Energy-Efficient Thermal-Aware Scheduling for RT Tasks Using $TCPN$

L. Rubio-Anguiano * **G. Desirena-López** *
A. Ramírez-Treviño * **J.L. Briz** **

* *CINVESTAV-IPN Unidad Guadalajara, Av. del Bosque 1145, CP 45019, Zapopan, Jalisco, Mexico*
** *DIIS/I3A Univ. de Zaragoza, María de Luna 1 - 50018 Zaragoza, España.*

Abstract:
This work leverages TCPNs to design an energy-efficient, thermal-aware real-time scheduler for a multiprocessor system that normally runs in a low state energy at maximum system utilization but its capable of increasing the clock frequency to serve aperiodic tasks, optimizing energy, and honoring temporal and thermal constraints. An off-line stage computes the minimum frequency required to run the periodic tasks at maximum CPU utilization, the proportion of each task's job to be run on each CPU, the maximum clock frequency that keeps temperature under a limit, and the available cycles (slack) with respect to the system with minimum frequency. Then, a Zero-Laxity online scheduler dispatches the periodic tasks according to the offline calculation. Upon the arrival of aperiodic tasks, it increases clock frequency in such a way that all periodic and aperiodic tasks are properly executed. Thermal and temporal requirements are always guaranteed, and energy consumption is minimized.

Keywords: Real-Time scheduling, Timed Continuous Petri Nets, Modelling.

1. INTRODUCTION

There is a great interest in using multicore processors in embedded real-time systems. Multicores reduce the Size, Weight and Power (SWaP) requirements in avionics, have evident advantages in automotive electronics and satellite systems, and are already common in consumer devices. Limited power, battery lifespan and thermal restrictions require careful energy management, for which real-time schedulers can leverage power management mechanisms provided by current MPSoCs, such as dynamic voltage and frequency scaling (DVFS). Many DVFS schedulers, thermal-aware schedulers and schedulers that minimize energy consumption have been proposed over the past few years (Kong et al. (2014), Schor et al. (2012), Ahmed et al. (2016), Hettiarachchi et al. (2014)).

In this work we present a minimal energy, thermal-aware real-time scheduler for a multiprocessor system. We consider a set of periodic independent tasks subject to temporal, thermal and energy restrictions, modeled by means of a Timed Continuos Petri Net (TCPN). Aperiodic tasks arrive asynchronously, and should only be served when every periodic task is guaranteed to meet its deadline. An off-line stage first computes the minimum frequency required to run the periodic task set at maximum CPU utilization. By means of a Linear Programming Problem (LPP) it finds a maximum frequency (F^+) at which the system can run subject to temporal and thermal constraints. The final off-line step leverages a deadline partitioning approach (Funk et al. (2011)) to compute a schedule by calculating the proportion of each task that must be executed on each CPU during the interval. In this way, the execution of this schedule consumes minimum energy and meets temporal and thermal constrain. In a second on-line stage, a Zero Laxity scheduler (Davis and Burns (2011b)) performs the schedule computed off-line for the periodic task set. Upon arrival, aperiodic tasks are serviced if a controller can increase the CPU clock frequency to achieve the proper slack, or are rejected otherwise. The frequency is always kept as low as possible to meet the temporal constraints and minimize energy consumption. As far as we know, this is the first solution based on TCPNs to accommodate aperiodic real-time tasks while preserving temporal and thermal constraints, minimizing energy. Our preliminary results, leveraging a deadline partitioning scheme, yield a good compromise in terms of optimal utilization of CPU, context switches and migrations.

Section 2 provides basic definitions and concepts on Timed Continuous Petri Nets ($TCPN$), the formal model used in this work to represent tasks, CPUs, energy consumption and temporal and thermal behavior. Section 3 formulates the scheduling problem. Section 4 explains the system model. The off-line schedule calculation is explained in Section 5, whereas Section 6 presents the the on-line scheduler. Section 7 shows the procedure to deal with aperiodic aperiodic tasks. Section 8 shows some examples, and Section 9 summarizes conclusions and further work.

2. PERI NETS BACKGROUND

This section provides basic definitions and concepts on Timed Continuous Petri Nets ($TCPN$), the formal model used in this work to represent tasks, CPUs, energy consumption, temporal and thermal behavior. For a deeper

insight on Petri Nets see Silva and Recalde (2007), David and Alla (2008), Silva et al. (2011).

Definition 2.1. A Petri net (PN) is a 4-tuple $N = (P, T, \mathbf{Pre}, \mathbf{Post})$ where $P = \{p_1, ..., p_{|P|}\}$ and $T = \{t_1, ..., t_{|T|}\}$ are finite disjoint sets of places and transitions. $\mathbf{Pre}$ and $\mathbf{Post}$ are $|P| \times |T|$ incidence matrices, where $\mathbf{Pre}[i,j] > 0$ (resp. $\mathbf{Post}[i,j] > 0$) if there is an arc going from p_i to t_j (or going from t_j to p_i), $\mathbf{Pre}[i,j] = 0$ (or $\mathbf{Post}[i,j] = 0$) otherwise.

