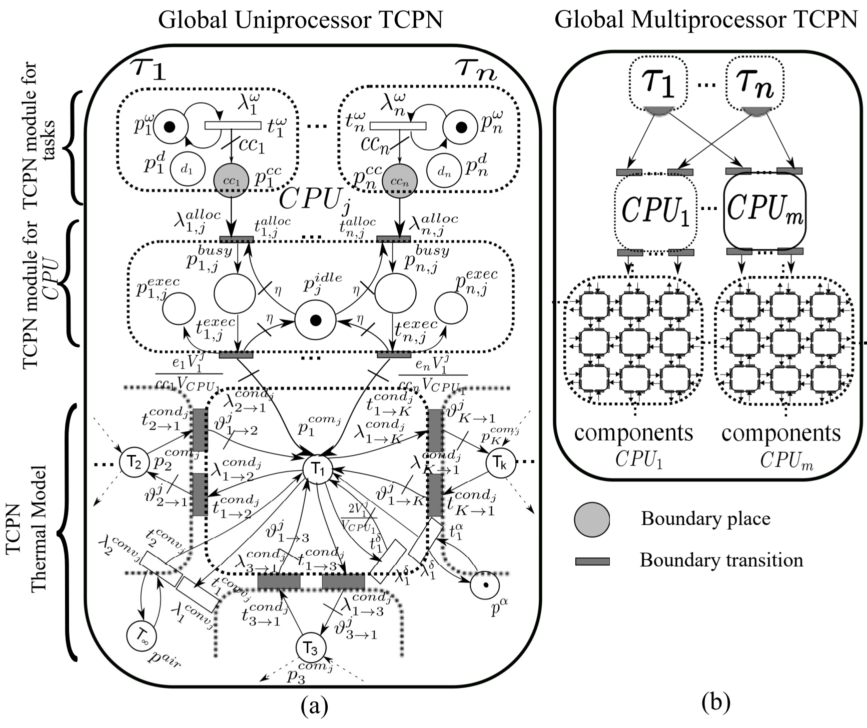

Fig. 1: TCPN model integrating task, CPU and thermal modeling. (a) details the case for a single CPU, and (b) zooms out and extend the model for a chip multiprocessor.

deadlines are modeled as the marking of places p_i^d. Relative deadlines are a constant parameter, therefore the marking of p_i^d remains constant, and places p_i^d have no input nor output arc, since we assume implicit deadlines, $d_i = \omega_i$.

Fig. 1a also shows the TCPN model of a single CPU (middle dotted box). We model the allocation of task cycles to different CPUs by linking the place p_i^{cc} of each τ_i to a transition boundary $t_{i,j}^{alloc}$ of each CPU_j. Fig. 1b provides an overview of a set of tasks and CPUs. Places p_j^{idle} and $p_{i,j}^{busy}$ respectively represent the idle state and the busy state of the CPU. The marking of place p_j^{idle} models the available CPU cycles (*throughput capacity*). The initial marking at p_j^{idle} is set to 1, indicating that CPU_j is idle.

The firing of $t_{i,j}^{alloc}$ allocates CPU cycles of task τ_i to CPU_j by moving tokens from place p_i^{cc} to place $p_{i,j}^{busy}$. The firing of $t_{i,j}^{exec}$ executes the allocated CPU cycles. The required CPU capacity is reserved during the allocation period and continuously released as task CPU cycles are executed. Arcs from p_j^{idle} to $t_{i,j}^{alloc}$ and from $t_{i,j}^{exec}$ to p_j^{idle} are weighted by a constant value η, to ensure that

Table 2: Notation for the TCPN model

TCPN symbol	**Description**
TCPN Module for task τ_i	
p_i^{ω}	Period place of τ_i
p_i^{d}	Deadline place of τ_i
p_i^{cc}	Cpu cycles place of τ_i
t_i^{ω}	period transition of τ_i
λ_i^{ω}	Fire rate of transition t_i^{ω}
TCPN Module for CPU_j	
p_j^{idle}	Idle state place of CPU_j
p_j^{pow}	Power place of CPU_j
$p_{i,j}^{busy}$	Busy state place of τ_i in CPU_j
$t_{i,j}^{alloc}$	Allocation transition of τ_i in CPU_j
$t_{i,j}^{exec}$	Execution transition of τ_i in CPU_j
$\lambda_{i,j}^{alloc}$	Fire rate transition of $t_{i,j}^{alloc}$
$\lambda_{i,j}^{exec}$	Fire rate transition of $t_{i,j}^{exec}$
η	CPU modeling parameter
TCPN Thermal model	
p_q^{com}	Place of component q
p_q^{air}	Place of component q
p_q^{α}	Place for leakage power of component q
$t_{q\to r}^{cond}$	Conduction transition from component q to component r
t_q^{conv}	Convection transition of component q
$t_{q\to\infty}^{conv}$	Convection transition of component q to air
t_q^{α}	Leakage power transition for α of component q
t_q^{δ}	Leakage power transition for δ of component q
$\lambda_{q\to r}^{cond}$	Fire rate transition of $t_{q\to r}^{cond}$
λ_q^{conv}	Fire rate transition of t_q^{conv}
$\lambda_{q\to\infty}^{conv}$	Fire rate transition of $t_{q\to\infty}^{conv}$
λ_q^{α}	Fire rate transition of t_q^{α}
λ_q^{δ}	Fire rate transition of t_q^{δ}

the flow in transitions $t_{i,j}^{alloc}$ is limited by the throughput capacity of the CPU (modeled by the marking of place p_j^{idle}).

4.2 Thermal modeling methodology

At the bottom of Fig. 1a (*TCPN Thermal Model*) we show a set of dotted boxes, only the central one being closed. Each box correspond to a prismatic element modeling the thermal properties and behavior of a specific chip area. Fig. 1b zooms out this view, to include a number of prismatic elements obeying a specific meshing of the chip multiprocessor. Firing the boundary transitions $t_{i,j}^{exec}$ (which model the execution of τ_i in CPU_j) adds tokens to places $p_i^{com_j}$ (at the center of each dotted box in the *TCPN Thermal Model*, with marking T_i). This activates the rest of transitions and places conforming this thermal model, representing heat transfer by conduction and convection, and thus causing temperature to increase because of the computing activity. A detailed explanation of the thermal model is provided in Desirena-Lopez et al. [2014].

4.3 Energy consumption modeling methodology

Let F be the CPU_j clock frequency during the time interval $(\zeta_1, \zeta_2]$. Then, the average energy consumed during this interval by the tasks running on CPU_j is defined as:

$$E_j = \int_{\zeta_1}^{\zeta_2} P_{CPU_j}(F) d_\zeta \tag{4}$$

$P_{CPU_j}(F)$ represents the power consumed by a CPU_j. It depends on the dynamic power due to computational activities of tasks ($P_{dyn}(F)$), and on the static power due to leakage (P_{leak}). It is computed as: $P_{CPU_j}(F) = P_{dyn_j}(F) + P_{leak_j} = \lambda_{i,j}^{exec} m_{i,j}^{busy}(\zeta) F^3 + P_{leak_j}$. P_{leak_j} can be modeled as a linear function of temperature (Ahmed et al. [2016]): $P_{leak} = \delta T + \rho$, where T is the CPUs temperature and δ and ρ are modeling constants.

