Real time scheduler for multiprocessor systems based on continuous control using Timed Continuous Petri Nets [⋆]

L. Rubio-Anguiano [*] **A. Ramírez-Treviño** [*] **A. Chils** [**] **J.L. Briz** [**]

[*] *CINVESTAV-IPN Unidad Guadalajara, Av. del Bosque 1145, CP 45019, Zapopan, Jalisco, Mexico (e-mail: {le.rubio, art}@gdl.cinvestav.mx).*
[**] *DIIS/I3A Univ. de Zaragoza, María de Luna 1 - 50018 Zaragoza, España (e-mail: {718997, briz}@unizar.es).*

Abstract: This work exploits Timed Continuous Petri Nets (TCPN) to design and test a novel energy-efficient thermal-aware real-time global scheduler for a hard real-time (HRT) task set running on a multiprocessor system. The TCPN model encompasses both the system and task set, including thermal features. In previous work we calculated the share of each task that must be executed per time interval by solving off-line an Integer Programming Problem Problem (ILP). A subsequent on-line stage allocated jobs to processors. We now perform the allocation off-line too, including an allocation controller and an execution controller in the on-line stage. This adds robustness by ensuring that actual task allocation and execution honor the safe schedule provided off-line. Last, the on-line controllers allow the design of an improved soft RT aperiodic task manager. Also, ee experimentally prove that our scheduler yields fewer context switches and migrations on the HRT task set than RUN, a reference algorithm.

Keywords: Real Time, Scheduling, Petri Net, Linear Programming Problem, Unimodularity

1. INTRODUCTION

Multiprocessor Systems on a Chip (MPSoCs) are increasingly common in embedded real-time (RT) systems because of their potential to reduce the SWaP factor (Space, Weight and Power). However, they pose new scheduling challenges. Thus, a naive CPU task allocation could generate hot spots, reducing the MPSoC lifespan. Also, mobile and IoT devices demand careful energy management to extend battery life. Hence, new multicore RT schedulers must consider temperature and energy besides the timing constraints. To control task execution time and energy consumption, a scheduler can leverage the power management mechanisms provided by most current MPSoCs, such as dynamic voltage and frequency scaling (DVFS).

In Desirena-Lopez et al. (2019b) we tackled the thermal-aware RT global scheduling problem considering two stages, one to compute the workload, the other one to allocate tasks to CPUs. A fluid approach avoided the NP-completeness in the computation of the task workload per CPU, requiring a subsequent discretization algorithm. The fluid approach improved system's resilience to parameter variations, but led to a high number of context switches which hampered the feasibility of the scheduler. Authors in Thammawichai and Kerrigan (2018) propose an energy-efficient scheduler for two heterogeneous processors. They formulate a general case as a non-linear integer programming problem to obtain a schedule, which yields algorithms with high computational complexity. In Rubio-Anguiano et al. (2019) we proved that using a *DP-fair* (Funk et al. (2011)) scheme, the task workload can be solved in polynomial time, avoiding the discretization stage and reducing the number of context switches. Then, we resorted to a zero-laxity (ZL) policy (Davis and Burns (2011)) to allocate tasks to CPUs on-line. Nevertheless, this later approach solved the workload allocation through an integer linear programming problem (ILP) that ignored the relations between consecutive *scheduling intervals*, thus reducing the class of problems that could be solved.

Herein, the proposed scheme consist of three main components. The first component is an off-line scheduler similar to the one reported in Rubio-Anguiano et al. (2019). In this new proposal, the ILP is endowed with new constraints enlarging the class of problems that can be solved. The second component implements an on-line controller to add robustness to the scheduler by rejecting disturbances such as CPU detentions, or time drifting due to latencies or other parameters obviated in the model. Finally, the third component adds the capability of scheduling soft real-time (SRT) aperiodic tasks.

2. BACKGROUND ON PETRI NETS

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 con-

[⋆] 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"*.

sumption, 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 structure (PN) is a 4-tuple $N = (P, T, \boldsymbol{Pre}, \boldsymbol{Post})$ where $P = \{p_1, ..., p_{|P|}\}$ and $T = \{t_1, ..., t_{|T|}\}$ are finite disjoint sets of places and transitions. $\boldsymbol{Pre}$ 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 p_i to t_j (or going from t_j to p_i), $\boldsymbol{Pre}[i,j] = 0$ (or $\boldsymbol{Post}[i,j] = 0$) otherwise.

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

A transition t_i is *enabled* at $\boldsymbol{m}$ if $\forall\ p_j \in^{\bullet} t_i, \boldsymbol{m}[p_j] > 0$; and its *enabling degree* is defined as $enab(t_i, \boldsymbol{m}) = \min\limits_{p_j \in^{\bullet} t_i} \frac{\boldsymbol{m}[p_j]}{\boldsymbol{Pre}[p_j, t_i]}$. Firing t_i in a certain amount $\alpha \leq enab(t_i, \boldsymbol{m})$ yields a new marking $\boldsymbol{m'} = \boldsymbol{m} + \alpha \boldsymbol{C}[P, t_i]$, where $\boldsymbol{C} = \boldsymbol{Post} - \boldsymbol{Pre}$.