Definition 2.2. A continuous Petri net ($ContPN$) is a pair $ContPN = (N, \mathbf{m}_0)$ where $N = (P, T, \mathbf{Pre}, \mathbf{Post})$ is a PN (PN) and $\mathbf{m}_0 \in \{\mathbb{R}^+ \cup 0\}^{|P|}$ is the initial marking.

A transition t_i is *enabled* at $\mathbf{m}$ if $\forall\, p_j \in {}^\bullet t_i, \mathbf{m}[p_j] > 0$; and its *enabling degree* is defined as $enab(t_i, \mathbf{m}) = \min_{p_j \in {}^\bullet t_i} \frac{\mathbf{m}[p_j]}{\mathbf{Pre}[p_j, t_i]}$. Firing t_i in a certain amount $\alpha \leq enab(t_i, \mathbf{m})$ yields a marking $\mathbf{m}' = \mathbf{m} + \alpha \mathbf{C}[P, t_i]$, where $\mathbf{C} = \mathbf{Post} - \mathbf{Pre}$. If $\mathbf{m}$ is reachable from $\mathbf{m}_0$ by firing the finite sequence σ of enabled transitions, then $\mathbf{m} = \mathbf{m}_0 + \mathbf{C}\vec{\sigma}$ is named the fundamental equation where $\vec{\sigma} \in \{\mathbb{R}^+ \cup 0\}^{|T|}$ is the firing count vector, i.e $\vec{\sigma}[j]$ is the cumulative amount of firings of t_j in the sequence σ.

Definition 2.3. A timed continuous PN ($TCPN$) is a time-driven continuous-state system described by the tuple $(N, \_, \mathbf{m}_0)$ where $(N, \mathbf{m}_0)$ is a continuous PN and the vector $\_ \in \{\mathbb{R}^+ \cup 0\}^{|T|}$ represents the transitions rates determining the temporal evolution of the system. Transitions fire according to a certain speed, which is generally a function of the transition rates and the current marking. Such function depends on the semantics associated to the transitions. Under infinite server semantics, the flow through a transition t (or t firing speed, denoted as $\mathbf{f}(\mathbf{m})$) is the product of the rate, λ_i, and $enab(t_i, \mathbf{m})$, the instantaneous enabling of the transition, i.e., $f_i(\mathbf{m}) = \lambda_i\, enab(t_i, \mathbf{m})$.

The firing rate matrix is defined by $\mathbf{\Lambda} = diag(\lambda_1, ..., \lambda_{|T|})$. For the flow to be well defined, every continuous transition must have at least one input place, so we 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 $\mathbf{m}$ is a set of (p,t) arcs describing the effective flow of each transition, and 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:

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

$\mathbf{f}(\mathbf{m}) = \mathbf{\Lambda\Pi}(\mathbf{m})\mathbf{m}$ is the vectorial form of the flow of a transition. The following fundamental equation describes the dynamic behaviour of a PN system:

$$\mathbf{m} = \mathbf{C}\,\mathbf{f}(\mathbf{m}) = \mathbf{C\Lambda\Pi}(\mathbf{m})\mathbf{m} \quad (2)$$

A control action can be applied to (2) by adding a term $\mathbf{u}$ to every transition t_i such that $0 \leq u_i \leq f_i$, indicating that its flow can be reduced. Thus, the controlled flow of transition t_i becomes $w_i = f_i - u_i$ and the forced state equation is: $\mathbf{m} = \mathbf{C}[\mathbf{f} - \mathbf{u}] = \mathbf{Cw}$.

3. PROBLEM DEFINITION

Definition 3.1. Let $\mathbf{T} = \{\tau_1, ..., \tau_n\}$ be a set of n independent periodic tasks. Each task is identified by the 3 *tuple* $\tau_i = (cc_i, d_i, \omega_i)$, where cc_i is the worst-case execution in CPU cycles, ω_i the period and d_i is the relative implicit deadline ($d_i = \omega_i$) (Baruah et al. (2015)). Let $\mathbf{P} = \{CPU_1, \ldots, CPU_m\}$ be a set of m identical processors with an homogeneous clock frequency $F \in \mathbf{F} = [F_1, ..., F_{max}]$. We assume that all task parameters, including task period and CPU cycles are integers and that any task can be preempted at any time. The hyper-period 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., $U = \sum_{i=1}^{n} \frac{c_i}{T_i}$. 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)).