4.4 TCPN model behavior

The dynamic behavior of the TCPN model introduced in Fig. 1a is described by the following equations:

$$\begin{aligned} \dot{\boldsymbol{m}}_{\boldsymbol{T}} =& \boldsymbol{C}_{\boldsymbol{T}} \boldsymbol{\Lambda}_{\boldsymbol{T}} \boldsymbol{\Pi}_{\boldsymbol{T}}(\boldsymbol{m}) \boldsymbol{m}_{\boldsymbol{T}} + \boldsymbol{C}_{\boldsymbol{a}} \boldsymbol{\Lambda}_{\boldsymbol{a}} \boldsymbol{\Pi}_{\boldsymbol{a}}(\boldsymbol{m}) \boldsymbol{m}_{\boldsymbol{a}} \\ & + \boldsymbol{C}_{\boldsymbol{\mathcal{P}}}^{\boldsymbol{exec}} \boldsymbol{f}^{\boldsymbol{exec}} \end{aligned} \tag{5a}$$

$$\dot{\boldsymbol{m}}_{\boldsymbol{a}} = 0 \tag{5b}$$

$$\dot{\boldsymbol{m}}_{\boldsymbol{\mathcal{T}}} = \boldsymbol{C}_{\boldsymbol{\mathcal{T}}} \boldsymbol{\Lambda}_{\boldsymbol{\mathcal{T}}} \boldsymbol{\Pi}_{\boldsymbol{\mathcal{T}}}(\boldsymbol{m}) \boldsymbol{m}_{\boldsymbol{\mathcal{T}}} - \boldsymbol{C}_{\boldsymbol{\mathcal{T}}}^{\boldsymbol{alloc}} \boldsymbol{w}^{\boldsymbol{alloc}} \tag{5c}$$

$$\dot{\boldsymbol{m}}_{\boldsymbol{\mathcal{P}}} = \boldsymbol{C}_{\boldsymbol{\mathcal{P}}} \boldsymbol{\Lambda}_{\boldsymbol{\mathcal{P}}} \boldsymbol{\Pi}_{\boldsymbol{\mathcal{P}}}(\boldsymbol{m}) \boldsymbol{m}_{\boldsymbol{\mathcal{P}}} + \boldsymbol{C}_{\boldsymbol{\mathcal{P}}}^{\boldsymbol{alloc}} \boldsymbol{w}^{\boldsymbol{alloc}} \tag{5d}$$

$$\dot{\boldsymbol{m}}_{\boldsymbol{exec}} = \boldsymbol{C}_{\boldsymbol{\mathcal{P}}}^{\boldsymbol{exec}} \boldsymbol{f}^{\boldsymbol{exec}} \tag{5e}$$

where $\boldsymbol{C}_{\boldsymbol{x}}$, $\boldsymbol{\Lambda}_{\boldsymbol{x}}$, and $\boldsymbol{\Pi}_{\boldsymbol{x}}\boldsymbol{(m)}$ are the incidence matrix, the firing rate transitions and the configuration matrix ($x = \{T, a, \mathcal{T}, \mathcal{P}\}$) of the thermal, tasks, and processors subnet respectively. $\boldsymbol{C}_{\boldsymbol{T}}^{\boldsymbol{alloc}}$, $\boldsymbol{C}_{\boldsymbol{\mathcal{T}}}^{\boldsymbol{alloc}}$ and $\boldsymbol{C}_{\boldsymbol{\mathcal{P}}}^{\boldsymbol{alloc}}$ stand for the connections of transitions $t_{i,j}^{alloc}$ from (to) places in the thermal model, the task model, and CPU throughput model respectively. Matrix $\boldsymbol{C}_{\boldsymbol{\mathcal{P}}}^{\boldsymbol{exec}}$ has the columns of the transitions $t_{i,j}^{exec}$ of the incidence matrix $\boldsymbol{C}_{\boldsymbol{\mathcal{P}}}.\boldsymbol{w}^{\boldsymbol{alloc}}$ is the controlled flow of the *allocation transitions* (i.e. the allocation rate of tasks to

CPUs). All matrices are computed using the TCPN theory presented in Section 2.

Eq. (5a) represents the evolution of system temperature. Eq. (5b) indicates that the environmental temperature keeps constant during observation time (its derivative is neglected). Eq. (5c) describes the arrival of periodic tasks to the system. Eq. (5d) models the CPUs cycles that are assigned to tasks. Finally, Eq. (5e) models task execution.

5 Off-line stage

This stage computes the time that each task job τ_i must run during the intervals defined by all deadlines in the task set. First, we compute the thermal constraint and frequency ranges. Then, we use these results as the constraints in an integer linear problem (ILP) to compute the execution times for tasks during deadlines. Fortunately, the constraints can be represented by an unimodular matrix, thus this ILP can be efficiently solved by a linear programming problem (LPP).

5.1 Thermal constraint

The TCPN thermal equation Eq. (5a) is used to derive the thermal constraints. Since the schedule must be periodic, i.e. it must be repeated every hyperperiod, so must be the thermal solution. In this case, the initial and final temperature in every hyperperiod must be equal. This can be approached by considering the steady state of the temperature, i.e. the steady state of the thermal equation:

$$\begin{aligned} \dot{\boldsymbol{m}}_{\boldsymbol{T}} &= \boldsymbol{A}\boldsymbol{m}_{\boldsymbol{T}} + \boldsymbol{B'}\boldsymbol{m}_{\boldsymbol{a}} + F^3\boldsymbol{B}\boldsymbol{f}^{\boldsymbol{exec}} \\ \boldsymbol{Y}_{\boldsymbol{T}} &= \mathbf{S}\mathbf{m}_{\mathbf{T}} \end{aligned} \tag{6}$$

where $\boldsymbol{A} = \boldsymbol{C_T \Lambda_T \Pi_T(m)}$, $\boldsymbol{B} = \boldsymbol{C_P^{exec}}$ and $\boldsymbol{B'} = \boldsymbol{C_a \Lambda_a \Pi_a(m)}$.

In a steady state temperature ($\boldsymbol{m_{T_{ss}}}$) when time tends to infinite $\dot{\boldsymbol{m}}_{\boldsymbol{T}} = \mathbf{0}$. Hence $\boldsymbol{m_{T_{ss}}} = -\boldsymbol{A^{-1}}(F^3\boldsymbol{Bw^{alloc}} + \boldsymbol{B'm_a})$. In order to not violate the thermal constraint of CPUs, the steady state temperature must be less than or equal to its maximum temperature level, i.e., $\boldsymbol{Sm_{T_{ss}}} \leq \boldsymbol{T_{max}}$ (thermal constraint) then:

$$-\boldsymbol{SA^{-1}}\, F^3\boldsymbol{Bw^{alloc}} \leq \boldsymbol{T_{max}} + \boldsymbol{SA^{-1}B'm_a} \tag{7}$$

This equation provides the thermal constraints that the allocation of tasks to the processors ($\boldsymbol{w^{alloc}}$) must fulfill. It includes the allocated tasks (which in the steady state are equivalent to the executed tasks), the clock frequency and the temperature bounds. This equation will be used to compute the range of feasible operation frequencies.

5.2 Minimum frequency

The proposed approach aims to minimize the system dynamic energy consumption under the constraints of the RT task deadlines and thermal limit. We explore the energy minimization under DVFS, which vary processor frequency by selecting one from a finite set of a preset values, i.e. $\mathcal{F} = \{F_1, ..., F_{max}\}$. For convenience, we normalize this set as $\phi = \{\phi_{min} = \frac{F_1}{F_{max}}, ..., 1\}$. Since all the design parameters are fixed in the CPU, the consumed energy is minimized in Eq. 4 *iff* the clock frequency F is minimized. Nevertheless, F must be high enough to ensure that the temporal constraints are met. The next proposition obtains the minimum clock frequency that fulfills the temporal constraints.

Proposition 1 *Assuming that the task utilization is less than the number of processors in the METARTS problem, the normalized clock frequency that minimizes the total energy consumption while meeting temporal constraints is constant:*

$$\Phi^* = \max\{\phi_{min}, \frac{1}{m}\sum_{i=1}^{n} \frac{cc_i}{\omega_i F_{max}}\}. \tag{8}$$

Proof According to Eq. (4), the energy has a minimum *iff* the consumer power is minimum. This occurs when ϕ^3 is minimum and fulfills that $\sum_{i=1}^{n} \frac{cc_i}{\phi w_i F_{max}} = m$, and $\phi \geq \phi_{min}$. Using Lagrange multipliers, the Lagrangian function is $L = \phi^3 + \mu_1(\frac{1}{\phi}\sum_{i=1}^{n} \frac{cc_i}{w_i F_{max}} - m) + \mu_2(\phi_{min} - \phi)$. The solution yields four cases: a) Both multipliers are inactive ($\mu_{1,2} = 0$); b) Both multipliers are active ($\mu_{1,2} \geq 0$); c) $\mu_1 = 0$ and $\mu_2 \geq 0$; and d) $\mu_1 \geq 0$ and $\mu_2 = 0$. The first case is unfeasible, because ϕ cannot be zero. In the second case, the only solution is $\phi = \phi_{min} = \frac{1}{m}\sum_{i=1}^{n} \frac{cc_i}{w_i F_{max}}$. Finally, if one multiplier is active while the other one is inactive there are two possible solutions: $\phi = \phi_{min}$ or $\phi = \frac{1}{m}\sum_{i=1}^{n} \frac{cc_i}{w_i F_{max}}$. Consequently, in order to fulfill both constraints, the normalized clock frequency that minimizes the total energy consumption becomes $\Phi^* = \max\{\phi_{min}, \frac{1}{m}\sum\limits_{i=1}^{n} \frac{cc_i}{\omega_i F_{max}}\}$. □