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

Definition 2.3. A timed continuous PN ($TCPN$) 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. Under infinite server semantics, the flow through a transition t (or t firing speed, denoted as $\boldsymbol{f(m)}$) is the product of the rate, λ_i, and $enab(t_i, \boldsymbol{m})$, the instantaneous enabling of the transition, i.e., $f_i(\boldsymbol{m}) = \lambda_i\, enab(t_i, \boldsymbol{m})$.

The firing rate matrix is defined 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, 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 $\boldsymbol{m}$ 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:

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

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

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

A control action can be applied to (2) by adding a term $\boldsymbol{u}$ 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: $\dot{\boldsymbol{m}} = \boldsymbol{C}[\boldsymbol{f} - \boldsymbol{u}] = \boldsymbol{Cw}$.

3. PROBLEM DEFINITION

Definition 3.1. Let $\mathcal{T} = \{\tau_1, ..., \tau_n\}$ be a set of n independent periodic tasks under hard real-time (HRT) constraints. 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 in processor cycles (WCET) that takes to complete any instance of the task (job), ω_i is the period and d_i is the relative implicit deadline ($d_i = \omega_i$) (Baruah et al. (2015)). Let $\mathcal{P} = \{P_1, \ldots, P_m\}$ be a set of m identical processors with an homogeneous clock frequency $F \in [F_1, F_{max}]$.

We assume that all task parameters, including task period and processor 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 P_i at its maximum frequency F_{max}, requires $c_i = \frac{cc_i}{F_{max}}$ processor time at every ω_i interval. Thus, the system utilization is defined as $U = \sum_{i=1}^{n} \frac{c_i}{\omega_i}$, i.e. the fraction of time during which the processors are busy running the task, which should be less or equal to the number of processors ($U \leq m$) (Baruah et al. (1996)). As in deadline partitioning approaches (Funk et al. (2011)), we consider the ordered set of all task deadlines to define scheduling intervals. In this context we define the *workload* as the amount of processor cycles that a task τ must execute during a given time interval. We also consider the arrival of asynchronous aperiodic tasks that need to be executed on the system.

Definition 3.2. Let $\mathcal{T}_a = \{\tau_1^a, ..., \tau_p^a\}$ be a set of p independent aperiodic tasks. Each task is identified by the $3-tuple$ $\tau_i^a = (cc_i^a, d_i^a, r_i^a)$, in which cc_i^a (WCET) and d_i^a (deadline) are known, but the arrival time r_i^a is unknown.

Formally, the problem addressed in this work is stated as:

Problem *3.1.* Control RT Scheduler (CRTS). Given the system defined in Def.3.1, the CRTS problem consists in designing a control law that tracks a set of references that successfully allocate the tasks in $\mathcal{T}$ to the m identical processors within the hyperperiod H, such that: the deadlines for $\mathcal{T}$ are always satisfied, the temperatures of the processors are kept below a bound T_{max} and the consumed energy is minimum. Additionally, the controller should execute or reject aperiodic tasks from Def.3.2 upon arrival, subject to the temporal and thermal constraints from the HRT tasks.

3.1 System model

This subsection summarizes the TCPN model that captures under a single formalism the behaviour of processors, arrival of tasks, their thermal activity and the power consumption. Leveraging a TCPN allows the use of differential equations as an aid for control design. The complete description of the model appears in Desirena-Lopez et al. (2019b). Fig. 1a details a section of the model corresponding to a system with n tasks (τ_i) and one processor (P_j).

The dynamic behavior of the global model (Fig. 1) is provided by the following equations:

$$\dot{\boldsymbol{m}}_{\mathcal{T}} = \boldsymbol{C}_{\mathcal{T}}\boldsymbol{\Lambda}_{\mathcal{T}}\boldsymbol{\Pi}_{\mathcal{T}}(\boldsymbol{m})\boldsymbol{m}_{\mathcal{T}} + \boldsymbol{C}_{\mathcal{T}}^{\boldsymbol{alloc}}\boldsymbol{w}^{\boldsymbol{alloc}} \tag{3a}$$

$$\dot{\boldsymbol{m}}_{\mathcal{P}} = \boldsymbol{C}_{\mathcal{P}}\boldsymbol{\Lambda}_{\mathcal{P}}\boldsymbol{\Pi}_{\mathcal{P}}(\boldsymbol{m})\boldsymbol{m}_{\mathcal{P}} + \boldsymbol{C}_{\mathcal{P}}^{\boldsymbol{alloc}}\boldsymbol{w}^{\boldsymbol{alloc}} \tag{3b}$$