Problem *3.1.* Minimum Energy Thermal Aware Real Time Scheduler ($METARTS$). Given the sets of tasks $\mathbf{T}$ and CPUs $\mathbf{P}$, the $METARTS$ problem consists in designing an algorithm to allocate within the hyperperiod H the tasks in $\mathbf{T}$ to the m identical CPUs in such a way that the deadlines for $\mathbf{T}$ are always satisfied and the CPU temperatures are kept always below a given temperature bound $\mathbf{T}_{max}$ and the consumed energy is minimum.

In addition to previous problem, aperiodic tasks, that must be allocated to CPUs arrive to the system.

Definition 3.2. **Aperiodic Task**. Let $\mathbf{T}^a = \{\tau_1^a, \ldots, \tau_z^a\}$ be an independent aperiodic task set. Each τ_i^a is defined as a 3-tuple (cc_i^a, d_i^a, r_i^a) where cc_i^a (required CPU cycles), d_i^a (its deadline) are known a priory, and r_i^a is the arrival time, which is unknown.

4. SYSTEM MODEL

The CPU thermal activity and the tasks fluid allocation to CPUs happens in a continuous space and honor a set of differential equations. The use of a TCPN allows modelling this continuous activity while providing data about the system's state at any moment. This section summarizes the TCPN model of the system (CPUs, tasks, thermal behavior, energy). For details on the thermal model we refer to Desirena-Lopez et al. (2014). The task model is based on Desirena-Lopez et al. (2016). Fig. 1 details a part of the model corresponding to a one task (τ_1) and one CPU (CPU_1).

Task and CPU Model.- The places p_i^I, p_i^{cc} and t_i^I, together with their arcs, model the i th task. The places $p_{1;j}^{busy}, ..., p_{n;j}^{busy}$, $p_{1;j}^{exec}, ..., p_{n;j}^{exec}$, p_j^{idle}, and transitions $t_{1;j}^{alloc}, ..., t_{n;j}^{alloc}$ and $t_{1;j}^{exec}, ..., t_{n;j}^{exec}$ together with their arcs, model the j th CPU. The weighted arc cc_i corresponds to the duration of τ_i (in CPU cycles). The CPU cycles required to run a task's job are stored in place p_i^{cc}. The transitions $t_{i;j}^{exec}$ represent the execution on the CPU_j. The current marking of places $p_{i;j}^{exec}$ ($m_{i;j}^{exec}$) represents the CPU worst-case execution time of task τ_i currently

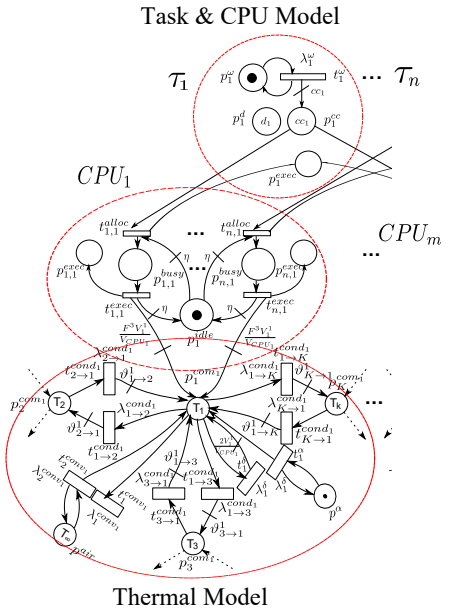

Fig. 1. Equivalent TCPN model for a multiprocessor system.

executed at a frequency F. Places p_j^{idle} and $p_{i;j}^{busy}$ represent the idle state and the busy state of the processor. 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 arcs going from transitions $t_{1;j}^{exec}, \ldots, t_{n;j}^{exec}$ to place p_j^{idle} and from place p_j^{idle} to transitions $t_{1;j}^{alloc}, \ldots, t_{n;j}^{alloc}$ are weighted by a constant value η, to ensure that the flow in transitions $t_{1;j}^{alloc}, \ldots, t_{n;j}^{alloc}$ is limited by the throughput CPU capacity modeled by place p_j^{idle}.

Thermal model.- The execution of cycles of τ_i in CPU_j is modeled by firing transitions $t_{i;j}^{exec}$ (Fig. 1), which adds tokens to places $p_i^{com_j}$, activating the thermal model for CPU_j (whose temperature will increase because of the activity). The rest of transitions and places in this thermal model represent heat transport by conduction and convection, for a deeper explanation see Desirena-Lopez et al. (2014).