The normalized frequency Φ^* meets the temporal constraints. To guarantee that the thermal constraints are also fulfilled, we must compute $\boldsymbol{w^{alloc}}$ and solve Eq. (7). At frequency Φ^* the system utilization is $U = \sum_{i=1}^{n} \frac{cc_i}{\omega_i \phi^* F_{max}}$ $= m$ and the processor frequency is,

$$F^* = \min\{F \in \mathcal{F} | F \geq \Phi^* F_{max}\} \tag{9}$$

given the nature of the discrete set of frequencies. When computing Φ^* (Eq. 8) we assume a fully utilized system, but actual F^* in Eq. 9 can make the execution faster, causing the utilization to become below 100%, in those cases we introduce a dummy task $u_{dummy} = m - \sum\limits_{i=1}^{n} \frac{cc_i}{F^* \omega_i}$ to assure that system utilization is 100% . To make the distribution of the CPU cycles required to execute all tasks homogeneous, the task allocation ratio is calculated as:

$$\boldsymbol{w^{alloc}} = [\frac{1}{m}\sum_{i=1}^{n}\frac{cc_i}{\omega_i F^*}, \ \dots \ , \frac{1}{m}\sum_{i=1}^{n}\frac{cc_i}{\omega_i F^*}]^T \tag{10}$$

$\boldsymbol{w^{alloc}}$ controls the flow of the *allocation transitions* in the TCPN ($t_{i,j}^{alloc}$ in Fig. 1), thus modeling the allocation rate of tasks to CPUs (Eq. 5). If $\boldsymbol{w^{alloc}}$ satisfies Eq. (7), then the thermal constraints are also satisfied. Otherwise, the $METARTS$ problem does not have a solution. If it has a solution (Φ^* is feasible), then we can compute the maximum CPU cycles available for aperiodic tasks, and the maximum clock frequency that can be used subject to thermal constraints.

5.3 Maximum CPU cycles and clock frequency

The *maximum thermal frequency* $F^+ \in \mathcal{F}$ is the greatest frequency at which all CPUs can operate at 100% of utilization so that temperature meets the thermal constraint. To compute F^+, first we solve the programming problem in Eq.(11) to find the frequency upper bound F_c that satisfies the constraints.

$$\begin{aligned}
&max \quad F_c \\
&s.t. \\
&-\boldsymbol{SA^{-1}F_c^3B}\left[\tfrac{CC_1}{F_cH}, \ \dots \ , \tfrac{CC_m}{F_cH}\right]^T \leq \boldsymbol{T_{max}} + \boldsymbol{SA^{-1}B'm_a} \\
&\quad \tfrac{CC_j}{F_cH} = 1 \qquad \forall j = 1, \dots, m \\
&\quad F^* \leq F_c \leq F_{max}
\end{aligned} \tag{11}$$

The first constraint establish the thermal requirements. CC_j represents the cycles that CPU_j must execute per hyperperiod. Since all CPUs must work at their maximum capacity, the second constraint implies that the CPU utilization is 100%. The last constraint bounds F_c to the actual clock frequency range of CPUs. Finally, the solution for F^+ has to be in the set $\mathcal{F}$ of discrete frequencies, thus the processor frequency F^+ is calculated as,

$$F^+ = \max\{F \in \mathcal{F} | F \leq F_c\}. \tag{12}$$

With the minimum frequency $F*$ and the maximum thermal frequency F^+ we define

$$\mathcal{F}^s = \{F \in \mathcal{F} | F^* \leq F \leq F^+\}, \tag{13}$$

as the set of operating frequencies that meet the thermal constraint.

5.4 Deadline partitioning

Once frequency F^* is known (Eq. 9), the WCET that each task must run at each deadline interval can be computed. For this, we consider the ordered set of the deadlines of all tasks to define scheduling intervals, as in deadline partitioning (Funk et al. [2011]). Each task τ_i must be executed $n_i = \frac{H}{\omega_i}$ times within the hyperperiod H. Thus every $q_i * \omega_i$, where $q_i = 1, ..., n_i$ is a deadline that must be considered in the analysis. These deadlines $sd_i^{q_i}$ are ordered and joined in the set $SD_i = \{sd_i^1, ..., sd_i^{n_i}\}$. The general set of deadlines is defined as $SD = SD_0 \cup ... \cup SD_{|\mathcal{T}|}$ where $SD_0 = \{0\}$. The elements in SD can be arranged in ascendant order and renamed as $SD = \{sd_0, ..., sd_\alpha\}$, where sd_α is the last deadline. The scheduling interval $I_{SD}^k = [sd_k, sd_{k+1})$ is defined and $|I_{SD}^k| = sd_{k+1} - sd_k$ represents the scheduling interval duration. The proposed deadline partitioning problem assumes a 100% utilization on every CPU. However, because $F^* = \min\{F \in \mathcal{F} | F \geq \Phi^* F_{max}\}$, $F^* \neq \Phi^* F_{max}$ in general, and consequently idle cycles may appear. In order to solve this problem we introduce a dummy task with period H and utilization $u_{dummy} = m - \sum_{i=1}^{n} \frac{cc_i}{F^* \omega_i}$. Then the cycles that each HRT task must execute in the I_{SD}^k , i.e x_i^k,are computed through the following linear programming problem.

Let $cc_i^* = \omega_i * F^* - cc_i$ be the cycles that task τ_i can be idle. Thus, the total amount of cycles $(sd_k * F^*)$ in sd_k can be rewritten as $sd_k * F^* = q * \omega_i * F^* + r_i$, where $0 \leq r_i < \omega_i * F^*$, $q \in \mathcal{Z}$, and q represents the occurrences of a task at sd_k. If $r_i = 0$, it means that the deadline of τ_i is sd_k. Then the following LPP can be posed to compute $X_{HRT}^k = \{x_1^k, ..., x_i^k\}$, i.e the workload per scheduling interval k.

$$\begin{array}{ll} \min & \sum_{i=1}^{n} x_i^k \quad \forall k = 1, \ldots, \alpha \\ s.t & \\ & \forall k \;\; \sum_{i=1}^{n} x_i^k = m * |I_{SD}^k| * F^* \\ & \text{if } r_i = 0 \;\; \sum_{\gamma=1}^{k} x_i^\gamma = q * cc_i \\ & \text{if } r_i \neq 0 \;\; \sum_{\gamma=1}^{k} x_i^\gamma \geq -q * cc_i^* + max\{0, \sum_{\gamma=1}^{k} |I_{SD}^\gamma| * F^* - cc_i^*\} \\ & \forall i \quad x_i^k \leq |I_{SD}^\gamma| * F^* \end{array} \tag{14}$$

The first constraint implies that the CPU utilization is 100%. It is required since Φ^* indicates that CPU utilization is 100%. The second constraint guarantees that those tasks that must complete their execution in the current interval do actually end. The last constraint ensures deadline fulfillment.

The following proposition guarantees that if the former LLPs (Eq. 14) are orderly solved according to the $k-th$ interval, then the computed amount of time that each task must run per interval yields a feasible schedule.