$$\dot{\boldsymbol{m}}_{\boldsymbol{exec}} = \boldsymbol{C}_{\mathcal{P}}^{\boldsymbol{exec}}\boldsymbol{f}^{\boldsymbol{exec}} \tag{3c}$$

$$\begin{aligned}\dot{\boldsymbol{m}}_{\boldsymbol{T}} = {} & \boldsymbol{C}_{\boldsymbol{T}}\boldsymbol{\Lambda}_{\boldsymbol{T}}\boldsymbol{\Pi}_{\boldsymbol{T}}(\boldsymbol{m})\boldsymbol{m}_{\boldsymbol{T}} + \boldsymbol{C}_{\boldsymbol{a}}\boldsymbol{\Lambda}_{\boldsymbol{a}}\boldsymbol{\Pi}_{\boldsymbol{a}}(\boldsymbol{m})\boldsymbol{m}_{\boldsymbol{a}} \\ & + \boldsymbol{C}_{\mathcal{P}}^{\boldsymbol{exec}}\boldsymbol{f}^{\boldsymbol{exec}}\end{aligned} \tag{3d}$$

$$\dot{\boldsymbol{m}}_{\boldsymbol{a}} = 0 \tag{3e}$$

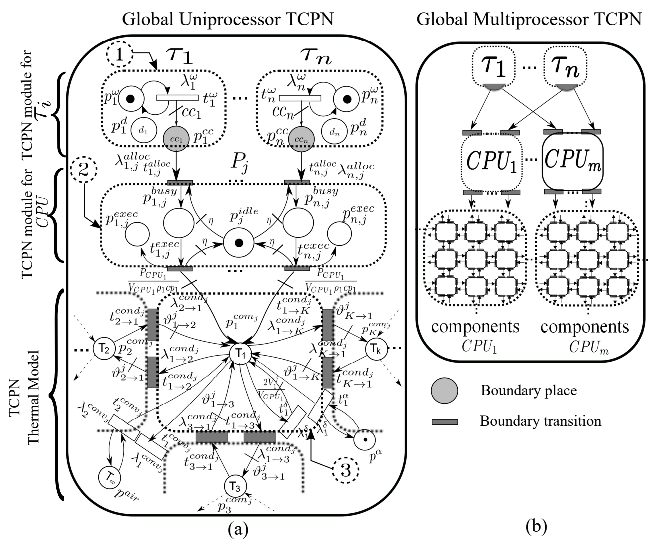

Fig. 1. TCPN model integrating task ①, processor ② and thermal modeling ③. (a) details the case for a single processor, and (b) zooms out and extends the model for a collocation.

where $\boldsymbol{C_x}$, $\boldsymbol{\Lambda_x}$, and $\boldsymbol{\Pi_x(m)}$ are the incidence matrix, the firing rate transitions and the configuration matrix ($x = \{T, \mathcal{T}, \mathcal{P}\}$) of the thermal, tasks, and processors subnet respectively. Each equation from system (3) stands for a module from the TCPN representation on Fig. 1. Eq. (3a) describes the periodic arrival of each task at a rate of $1/\omega_i$, which is defined by the TCPN module ①. The allocation of these tasks is controlled by the flow $\boldsymbol{w^{alloc}}$ through boundary transitions $t^{alloc}_{i,j}$. The processor behaviour is expressed on Eq. (3b), where the state vector $\boldsymbol{m_{\mathcal{P}}}$ stands for the allocation and execution of each task on every processor, specifically on places $p^{busy}_{i,j}$ and $p^{exec}_{i,j}$, respectively. The current accumulated execution of each task is given by Eq. (3c).

Eq. (3d) represents the evolution of system temperature. At the bottom of Fig.1a (*TCPN Thermal Model*) we show a dotted box ③, that corresponds to a prismatic element modeling the conduction, convection and heat generation of a specific chip area. Eq. (3e) indicates that the environmental temperature keeps constant during observation time (its derivative is neglected).

4. CONTROL DESIGN

This section describes *AlECS* (Allocation and Execution Control Scheduler) as a solution to the CRTS problem. *AlECS* consists of three components. The first one yields, entirely off-line, a correct schedule for a HRT task set that minimizes energy and ensures a processor temperature below a thermal bound T_{max}.