Task and thermal model evolution.- The dynamic behavior of the global model (Fig. 1) is provided by the following equations:

$$\dot{\boldsymbol{m}}_T = \boldsymbol{C}_T \quad _T \quad _T(\boldsymbol{m})\boldsymbol{m}_T + \boldsymbol{C}_a \quad _a \quad _a(\boldsymbol{m})\boldsymbol{m}_a + \boldsymbol{C}_P^{exec}\boldsymbol{f}^{exec} \tag{3a}$$

$$\dot{\boldsymbol{m}}_a = 0 \tag{3b}$$

$$\dot{\boldsymbol{m}}_T = \boldsymbol{C}_T \quad _T \quad _T(\boldsymbol{m})\boldsymbol{m}_T \quad \boldsymbol{C}_T^{alloc}\boldsymbol{w}^{alloc} \tag{3c}$$

$$\dot{\boldsymbol{m}}_P = \boldsymbol{C}_P \quad _P \quad _P(\boldsymbol{m})\boldsymbol{m}_P + \boldsymbol{C}_P^{alloc}\boldsymbol{w}^{alloc} \tag{3d}$$

$$\dot{\boldsymbol{m}}_{exec} = \boldsymbol{C}_P^{exec}\boldsymbol{f}^{exec} \tag{3e}$$

$\boldsymbol{C}_x$, $\boldsymbol{\Lambda}_x$, and $\boldsymbol{\Pi}_x(\boldsymbol{m})$ are the incidence matrix, the firing rate transitions and the configuration matrix respectively ($x = fT, a, T, Pg$) of the thermal, task and processors subnet. $\boldsymbol{C}_T^{alloc}$, $\boldsymbol{C}_T^{alloc}$ and $\boldsymbol{C}_P^{alloc}$ stand for the connections of transitions $t_{i;j}^{alloc}$ from (to) places in the thermal model, task and CPU throughput model, respectively. Matrix $\boldsymbol{C}_P^{exec}$ holds the columns of the transitions $t_{i;j}^{exec}$ of the incidence matrix $\boldsymbol{C}_P$. $\boldsymbol{w}^{alloc}$ is the controlled flow of the *allocation transitions* (i.e. the task allocation rate to CPUs). Eq. (3a) represents the system's temperature evolution. Eq. (3b) indicates that the environmental temperature keeps constant at all times (its derivative is neglected). Eq. (3c) describes the arrival of periodic tasks to the system. Eq. (3d) models the CPUs cycles that are assigned to tasks. Finally, Eq. (3e) models task execution.

5. OFF-LINE STAGE

In our approach, prior to considering the aperiodic tasks, we need to find the minimum and maximum frequencies to execute the periodic task set subject to temporal and thermal constraints. This requires first to study the system thermal behavior. Later, we will compute a schedule applying a deadline partitioning approach.

Steady state Thermal Analysis.- Task execution generates a thermal activity given by Eq. (3a), where $\boldsymbol{C}_P^{exec} = F^3\boldsymbol{C}_P'^{exec}$. This can be rewritten as a space state equation: $\dot{\boldsymbol{m}}_T = \boldsymbol{A}\boldsymbol{m}_T + \boldsymbol{B}^0\boldsymbol{m}_a + F^3\boldsymbol{B}\boldsymbol{f}^{exec}$ and $\boldsymbol{Y}_T = \boldsymbol{S}\boldsymbol{m}_T$ (the CPUs temperature), where $\boldsymbol{A} = \boldsymbol{C}_T\boldsymbol{\Lambda}_T\boldsymbol{\Pi}_T(\boldsymbol{m})$, $\boldsymbol{B} = \boldsymbol{C}_P'^{exec}$ and $\boldsymbol{B}^0 = \boldsymbol{C}_a\boldsymbol{\Lambda}_a\boldsymbol{\Pi}_a(\boldsymbol{m})$. Since the schedule is periodic, the temperature is a non-decreasing function reaching a steady state temperature ($\boldsymbol{m}_{T_{ss}}$), i.e. $\dot{\boldsymbol{m}}_T = \boldsymbol{0}$ when time tends to infinite. Hence $\boldsymbol{m}_{T_{ss}} = \boldsymbol{A}^{-1}(F^3\boldsymbol{B}\boldsymbol{w}^{alloc} + \boldsymbol{B}^0\boldsymbol{m}_a)$.

Since $\boldsymbol{S}\boldsymbol{m}_{T_{ss}} \quad \boldsymbol{T}_{max}$ then:

$$\boldsymbol{S}\boldsymbol{A}^{-1}F^3\boldsymbol{B}\boldsymbol{w}^{alloc} \quad \boldsymbol{T}_{max} + \boldsymbol{S}\boldsymbol{A}^{-1}\boldsymbol{B}^0\boldsymbol{m}_a \tag{4}$$

This equation provides the thermal constraints that the allocation of tasks to the processors ($\boldsymbol{w}^{alloc}$) must fulfill.