Proposition 2 *Given a task set $\mathcal{T}$, where the task utilization at F^* is equal to the number of CPUs, then the solution of the linear programming problems in Eq. (14) is always integer. Moreover, if each task τ_i is executed exactly x_i^k cycles during the $k-th$ interval, then a feasible schedule is obtained.*

Proof Let $\mathcal{T}^k = \mathcal{T}_1^k \cup \mathcal{T}_2^k$, where $\mathcal{T}_1^k$ and $\mathcal{T}_2^k$ partition the task set. $\mathcal{T}_1^k = \{\tau_1, ..., \tau_v\}$ is the set of tasks that have their deadlines at sd_k and $\mathcal{T}_2^k = \{\tau_{v+1}, ..., \tau_n\} = \mathcal{T} - \mathcal{T}_1^k$. In the LPP 14, the last two constraints must be converted into equality equations. This is solved by adding slack variables h_i to each constraint. Then all the constraints are represented as $My = b$ where the vector of variables is composed of the workload and slack variables, i.e. $y = [x\ h]^T$. Notice that vector b is always integer. By construction, the restriction matrix M has the form:

$$M = \left[\begin{array}{c|c} L_{(v+1)\times n} & \varnothing \\ \hline Q_{(2n-v)\times n} & I_{(2n-v)} \end{array}\right] \tag{15}$$

where L has the form:

$$L_{v+1\times n} = \left[\begin{array}{cccc|ccc} 1 & 1 & \cdots & 1 & 1 & \cdots & 1 \\ \hline 1 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & 0 & \cdots & 0 \end{array}\right] \tag{16}$$

It is easy to see that the rank of L is $v+1$. Hence, the rank of M is $rank(M) = rank(L) + rank(I) = 2n+1$, i.e M is a full row rank matrix. In order to prove that the solution is always integer, we will demonstrate that the restriction matrix M is unimodular.

Since M is a full row rank matrix, then the determinant of every square submatrix (M_{si}) of order $2n+1$, obtained by removing columns, must be equal to 1, 0 or -1 to proof that M is unimodular Sierksma [2001]. When the columns are removed, the following three scenarios are possible.

1) If any of the first v columns is removed, M_{si} losses rank, because a row with only zero elements remains, hence the determinant is 0. 2) If the removed column $M(\bullet, j)$ contains a nonzero entry $M(i, j)$, where $j > v$, then the corresponding row $M(i, \bullet)$ has a nonzero element among the first v columns, then M_{si} losses rank since the resulting row is duplicated among the first v rows. Thus the determinant is 0. 3) When any other column not listed before is deleted, the resulting matrix always can be arranged as

$$M_{si} = \left[\begin{array}{c|c} A & \varnothing \\ \hline B & I \end{array}\right] \tag{17}$$

Then, according to theorem 3.2 in Sierksma [2001], A is always a totally unimodular matrix, thus $det(A) = 0, \pm 1$. Also the determinant for the identity

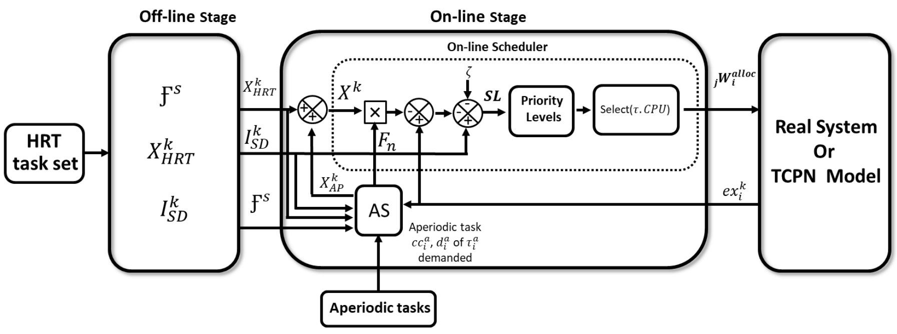

Fig. 2: Overall view of the Minimum Energy Thermal-Aware scheduler with aperiodic task management.

matrix is always 1; therefore, applying determinant per blocks, $det(M_{si}) = det(A) = 0, \pm 1$. □

6 On-line stage

The previous section described the off-line stage, that computes the CPU clock frequency F^*, the maximum clock frequency F^+ and the task execution time per deadline interval. Fig. 2 provides an overview of the two stages of the scheduler, the parts involved, and their linking signals. The process to build up all these parts and run simulations has been fully automated, as part of a publicly available simulation framework Desirena et al. [2019]. The package also includes other schedulers that can be simulated out-of-the-box. It provides tools to manually or automatically generate RT task sets for simulation, and to plot the results. This Section describes each of the components of the on-line stage and their relation to the whole system.

The inputs of the on-line stage are the deadline partition (i.e. the ordered set of deadline intervals (I^k_{SD}), the cycles that each task must execute per deadline interval (X^k_{HRT}), the set of actual feasible CPU frequencies ($\mathcal{F}^s$), the actual cycles that each job have executed since the last scheduling event (ex^k_i), and the aperiodic task parameters and arrivals. The output signal ${}_jW^{alloc}_i$ (Eq. 10), is the vector that determines the task allocation ratio of the allocation transitions ($t^{alloc}_{i,j}$) in the TCPN model (Eq. 5). If the scheduler is implemented on a real system instead of tested on a TCPN model of the system, it represents the vector of pairs (τ_i, CPU_j) determining the allocation of tasks to CPUs.

ALGORITHM 1: On-line Scheduler

```
Input:
$I^k_{SD}$ – Scheduling (deadline) intervals;
$X^k$ – Task cycles per deadline interval;
$ex^k_i$– Cycles actually executed for each task since the last scheduling event;
$F_n$ – CPU frequency
Output: A feasible schedule
1  Initialize $k = 0$
2  while true do
3      Compute the ordered set $SL$ of laxities;
4      if reach a new $I^k_{SD}$ then
5          $k = k + 1$ ;                          /* Update scheduling interval */
6          Compute task priorities using Priority Levels ;
7          Execute the $m$ tasks with higher priority;
8      else if reach a zero laxity then
9          Compute task priorities using Priority Levels ;
10         Execute the $m$ tasks with higher priority;
11     else if aperiodic task arrival then
12         call the Adaptive Scheduler
13     end
14 end
```

6.1 On-line scheduler

The on-line scheduler (Alg. 1) leverages a Fixed Priority Zero-Laxity (FPZL) algorithm (Davis and Burns [2011a]) to allocate tasks to processors until the next scheduling event. The inputs are the outputs of the off-line stage, and the runtime of each task accumulated since the previous scheduling event (ex^k_i, Fig. 2). A scheduling event occurs when a job reaches its zero laxity, a job ends, or an aperiodic task arrives. In the latter event, if the aperiodic task is accepted, the adaptive scheduler (AS) provides the cycles that the aperiodic task must run during the deadline interval ($x^k_{\tau_{a_i}}$), along with the adjusted CPU frequency (F_n).

Priority Levels.- Whenever an event occurs, task priorities are updated according to their laxity. Per-job laxities are calculated and ordered in a set $SL = \{l_i | l_i = sd_{k+1} - (F_n * x^k_i - ex^k_i) - \zeta\}$ (Alg. 1, step 3). Jobs reaching their zero-laxity time are given the maximum priority (= 1). Jobs being executed and with laxity different from zero receive priority equal to 2. The remaining jobs receive priority level equal to 3 (the lowest one). Thus, zero laxity tasks have the highest priority and must be executed immediately.

Execution of m tasks with the highest priority.- In Alg. 1, steps 7, 10, m tasks are dispatched to the m CPUs. In order to reduce the number of migrations, tasks that are executed during two consecutive events are allocated to the same CPU. In a system simulated by a TCPN, this step means to compute Eq. 5 according to ${}_jW^{alloc}_i$ (Eq. 10), in order to advance the simulation.

In a real system, the set of m tasks are just passed to the dispatcher of the operating system.

6.2 Preemptions and migrations bound

Job reemption is one of the causes of run time overhead and large memory requirements in RT scheduling. In this section we prove that the number of preemptions and migrations incurred by Alg. 1 is bounded.

Proposition 3 *Assuming that the conditions of Proposition 2 hold, then the context switches at each scheduling interval I^k_{SD} caused by Algorithm 1 have an upper bound given by*

$$2m + n_a \tag{18}$$

where n_a represents the number of active tasks in each I^k_{SD}.

Proof . Let X be the solution of the LPPs in Eq.(14), and $x^k_i \in X$. During the I^k_{SD} interval, task τ^k_i must run for x^k_i cycles at a given F_n clock frequency, but it can be the case that $x^k_i = 0$, hence the number of active tasks n^k_a in the I^k_{SD} interval can be less than n. Recall that a task can only leave a processor under two conditions: it has finished its execution or it is preempted by another task that reached zero laxity (ZL). On the best case scenario, all tasks assigned to the processors are already in ZL, therefore $n^k_a = m$. Under this condition each task is allocated to a processor, generating m context switches (CS) and another m CS at the end of their execution, thus, producing a total of $2m$ CS. Also, it is possible that the m tasks that were first allocated finished their execution before I^k_{SD}. Therefore, at most $n^k_a - m$ tasks will be allocated and incurring in $(n^k_a - m) + 2m$ CS. Finally, there are cases when some tasks reach ZL and others just finished their execution. Since the schedule is feasible there could only be m tasks reaching ZL at most, lest they miss their deadline. Hence the upper bound of CS is $(n^k_a - m) + 2m + m$, which is the same as $2m + n^k_a$. □

Proposition 4 *Assuming that Proposition 2 holds, Algorithm 1 causes at most $m - 1$ migrations of tasks.*

Proof . Recall that a task τ_i is preempted only when another task has reached zero laxity, hence when τ_i is preempted it will migrate because its original processor will become unavailable. Therefore at most there will be $m - 1$ migrations.

6.3 Aperiodic tasks and Adaptive Scheduler

Aperiodic tasks arrive asynchronously to the system. An *adaptive scheduler* (AS) determines if these tasks can be executed without compromising the HRT constraints of the periodic task set (Fig. 2, Alg. 2). If so, a new CPU clock frequency is computed allowing the execution of the aperiodic task. The computed frequency must be in the range $\mathcal{F}^s = \{F^* \dots F^+\}$, because from the off-line stage we know that these frequencies meet the thermal constraints. Moreover, the frequency must also be as low as possible, in order to guarantee a minimum power consumption while meeting the temporal constraints.

Upon an aperiodic arrival, the AS determines the current I^j_{SD} and the scheduling interval Γ at which τ^a_i has its deadline $(r^a_i + d^a_i)$. Recall that the scheduler is periodic on the hyperperiod, such that the scheduling interval I^Γ_{SD} is the one that contains the element $g = (r^a_i + d^a_i) \bmod H$.

Then the AS computes the required CPU cycles C_u for all active tasks from the current scheduling interval j to Γ as:

$$C_u = \sum_{i=1}^{|X^k|} (x^k_i - ex^k_i) + \sum_{\gamma=k+1}^{\Gamma+g} \sum_{i=1}^{|X^\gamma|} x_i^{\gamma \bmod \alpha} \tag{19}$$

where α is the number of scheduling interval and ex^k_i is the execution of active task i in CPU cycles, since the last scheduling event and $X^k = X^k_{HRT} \cup X^k_{ap}$ represents the set of CPU cycles that every active task must execute during the $k-th$ scheduling interval. Hence the first sum in Eq.(19) stands for the CPU cycles that the system still need to execute, while the second sum does the same for the subsequent scheduling intervals up to I^Γ_{SD}. With this information, the algorithm computes the maximum amount of CPU cycles (C_{free}) that the processors can spare when running at maximum frequency F^+,

$$C_{free} = m * d^a_i * F^+ - C_u \tag{20}$$

where m is the number of CPUs and d^a_i is the aperiodic task relative deadline. If C_{free} is greater than the cycles demanded by the aperiodic task (cc^a_i), the system is capable of serving τ^a_i, hence the AS determines the new operating frequency F_n as,

$$F_n = \min \left\{ F \in \mathcal{F}^s | F \geq \frac{C_u + cc^a_i}{m * d^a_i} \right\} \tag{21}$$

Once F_n has been determined, the algorithm calculates the number of cycles $x^k_{\tau^a_i}$ from τ^a_i that will be executed in each scheduling interval from I^j_{SD} to I^Γ_{SD},

ALGORITHM 2: Adaptive Scheduler (AS)

```
Input:
$X^k$ – Task cycles per deadline interval; $I^k_{SD}$ – Scheduling intervals;
$cc^a_i, d^a_i$– Aperiodic tasks parameters;
$\mathcal{F}^s = \{F^*, \ldots, F^+\}$–Clock frequencies;
$F_n$– CPUs operating frequency;
$ex^k_i$– Cycles executed for each task since the last scheduling event;
Output:
$F_n$ updated operating frequency,
$x^k_{\tau_{a_i}}$ cycles of the aperiodic task per scheduling interval;
1  if periodic task arrives then
2      Determine interval $I^{\Gamma}_{SD}$, where $\tau^a_i$ has its deadline
3      Compute required CPU cycles for active tasks during interval;   /* Eq. (19) */
4      Calculate free CPU cycles $C_{free}$ ;                           /* Eq.(20) */
5      if $C_{free} \geq cc^a_i$ then
6          Accept task $\tau^a_i$;
7          Determine new $F_n$;                                        /* Eq.(21) */
8          Calculate $x^k_{\tau^a_i}$ from current $I^k_{SD}$ to $I^{\Gamma}_{SD}$;          /* Eq. (22) */
9      else
10         Reject task;
11     end
12 end
13 if an aperiodic task finishes then
14     Discard the CPU cycles associated to the aperiodic task;
15     Recalculate the new frequency;
16 end
```

which is an iterative procedure:

$$
\begin{aligned}
&cc_r = cc^a_i \\
&x^k_{\tau^a_i} = \min\left\{ m(|I^k_{SD}| - r^a_i)F_n - \sum_{i=1}^{|X^k|}(x^k_i - ex^k_i),\ cc_r \right\} \\
&\textbf{For } \gamma = k+1 \textbf{ to } (\Gamma + g) \\
&\quad cc_r = cc_r - x^{(\gamma \bmod \alpha)-1}_{\tau^a_i} \\
&\quad x^{\gamma \bmod \alpha}_{\tau^a_i} = \min\left\{ m(|I^{\gamma \bmod \alpha}_{SD}|)F_n - \sum_{i=1}^{|X^{\gamma \bmod \alpha}|} x^{\gamma \bmod \alpha}_i,\ cc_r \right\}
\end{aligned}
\tag{22}
$$

6.4 Complexity

The complexity of the on-line stage depends on two algorithms. The priority level and the computation of laxity in Alg. 1 is linear in the number of tasks. At most $n = |\mathcal{T}|$ tasks will end its execution x^k_i in the $k-th$ interval (there are at most n tasks). Also, n tasks will reach their zero laxity at most. If q aperiodic tasks arrive in the $k-th$ interval, then the nested while loop ends in

$(n+n+q)\times(n+n)$ (*number of events* $\times$ *number of operations*). Considering that the outer loop runs $\alpha = |I_{SD}|$ times, then the number of steps of this algorithm is polynomial in the order of tasks. Alg. 2 runs on the arrival of an aperiodic task and is polynomial in the order of tasks and independent of the number of CPUs. Thus the proposed algorithm is polynomial in the order of tasks.

7 Experimental Results

In this Section we simulate the behavior of *EETAMS*, a scheduler implemented according to the on-line and off-line stages herein proposed. First, we present an example to study the thermal behavior and real utilization considering the HRT task set. Then, we proof the ability to deal with the arrival of an aperiodic task while maximizing CPU utilization, controlling temperature and optimizing energy consumption. Last, we compare *EETAMS* with *gEDF*, a non-fluid global RT scheduler, and *RT-TCPN*, a *pfair / deadline partitioning* hybrid algorithm. Both *gEDF* and *RT-TCPN* were introduced in Section 2.3. In the case of *gEDF* we have applied a straightforward implementation. The comparison focuses on temperature control, number of context switches and consumed energy.

7.1 Experimental environment

We assume a platform composed of two homogeneous Intel XScale silicon microprocessors mounted over a copper heat spreader for all the experiments. The isotropic thermal properties and dimensions of the materials are taken from Desirena-Lopez et al. [2014]. The power model for the Intel XScale is based on Chen and Kuo [2007]. The processor supports five operating frequency levels $\mathcal{F} = \{0.15,\ 0.4,\ 0.6,\ 0.8,\ 1\}$ GHz, consuming $P_{CPU} = \{80,\ 170,\ 400,\ 900,\ 1600\}$ mWatt respectively. Thus, the power consumption function can be modeled approximately as $P_{CPU} = 0.08 + 1.52 \cdot \phi^3$ Watt. The temperature of the surrounding air is constant and set to 45^o C. In the experiments we assume cache memories and speculative mechanisms non-existent or turned off.