Energy consumed by a schedule.- If the CPU_j clock frequency is F 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:

$$E_j = \int_{\zeta_1}^{\zeta_2} P_{CPU_j}(F)d \tag{5}$$

$P_{CPU_j}(F)$ is the power consumed by a CPU_j. It depends on $P_{dyn}(F)$, the dynamic power due to computational activities of tasks, and P_{leak}, the static power due to leakage. It is computed as: $P_{CPU_j}(F) = P_{dyn_j}(F) + P_{leak_j} = \lambda_{i;j}^{exec} m_{i;j}^{busy}(\zeta)F^3 + P_{leak_j}$. Where 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. The consumed energy is minimized *iff* the clock frequency F is minimized, but F must be high enough to ensure that temporal constraints are met. Next, we compute this frequency.

Minimum frequency Modern CPUs vary their clock frequency according to a number of preset values, i.e. $F = [F_1, ..., F_{max}]$. We normalize this set as $\phi = [\phi_{min} = \frac{F_1}{F_{max}}, ..., 1]$. The next proposition obtains the minimum clock frequency that fulfills temporal constraints.

Proposition 5.1. 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:

$$\Phi^* = \max\{\phi_{min}, \frac{1}{m} \sum_{i=1}^{n} \frac{cc_i}{\omega_i F_{max}}\} \quad (6)$$

Proof 5.1. According to Eq. (5), the energy has a minimum *iff* the consumer power is minimum. This occurs when ϕ^3 is minimum and fulfills that $\sum_{i=1}^{n} \frac{cc_i}{w_i F_{max}} = m$, and $\phi \geq \phi_{min}$. Using Lagrange multipliers, the Lagrangian function is $L = \phi^3 + \mu_1(\sum_{i=1}^{n} \frac{cc_i}{w_i F_{max}} - m) + \mu_2(\phi_{min} - \phi)$. The solution yields four cases: a) Both multipliers are inactive ($\mu_{1;2} = 0$); b) Both multipliers are active ($\mu_{1;2} > 0$); c) $\mu_1 = 0$ and $\mu_2 > 0$; and d) $\mu_1 > 0$ 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 minimize the total energy consumption becomes $\Phi^* = \max\{\phi_{min}, \frac{1}{m} \sum_{i=1}^{n} \frac{cc_i}{\omega_i F_{max}}\}$. □

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. (4). At frequency Φ^* the total system utilization is $U = \sum_{i=1}^{n} \frac{cc_i}{\omega_i F_{max}} = m$ and the processor frequency is $F^* = \min\{F \in F | F \geq \Phi^* F_{max}\}$, given the nature of the discrete set of frequencies. Since we have a fully utilized system, the distribution of the CPU cycles required to execute all tasks must be homogeneous, i.e., $\boldsymbol{w}^{alloc} = [\frac{1}{m} \sum_{i=1}^{n} \frac{cc_i}{\omega_i F}, \cdots, \frac{1}{m} \sum_{i=1}^{n} \frac{cc_i}{\omega_i F}]^T$. Moreover, if $\boldsymbol{w}^{alloc}$ satisfies Eq. (4), 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.

Maximum CPU cycles and clock frequency The *maximum thermal frequency* F^+ is the greatest frequency at which all CPUs operate at 100% of utilization and the temperature never exceeds the maximum thermal constraints. F^+ can be computed by using the next programming problem. The first constraint is thermal. 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^+ to the actual clock frequency range in the MPSoC.

$$\begin{aligned} &\max \quad F^+ \\ &s.t: \\ &\quad \boldsymbol{SA}^{-1}\boldsymbol{F}^{+3}\boldsymbol{B} \left[\frac{CC_1}{F^+H}, \cdots, \frac{CC_m}{F^+H}\right]^T \leq \boldsymbol{T_{max}} + \boldsymbol{SA}^{-1}\boldsymbol{B}^0\boldsymbol{m_a} \\ &\quad \frac{CC_j}{F^+H} = 1 \qquad \forall j = 1, \cdots, m \\ &\quad F^- \leq F^+ \leq F_{max} \end{aligned} \quad (7)$$

The solution for F^+ has to be in the set $\boldsymbol{F}$ of discrete frequencies. Thus the processor frequency is updated as $F^+ = \max\{F \in F | F \leq F^+\}$.

5.1 Deadline partitioning

We consider the ordered set of all tasks' jobs deadlines to define scheduling intervals, as in deadline partitioning (Funk et al. (2011)). Each task τ_i must be executed $n_i = \frac{H}{T_i}$ times within the hyperperiod H. Thus every $q \cdot \omega_i$, where $q = 1, ..., n_i$ is a deadline that must be considered in the analysis. These deadlines can be ordered and joined in the set $SD_i = \{sd_i^1, ..., sd_i^{n_i}\}$. A general set of deadlines is defined as $SD = SD_0 \cup ... \cup SD_{|\mathcal{T}|}$ where $SD_0 = \{0\}$. The elements of SD can be arranged in ascendant order and renamed as $SD = \{sd_0, ..., sd_\alpha\}$, where α is the last deadline. The scheduling interval $I_{SD}^k = [sd_{k-1}, sd_k]$ is defined and $|I_{SD}^k| = sd_k - sd_{k-1}$ represents the scheduling interval duration. The proposed deadline partitioning problems assumes a 100% utilization on every CPU, but recalling that $F^* = \min\{F \in F | F \geq \Phi^* F_{max}\}$, in most cases $F^* > \Phi^* F_{max}$ consequently idle cycles appear. To solve this problem we introduce a dummy task with period H and utilization $m - \sum_{i=1}^{n} \frac{c_i}{F \cdot T_i}$. Then the cycles that each task must execute in the I_{SD}^k, i.e x_i^k, can be computed as follows.

Let $cc_i^* = \omega_i \cdot F^* - cc_i$ be the cycles that task τ_i can be idle. Thus, the total amount of cycles ($sd_k \cdot F^*$) in sd_k can be rewritten as $sd_k \cdot F^* = q \cdot \omega_i \cdot F^* + r_i$, where $0 \leq r_i < \omega_i \cdot F^*$ and $q \in \mathcal{Z}$, where q represents the occurrences of a task (the task's jobs) in the system. If $r_i = 0$, it means that τ_i has its deadline in the scheduling interval. Then the following LPP can be posed to compute x_i^k.

$$\begin{aligned} &\min \quad \sum_{i=1} x_i^k \qquad \forall k = 1, \cdots, \\ &s.t \\ &\quad \forall k \quad \sum_{i=1} x_i^k = m \cdot |I_{SD}^k| \cdot F \\ &\quad \text{if } r_i = 0 \quad \sum_{=1} x_i = q \cdot cc_i \\ &\quad \text{if } r_i > 0 \quad \sum_{=1} x_i \geq q \cdot cc_i + \max\{0; \sum_{=1} |I_{SD}| \cdot F - cc_i\} \\ &\quad \forall i \quad x_i^k \leq |I_{SD}| \cdot F \end{aligned} \quad (8)$$

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 execution in this interval actually end. The last constraint guarantees deadline fulfillment. The following proposition guarantees that if the former LLPs 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 5.2. Given a task set T presented in Definition 3.1, where the task utilization at F^* is equal to the number of CPUs, the solution of the linear programming problems in Eq. (8) is always integer and if each task τ_i is executed exactly x_i^k cycles during the k th interval, then a feasible schedule is obtained.

Proof 5.2. Let $T^k = T_1^k \ [\ T_2^k$ be a task set partition, where $T_1^k = f\tau_1, ..., \tau_v g$ is the set of tasks that have their deadlines at sd_k and $T_2^k = f\tau_{v+1}, ..., \tau_n g = T$ T_1^k. Some slack variables h_i are added to form a set of constraints that can be represented as $My = b$ where $y = [x \ h]^T$ (i.e., equality constraints are obtained). Notice that vector b is always integer. By construction, the restriction matrix M has the form: $M = \begin{array}{c|c} L_{(v+1) \ n} & \varnothing \\ \hline Q_{(2n \ v) \ n} & I_{(2n \ v)} \end{array}$, and L has the form: $L_{(v+1)\times n} = \begin{array}{cc} 1 & 1 \\ \hline I_v & v \ j \ \varnothing \end{array}$. It is easily seen that $rank(L) = v + 1$ and $rank(M) = rank(L) + rank(I) = 2n + 1$, i.e M is a full row rank. The solution is always integer if M is unimodular, i.e., the determinant of every square submatrix (M_{si}) of M is either 0,+1 or -1. All M_{si} are obtained deleting columns, and there are three possible scenarios: first, if any of the first v columns is removed, M_{si} loses rank, hence $det(M_{si}) = 0$. Second, if any deleted column contains a nonzero entrie where its corresponding row has a nonzero element among the first v columns, M_{si} loses rank since the resulting row is duplicated among the first v rows. Thus, $det(M_{si}) = 0$. Finally, when any other column not listed before is deleted, the resulting matrix always can be arranged as $M_{si} = \begin{array}{c|c} A & \varnothing \\ \hline B & I \end{array}$, where matrix A is always TUM, according to **Theorem 3.4** reported in Sierksma (2001), thus $det(A) = 0, \ 1$. Therefore, $det(M_{si}) = det(A) \ det(I) = 0, \ 1$. 2

6. ON-LINE STAGE: SCHEDULER

The previous section computed the CPU clock frequency F^*, maximum clock frequency F^+ and task execution time per scheduling interval. The on-line scheduler uses these data to implement a Fixed Priority Zero-Laxity (FPZL) algorithm (Davis and Burns (2011a)). It allocates tasks' jobs during their respective scheduling interval upon the occurrence of three possible events: a zero-laxity event (a job must immediately execute lest it misses its deadline), job completion or the arrival of an aperiodic task. During the I_{SD}^k interval task τ_i^k must execute x_i^k cycles at a given F_n clock frequency.