We use an in-house simulation framework publicly available (Desirena et al. [2019]). As of today, this framework allows to generate automatically the TCPN model, the LPP equations and solutions, and to integrate other scheduling algorithms programmed in MATLAB [2018]. The schedulers used in this section are available out-of-the-box.

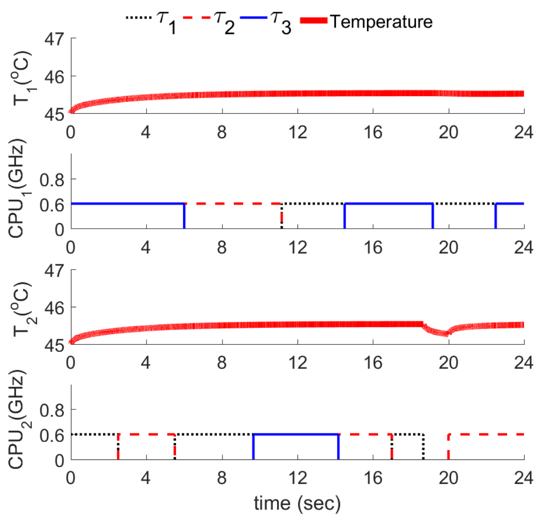

Fig. 3: Temperature evolution (upper plot) for the periodic schedule (lower plot) at CPU_1 (above) and CPU_2 (below). The maximum temperature produced by this schedule is $T_{CPU_{1,2}} = 45.3^oC$

.

7.2 Temperature control and utilization

In this first experiment, we study the thermal control capabilities and real utilization achieved by *EETAMS*. We consider a set of sporadic tasks with implicit deadlines $\mathcal{T} = \{\tau_1, \tau_2, \tau_3\}$, where $\tau_1 = (1.5e9, 4)$, $\tau_2 = (3e9, 8)$, $\tau_3 = (5e9, 12)$, the hyperperiod is $H = 24$. The maximum operating temperature level is set to $T_{max_{1,2}} = 50^o$ C. First, the minimum frequency for the periodic task set is computed off-line according to Eq. (8), obtaining $\Phi^* = 0.5833$. This frequency is raised to the nearest upper frequency actually available for the processor, which is $F^* = 0.6$ GHz. Eq. (11) provides the maximum clock frequency ($F^+ = 1$ GHz), so that the *METARTS* problem has a solution. We assume that scheduling and context-switch overheads are included in task WCET. Then, solving the LPP in Eq. (14) for F^* yields the CPU cycles of each task to be executed per interval (x_i^k).

Fig. 3 provides the schedule and temperature evolution produced by the algorithm without considering aperiodic tasks. The off-line LPP and the FPZL on-line scheduler are work-conserving and yield a theoretical 100% CPU utilization. However, the fact that $F^* > (\Phi^* \times F_{max})$ makes tasks allocated to CPU_2 to run faster in this simulation. This translates into the slack that appears in the lowest plot of the figure (interval $[18, 20]$), which temporarily lowers the temperature as shown in the corresponding temperature graph, and decreases the theoretical utilization by about 8%.

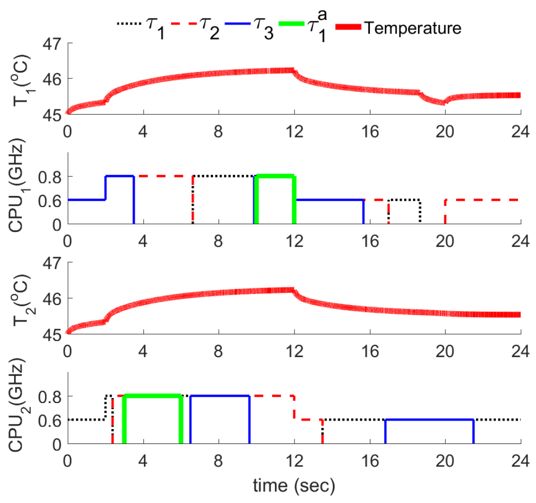

Fig. 4: Temperature evolution (upper plot) for the periodic schedule (lower plot) at CPU_1 (above) and CPU_2 (below) upon acceptance of the aperiodic task τ_1^a. The maximum temperature produced by this schedule is $T_{CPU_{1,2}} = 46.24^oC$

7.3 Handling aperiodic tasks

We now show the behavior of the *EETAMS* scheduler upon the arrival of an aperiodic task while running the same HRT task set considered in the previous experiment. Fig. 4 depicts the outcome when an aperiodic task $\tau_1^a = (4000, 10)$ arrives at $\zeta = 2$, during the I_{SD}^1 time slice. τ_1^a has an absolute deadline at $\zeta = 12$. Since $C_{free} \geq cc_1^a$, the *AS* accepts the aperiodic task, and computes $F_n = 0.8$ GHz $\in \mathcal{F}^s$ as the frequency at which the processors must execute during interval $[2, 12]$. The solid red line shows that temperature increases during this interval because of the execution of the tasks at frequency F_n, and then it decreases after $\zeta = 12$ because a new (lower) frequency has been calculated for the next interval. In both experiments, with and without the aperiodic tasks, CPU_1 achieves full utilization, whereas CPU_2 shows a slack (idle time) at about $\zeta = 18$, which translates into a temperature valley. As it happened in the previous experiment, this slack appears because the exact optimal frequency calculated in Eq. (8) is upper bounded by a frequency belonging to the discrete set of frequencies actually available in the microprocessor ($F_n \in \mathcal{F}^s$).

7.4 Comparison with *gEDF* and the *RT-TCPN pfair* scheduler

EETAMS is tied to the fluid scheduling concept. As we summarized in Section 2.3, this type of schedulers achieve 100% system utilization but yield a high overhead because of the many scheduling points, context switches and migrations. *EETAMS* is implemented with a *deadline partitioning* approach instead of a quantum-based one like *pfair* algorithms, which should lower the overhead. In this Section we compare *EETAMS* with *RT-TCPN*, which leverages a quantum calculated on a *deadline partitioning* basis, and with *gEDF*, a well-known scheduler unrelated to the fluid scheduling concept. We focus on temperature control and scheduling overhead. Since *gEDF* lacks HRT optimality, a comparison considering aperiodic tasks locks both *gEDF* and *RT-TCPN* at the maximum system clock frequency, which makes the comparison unfair. On the other hand, the fact that the real utilization achieved by *EETAMS* falls below 100% because of the rounding to an actual available system frequency, makes all the more interesting the comparison with *gEDF*.

Because *gEDF* is not HRT optimal, we need to assure that all the task sets are HRT schedulable under the three compared schedulers, for the comparison to be fair. Therefore, in this experiment we abide by the schedulability constraints of *gEDF*. An implicit-deadline HRT task set τ is schedulable under *gEDF* on m processors if $U_{tot}(\mathcal{T}) \leq m - (m-2) \cdot U_{max}(\mathcal{T})$, where $U_{tot}(\mathcal{T})$ ($U_{tot}(\mathcal{T}) = \sum_i u_i/m$) is the total utilization and $U_{max}(\mathcal{T})$ the maximum utilization of a task in $\mathcal{T}$ (Goossens and Funk [2003]). We consider $m = 2$ in our experiments, therefore to warrant the schedulability under *gEDF* $U_{max}(\mathcal{T})$ should be less than or equal to 0.5, with $U_{tot}(\mathcal{T}) \leq 1.5$ (75% of maximum utilization). Accordingly, we produced tasks with a utilization randomly generated under a uniform distribution by using the UUnifast algorithm (Bini and Buttazzo [2005]). We simulated ten task sets for each total utilization varying U_{tot} from 50% to 75% on the entire hyperperiod, working at maximum frequency.