Priority Levels Whenever an event occurs, the task priority is updated as follows. 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.

Algorithm 1 On-line schedule

1: **Input** I_{SD}^k { Schedulinng intervals; X^k { tasks CPU cycles per interval; ex_i^k { execution CPU cycles per interval
2: **Output** A feasible schedule;
$k = 0, \quad = 0$
3: Compute the ordered set of laxities as:
$SL = fl_i jl_i = sd_{k+1} \quad (F_n \quad x_i^k \quad ex_i^k) \quad g$
4: **while** H **do**
5: **while** sd_{k+1} **do**
6: Compute task priorities using *Priority Levels*
7: Execute the m tasks with higher priority until an event occurs (An event occurs if a task reaches its zero laxity, task ends or aperiodic task arrives.)
8: Compute the ordered set of laxities as:
$SL = fl_i jl_i = sd_{k+1} \quad (F_n \quad x_i^k \quad exec_i^k) \quad g$
9: $= \ +$ current time
10: **end while**
11: $k = k + 1$
12: **end while**

Execution of m tasks with the highest priority In Alg. 1 step 8, m tasks must be executed (i.e. allocated to a CPU). In order to reduce the number of migrations, tasks that are executed during two consecutive events are allocated to the same CPU.

7. APERIODIC TASKS

Aperiodic tasks arrive asynchronously to the system. The system determines if these tasks can be executed without compromising the hard real-time constraints of the periodic task set. If so, a new CPU clock frequency is computed to allow the execution of the aperiodic task. This frequency must be in the range $F^s = fF^* \dots F^+ g$ (every frequency in this range meets the thermal constraints), and it is kept as low as possible to guarantee a minimum energy consumption while meeting the temporal constraints.

Fig. 2 shows the scheme to schedule aperiodic tasks without compromising the hard real-time constraints of the periodic task set. The off-line stage computes the optimal and maximum allowed frequencies F^*, F^+, the scheduling intervals I_{SD}^k and the periodic task CPU cycles that must be executed per scheduling interval $X = fX^1, ..., X \ g$, where $X^k = fx_1^k, \dots, x_n^k g$ represents the set of CPU cycles x_i^k that must be executed during the k th scheduling interval of task τ_i. Job execution is tracked during the k th interval by the system and passed as the input to the *adaptive scheduler* (AS). AS is activated at the arrival or ending time of an aperiodic task τ_i^a. At this time, the task's CPU cycles cc_i^a and its relative deadline d_i^a are sent to AS. This information together with the outputs of the off-line stage are used to compute the new frequency and the CPU cycles $(x^k{}_a)$ required to execute τ_i^a per scheduling interval, such that $X^k = X^k \ [\ fx^k{}_a g$. When τ_i^a finishes its execution, the frequency is recalculated and the CPU cycles demanded by τ_i^a are discarded.

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 $n = jT j$ tasks will end its execution x_i^k in the k th interval (there at most n tasks). Also at most n tasks will reach their zero laxity. If q aperiodic tasks arrive in the k th interval, then the nested while loop ends in $(n + n + q)$

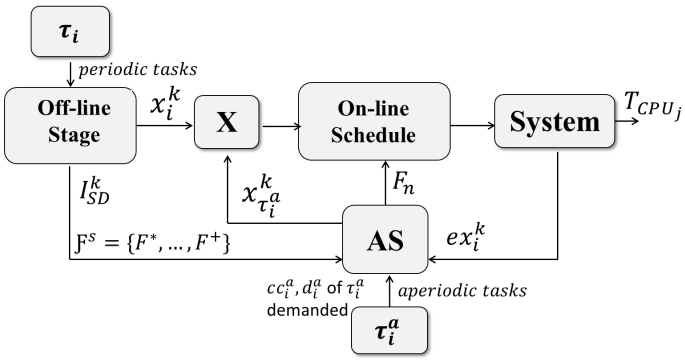

Fig. 2. Minimum Energy Thermal Aware scheme for periodic and aperiodic tasks.

$(n+n)$ $(number\ of\ events \quad number\ of\ operations)$. Considering that the outer loop runs $\alpha = jI_{SD}j$ 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.

Algorithm 2 Adaptive Scheduler (AS)

Input $cc_i^a; d_i^a\{$ Aperiodic tasks parameters; $ex_i^k\{$ Cycles executed in the system for all active tasks.
Output New Frequency F_n 2 $F^s = fF\ ;:::;F^+g$, task CPU cycles per I_{SD}^k, $x^k{}_{a_i}$ Per-interval CPU cycles for execution of the aperiodic task.
Initialize $n = jTj$, $m = jPj$, $F_n = F$, q the aperiodic tasks currently being attended.
BEGIN
if aperiodic task arrives **then**
  Let r_i^a = current time when $_i^a$ arrives;
  Compute required CPU cycles for active tasks from r_i^a to $r_i^a + d_i^a$;
  $C_u = \sum_{i=1}^{jPj} (x_i^k \quad ex_i^k) + \sum_{=k+1}^{} \sum_{i=1}^{jPj} x_i$;
  where k is the current scheduling interval at r_i^a and and is the scheduling interval at $r_i^a + d_i^a$;
  $C_{free} = m \ d_i^a \ F^+ \ C_u$; the free CPU cycles in the interval $[r_i^a; r_i^a + d_i^a]$;
  if $C_{free} \quad cc_i^a$ **then**
    Accept task $_i^a$;
    F_n = minimum F 2 F^s such that $F \quad \frac{C_u + cc_i^a}{m \ d_i^a}$;
    $cc_r = cc_i^a$; ( )
    $x^k{}_{a_i} = \min \quad m(jI_{SD}^k j \quad r_i^a)F_n \quad \sum_{i=1}^{jPj} (x_i^k \quad ex_i^k);\ cc_r$;
    in the k th interval;
    for $= k + 1$ **to** **do** - in other intervals
      $cc_r = cc_r \quad x_{\ i}^{\ a\ 1}$;
      $x_{\ i}^{\ a} = \min \quad m(jI_{SD}j)F_n \quad \sum_{i=1}^{jPj} x_i;\ cc_r$;
    end for
  else
    Reject task;
  end if
end if
if an aperiodic task nishes **then**
  Discard the CPU cycles associated to the aperodic task
  Recalculate the new frequency;
end if
END

8. SIMULATIONS RESULTS

In order to show how to use the proposed scheme, a proof of concept is presented. It consists of a set of sporadic periodic tasks $T = f\tau_1, \tau_2, \tau_3 g$, where $\tau_1 = (2000, 4)$, $\tau_2 = (5000, 8)$, $\tau_3 = (6000, 12)$, the hyperperiod is $H = 24$. These tasks run on two homogeneous microprocessors where the isotropic thermal properties and dimensions of the materials are taken from Desirena-Lopez et al. (2014). The processor supports four operating frequency levels $F = f0.5, 0.85, 0.95, 1gKHz$. The temperature of the surrounding air is set to 45^oC and it is constant. The maximum operating temperature level is set to $T_{max_{1,2}} = 50^oC$. The simulations herein presented consider CPUs with caches and speculative mechanisms non-existent or turned off. First, the minimum frequency for the periodic task set is off-line computed according to Eq. (6), obtaining $\Phi^* = 0.8125$, hence the selected frequency is $F^* = 0.85kHz$. Eq. (7) provides the maximum clock frequency ($F^+ = 1kHz$), so that the $METARTS$ problem has a solution. We assume that shceduling and context switch overheads are included in the tasks' $WCET$. Then, solving the LPP in Eq. (8) for F^* yields the CPU cycles of each task to be executed at each interval (x_i^k). Fig. 3 provides the schedule and temperature evolution produced by the algorithm without considering aperiodic tasks. Fig. 4 depicts the outcome of the algorithm and the evolution of the temperature when an aperiodic task $\tau_1^a = (2000, 10)$ arrives at $\zeta = 2$, during the I_{SD}^1 interval, thus τ_1^a has an absolute deadline at $\zeta = 12$. Since $C_{free} \quad cc_1^a$, the AS accepts the aperiodic task, and computes $F_n = 950kHz$ 2 F^s as the frequency at which the processors must execute during interval $[2, 12]$. Fig. 4 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. This slack appears because the exact optimal frequency calculated in Eq. (6) is ceiled to a frequency belonging the discrete set of frequencies available in the microprocessor $(F_n$ 2 $F^s)$.

9. CONCLUSIONS

This work shows that the TCPN formalism is a suitable way to model real-time task scheduling problems considering thermal, temporal and energy restrictions. Upon a TCPN model, we build a two-stage thermal-aware real-time system in which a periodic task set executes at minimum clock frequency to save energy and achieve maximum CPU utilization while honoring thermal constraints. Aperiodic tasks are also dynamically accepted as long as increasing the clock frequency allows to obtain a suitable slack. Time and thermal constraints are preserved in all cases. We have assumed that the aperiodic task (τ_i^a) fits in the slack inside the hyperperiod $(r_i^a + d_i^a \quad H)$. Immediate further work includes to relax that condition, and to improve our slack reclaiming approach to decrease context switching when accepting aperiodic tasks. Also, the fact that our underlying TCPN model allows modeling