Fig. 5 compares the performance of the three schedulers regarding context switches (y-axis) for our experimental range of system utilization (x-axis). We normalize context switches as *context switches per job*, calculated as the number of context switches divided by the number of jobs along the hyperperiod. The huge vertical gap between the *pfair*-like, quantum-based *RT-TCPN* and the other two schedulers requires the use of a logarithmic scale. Both, *EETAMS* and *gEDF* schedulers incurred in fewer context switches than the *RT-TCPN* scheduler. The number of context switches per job for the two fluid-based schedulers (*EETAMS* and *RT-TCPN*) decreases steadily as utilization increases, but we cannot fully observe the trend since we limit utilization to 1.5 (0.75%), in order for the comparison to be fair with *gEDF* as explained above.

The *percentage of maximum temperature* in Fig. 6 indicates the maximum value of temperature reached by the platform during the simulation of the computed schedule, as a percentage of the temperature bound. As a general rule, the greater the maximum temperature, the shorter the system lifespan. *gEDF* yields the highest temperature values. *EETAMS* obtains the lowest temperatures because it successfully finds the minimum frequency for the pe-

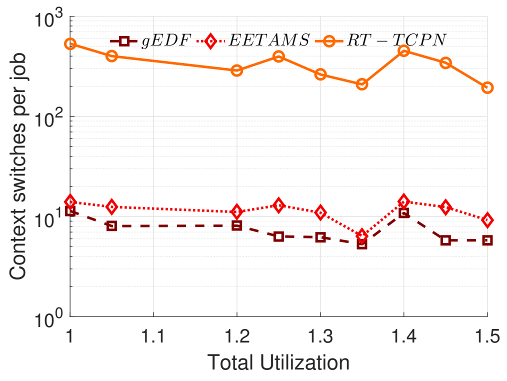

Fig. 5: Average amount of preemptions per job on 2 processors when the total utilization varies between 50% and 75%

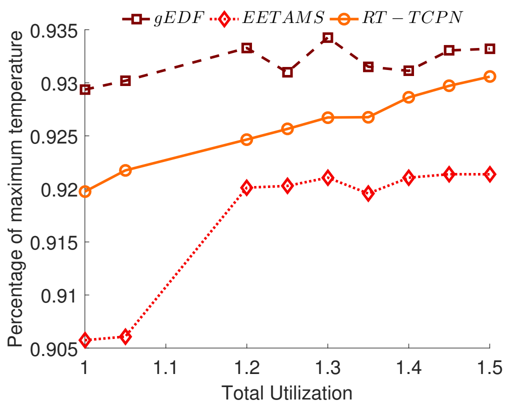

Fig. 6: Maximum temperature comparison for $m = 2$ and $U_{tot} = [1\ 1.5]$

riodic task set in order to achieve full system utilization. As expected, the percentage of maximum temperature tends to the same value when the total system utilization tends to 100%.

Fig. 7 indicates the energy consumed during the hyperperiod. The values correspond to the execution of the task sets with a power consumption $P_{CPU} = 0.08 + 1.52 \cdot \phi^3$. The scheduler *EETAMS* tends to consume less energy because it exploits DVFS, while *gEDF* and *RT-TCPN* always work at the maximum operating frequency.

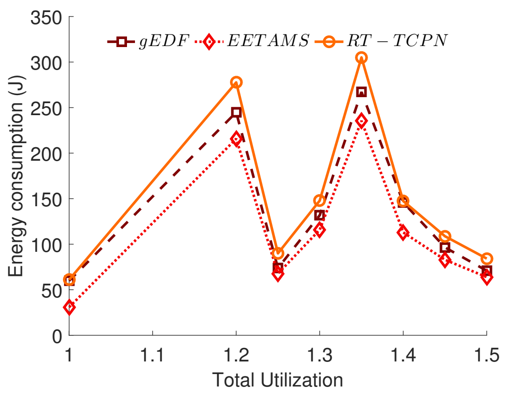

Fig. 7: Energy consumption.

8 Conclusions

This work shows that the TCPN formalism is a suitable tool for the design of thermal-aware RT schedulers, particularly when complexity rises because of further elements such as additional thermal constraints or aperiodic task management. Leveraging this formalism, we build a two-stage, energy-efficient thermal-aware scheduling system in which an HRT periodic task set executes at minimum clock frequency on a set of processors, with the ability to manage aperiodic tasks, optimizing power consumption, maximizing CPU utilization and honoring the HRT and thermal constraints in all cases. The TCPN models the activity of the tasks, their allocation to CPUs, heat generation and transfer, and how the latter affects to the overall system temperature. The thermal schedule feasibility is proved by an LPP that captures the RT and thermal restrictions as linear constraints. If there exists a feasible solution, then the LPP finds the maximum operating frequency F^+ to satisfy the thermal constraint.

This modeling methodology is automated by a software tool, integrated in a publicly available simulation framework which encompasses *EETAMS*, a simulated implementation of the scheduling system described in this paper, along with other schedulers equally available. We experimentally show that *EETAMS* achieves a successful thermal control while meeting the expected HRT constraints, maximizing CPU utilization, minimizing energy consumption and dealing with aperiodic tasks.

We compare *EETAMS* with *RT-TCPN*, which is a fluid scheduler implementation which uses a quantum, and with *gEDF*, a non-fluid scheduler with lower overhead but that lacks the HRT optimality of fluid schedulers. The comparison shows that *EETAMS* achieve superior thermal control and the lowest energy consumption while keeping context switching to a level comparable to *gEDF*.

Future work include feedback control to add robustness under disturbances, avoiding recalculations upon arrival of aperiodic tasks, further heuristics to minimize the number of migrations and context switches, and testing the proposed scheduler in a RT kernel.

Acknowledgement

This work has been supported by the *Ministerio de Ciencia, Innovación y Universidades* and the European ERDF under Grant TIN2016-76635-C2-1-R (AEI/ERDF, EU), and by the Aragon Government (T58_17R research group) and ERDF 2014-2020 *"Construyendo Europa desde Aragón"*.

References

Dong-Ik Oh and T. P. Bakker. Utilization bounds for n-processor rate monotone scheduling with static processor assignments. *Real-Time Systems*, 15 (2):183–192, 1998.

Nathan Fisher, Joël Goossens, and Sanjoy Baruah. Optimal online multiprocessor scheduling of sporadic real-time tasks is impossible. *Real-Time Syst.*, 45(1-2):26–71, June 2010.

Shelby Funk, Greg Levin, Caitlin Sadowski, Ian Pye, and Scott Brandt. Dp-fair: a unifying theory for optimal hard real-time multiprocessor scheduling. *Real-Time Systems*, 47(5):389–429, 2011.

S. Moulik, R. Devaraj, A. Sarkar, and A. Shaw. A deadline-partition oriented heterogeneous multi-core scheduler for periodic tasks. In *2017 18th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT)*, pages 204–210, 2017.

Björn B. Brandenburg and Mahircan Gül. Global scheduling not required: Simple, near-optimal multiprocessor real-time scheduling with semi-partitioned reservation. In *IEEE Real-Time Systems Symposium (RTSS 2016)*, pages 99–110, 2016.

Daniel Casini, Alessandro Biondi, and Giorgio Buttazzo. Semi-partitioned scheduling of dynamic real-time workload: A practical approach based on analysis-driven load balancing. In Marko Bertogna, editor, *29th Euromicro Conference on Real-Time Systems (ECRTS 2017)*, volume 76 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 13:1–13:23, 2017.

Robert I. Davis and Alan Burns. Fpzl schedulability analysis. In *Proceedings of the 2011 17th IEEE Real-Time and Embedded Technology and Applications Symposium*, RTAS '11, pages 245–256, Washington, DC, USA, 2011a. IEEE Computer Society. ISBN 978-0-7695-4344-4.

L. Rubio-Anguiano, G. Desirena-López, A. Ramírez-Treviño, and J.L. Briz. Energy-efficient thermal-aware scheduling for rt tasks using tcpn. *IFAC-PapersOnLine*, 51(7):236 – 242, 2018. doi: https://doi.org/10.1016/j.ifacol.2018.06.307. 14th IFAC Workshop on Discrete Event Systems WODES 2018.

Jian-Jia Chen and Tei-Wei Kuo. Procrastination determination for periodic real-time tasks in leakage-aware dynamic voltage scaling systems. In *Computer-Aided Design, 2007. ICCAD 2007. IEEE/ACM International Conference on*, pages 289–294, Nov 2007.

C. Mahulea, A. Ramirez-Trevino, L. Recalde, and M. Silva. Steady-state control reference and token conservation laws in continuous Petri net systems. *Automation Science and Engineering, IEEE Transactions on*, 5(2):307–320, April 2008. ISSN 1545-5955.

J. Desel and J. Esparza. *Free Choice Petri Nets.* Cambridge Tracts in Theoretical Computer Science 40, 1995.

M. Silva and L. Recalde. Redes de Petri continuas: Expresividad, análisis y control de una clase de sistemas lineales conmutados. *Revista Iberoamericana de Automática e informática Industrial*, 4(3):5–33, julio 2007. ISSN 1697-7912.

M. Silva, Jorge Júlvez, Cristian Mahulea, and C. Renato Vázquez. On fluidization of discrete event models: observation and control of continuous Petri nets. *Discrete Event Dynamic Systems*, 21(4)(3):427–497, December 2011.

Theodore P Baker. A comparison of global and partitioned edf schedulability tests for multiprocessors. In *In International Conf. on Real-Time and Network Systems.* Citeseer, 2005.

Robert I Davis and Alan Burns. A survey of hard real-time scheduling for multiprocessor systems. *ACM computing surveys (CSUR)*, 43(4):35, 2011b.

John M. Calandrino, James H. Anderson, and Dan P. Baumberger. A hybrid real-time scheduling approach for large-scale multicore platforms. In *Proceedings of the 19th Euromicro Conference on Real-Time Systems*, ECRTS '07, pages 247–258, Washington, DC, USA, 2007. IEEE Computer Society.

Marko Bertogna and Michele Cirinei. Response-time analysis for globally scheduled symmetric multiprocessor platforms. In *Real-Time Systems Symposium, 2007. RTSS 2007. 28th IEEE International*, pages 149–160. IEEE, 2007.

Joeè L Goossens and Shelby Funk. Priority-driven scheduling of periodic task systems on multiprocessors. *Real-Time Systems*, pages 2–3, 2003.

Marko Bertogna, Michele Cirinei, and Giuseppe Lipari. Improved schedulability analysis of edf on multiprocessor platforms. In *Proceedings of the 17th*

Euromicro Conference on Real-Time Systems, ECRTS '05, pages 209–218, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2400-1.

Sanjoy K Baruah, Neil K Cohen, C Greg Plaxton, and Donald A Varvel. Proportionate progress: A notion of fairness in resource allocation. *Algorithmica*, 15(6):600–625, 1996.

Sanjoy K Baruah, Johannes E Gehrke, and C Greg Plaxton. Fast scheduling of periodic tasks on multiple resources. In *ipps*, page 280. IEEE, 1995.

James H Anderson and Anand Srinivasan. Mixed pfair/erfair scheduling of asynchronous periodic tasks. In *Real-Time Systems, 13th Euromicro Conference on, 2001.*, pages 76–85. IEEE, 2001.

G. Desirena-Lopez, J. L. Briz, C. R. Vázquez, A. Ramírez-Treviño, and D. Gómez-Gutiérrez. On-line scheduling in multiprocessor systems based on continuous control using timed continuous petri nets. In *13th International Workshop on Discrete Event Systems*, pages 278–283, 2016.

Joonho Kong, Sung Woo Chung, and Kevin Skadron. Recent thermal management techniques for microprocessors. *ACM Computing Surveys*, 44(3): 13:1–13:42, 2014.

Pradeep M. Hettiarachchi, Nathen Fisher, Masud Ahmed, Le Yi Wang, Shinan Wang, and Weisong Shi. A design and analysis framework for thermal-resilent hard real-time systems. *Embedded Computing Systems, ACM Transactions on*, 13(5s):146:1–146:25, 2014.

Jian-Jia Chen, Chia-Mei Hung, and Tei-Wei Kuo. On the minimization fo the instantaneous temperature for periodic real-time tasks. In *13th IEEE Real Time and Embedded Technology and Applications Symposium (RTAS'07)*, pages 236–248. IEEE, 2007.

Lars Schor, Iuliana Bacivarov, Hoeseok Yang, and Lothar Thiele. Worst-case temperature guarantees for real-time applications on multi-core systems. In *2012 IEEE 18th Real Time and Embedded Technology and Applications Symposium*, pages 87–96. IEEE, 2012.

Rehan Ahmed, Parameswaran Ramanathan, and Kewal K Saluja. Necessary and sufficient conditions for thermal schedulability of periodic real-time tasks under fluid scheduling model. *ACM Transactions on Embedded Computing Systems (TECS)*, 15(3):49, 2016.

T. Chantem, X.S. Hu, and R.P. Dick. Temperature-aware scheduling and assignment for hard real-time applications on MPSoCs. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 19(10):1884–1897, Oct 2011. ISSN 1063-8210.

Yong Fu, Nicholas Kottenstette, Yingming Chen, Chenyang Lu, Xenofon D Koutsoukos, and Hongan Wang. Feedback thermal control for real-time systems. In *2010 16th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 111–120. IEEE, 2010.

James Donald and Margaret Martonosi. Techniques for multicore thermal management: Classification and new exploration. In *ACM SIGARCH Computer Architecture News*, volume 34, pages 78–88. IEEE Computer Society, 2006.

Francesco Zanini, David Atienza, and Giovanni De Micheli. A control theory approach for thermal balancing of mpsoc. In *2009 Asia and South Pacific Design Automation Conference*, pages 37–42. IEEE, 2009.

Xing Fu, Xiaorui Wang, and Eric Puster. Dynamic thermal and timeliness guarantees for distributed real-time embedded systems. In *2009 15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 403–412. IEEE, 2009.

Yong Fu, Nicholas Kottenstette, Chenyang Lu, and Xenofon D Koutsoukos. Feedback thermal control of real-time systems on multicore processors. In *Proceedings of the tenth ACM international conference on Embedded software*, pages 113–122. ACM, 2012.

ARINC. Specification 651: Design guide for integrated modular avionics, 1997.

N. Diniz and J. Rufino. Arinc 653 in space, 2005.

AUTOSAR. Specification of rte software, 2017.

Gabriel Fernandez, Javier Jalle, Jaume Abella, Eduardo Quiñones, Tullio Vardanega, and Francisco J. Cazorla. Computing safe contention bounds for multicore resources with round-robin and fifo arbitration. *IEEE Trans. Comput.*, 66(4):586–600, April 2017.

J. Cardona, C. Hernandez, E. Mezzetti, J. Abella, and F. J. Cazorla. Noco: Ilp-based worst-case contention estimation for mesh real-time manycores. In *2018 IEEE Real-Time Systems Symposium (RTSS)*, pages 265–276, Dec 2018.

G. Desirena-Lopez, A. Ramírez-Treviño, J. L. Briz, C. R. Vázquez, and D. Gómez-Gutiérrez. Thermal-aware real-time scheduling using timed continuous petri nets. *ACM Transactions on Embedded Computing systems. To appear, accepted Apr. 2019)*, 2019.

S. Baruah, M. Bertogna, and G. Butazzo. *Multiprocessor Scheduling for Real-Time Systems.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2015. ISBN 978-3-319-08695-8.

G. Desirena-Lopez, C. R. Vázquez, A. Ramírez-Treviño, and D. Gómez-Gutiérrez. Thermal modelling for temperature control in MPSoC's using fluid Petri nets. In *IEEE Conference on Control Applications part of Multi-conference on Systems and Control*, 2014.

Gerard Sierksma. *Linear and integer programming: theory and practice*. CRC Press, 2001.

G. Desirena, L. Rubio, A. Ramirez, and J.L. Briz. Thermal-aware hrt scheduling simulation framework, 2019. URL https://www.gdl.cinvestav.mx/art/uploads/TCPN-Thermal-Aware_Real-Time_Scheduling.zip.

MATLAB. *version 9.4 (R2018a)*. The MathWorks Inc., Natick, Massachusetts, 2018.

Enrico Bini and Giorgio C Buttazzo. Measuring the performance of schedulability tests. *Real-Time Systems*, 30(1-2):129–154, 2005.