To comply with the thermal bound T_{max}, we study the thermal behaviour of the system through a steady state analysis of Eq.(3) as in Rubio-Anguiano et al. (2019), where $\boldsymbol{-SA^{-1}}F^3\boldsymbol{Bw^{alloc}} \leq \boldsymbol{T_{max}} + \boldsymbol{SA^{-1}B'm_a}$ provides the thermal constraints that the allocation of tasks to the processors ($\boldsymbol{w^{alloc}}$) must fulfill. $\boldsymbol{S}$ represents an output matrix, such that $\boldsymbol{Y_T} = \boldsymbol{Sm_T}$ corresponds to the CPUs temperature, $\boldsymbol{A} = \boldsymbol{C_T\Lambda_T\Pi_T(m)}$, $\boldsymbol{B} = \boldsymbol{C'^{exec}_P}$ and $\boldsymbol{B'} = \boldsymbol{C_a\Lambda_a\Pi_a(m)}$. This constraint is used to find a set of operating frequencies $[F^*, F^+]$ such that frequency F^* minimizes power consumption, honoring temporal and thermal constraints, while the frequency can be throttled up to F^+ without violating the thermal restriction. Then, it starts by using a DP-fair approach (Funk et al. (2011)) to compute the *workload assignment*, i.e. the share of each task τ_i that must be executed per scheduling interval. Afterward, it applies a ZL policy and heuristics to schedule the workload onto the processors. The ZL policy determines when a task must start its execution, and the heuristic reduces context switching. Finally, an execution path generator yields a series of *task execution paths* (target functions) representing the fluid execution of tasks (a fluid schedule).

The second component is an on-line stage that add robustness by means of two controllers. A first controller acts upon the flow of trasnsition $t^{alloc}_{i,j}$ in the TCPN model, to ensure the correct allocation of tasks to processors according to the task execution paths calculated off-line. A second controller modifies the flow of transition $t^{exec}_{i,j}$ to ensure the timed execution of the task execution paths by adjusting the CPU frequency. This controller warrants that the off-line fluid schedule is met despite errors on the modeling section or unexpected (but bounded) overheads at run time.

Finally, the third component provides the ability of managing SRT aperiodic tasks by means of the Online Aperiodic Manager (OnAM). The following sections extend on each component of *AlECS*.

4.1 Off-line Zero Laxity schedule

This section details the off-line scheduler, describing the workload assignment of tasks per time interval, and the allocation of this workload to the processors.

Workload Assignment

To solve the workload assignment for the HRT task set $\mathcal{T}$, we formulate a linear programming problem (LPP), where each constraint captures a desired behaviour of our schedule. The solution to the LPP is a set X of x^k_is that represents the share of every task in $\mathcal{T}$ that should be executed per scheduling interval I^k_{SD}, $k = 1, .., \alpha$. We take a DP-fair approach to define these scheduling intervals, considering the ordered set of all job deadlines $SD = \{sd_0, ..., sd_\alpha\}$, where $sd_0 = 0$ and α is the last deadline. We define the scheduling interval as the time between deadlines $I^k_{SD} = [sd_{k-1}, sd_k)$ and $|I^k_{SD}| = sd_k - sd_{k-1}$ represents the duration of the scheduling interval.

We define the *laxity* of τ_i in cycles as $cc^*_i = (\omega_i F^*) - cc_i$, i.e the cycles that task τ_i can remain idle without compromising completion. The total processor cycles at time sd_k can be computed as $(sd_k F^*) = q_i(\omega_i F^*) + r_i$, where $0 \leq r_i < (\omega_i F^*)$ and $q_i \in \mathcal{Z}$, such that r_i represents the amount of cycles that task τ_i has been active since its last deadline and time sd_k. If $r_i = 0$, then sd_k is a deadline of τ_i.

The LPP is formulated in Eq. (4). Each constraint is defined per scheduling interval and per task, until the hyperperiod. The *Maximum utilization constraint* ensures that the system utilization per I^k_{SD} is 100%. The *Execution* *constraint* forces the individual task workloads to complete cc_i at its deadline. If τ_i has not reached its deadline, the *Laxity constraint* guarantees that the sum of the workloads from its last deadline to the current I_{SD}^k should be greater that $r_i - cc_i^*$, such that τ_i is not idle for more than its laxity.

$$
\begin{aligned}
&\max \quad \sum_{i=1}^{n}\sum_{k=1}^{\alpha} x_i^k \\
&s.t \\
&\forall k \sum_{i=1}^{n} x_i^k = m \times |I_{SD}^k| \times F^* && \text{Max. utilization constraint}\\
&\text{if } r_i = 0 \sum_{j=\gamma}^{k} x_i^j = cc_i && \text{Execution constraint}\\
&\text{if } r_i \neq 0 \sum_{j=\gamma}^{k} x_i^j \geq max\{0, r_i - cc_i^*\} && \text{Laxity constraint}\\
&\text{where } \gamma \text{ is 1 or the last deadline interval}\\
&\forall i \quad x_i^k \leq |I_{SD}^k| \times F^* && \text{Sequential constraint}
\end{aligned}
\tag{4}
$$

Proposition 4.1. [1] Given a task set $\mathcal{T}$ as in Def. 3.1, where task utilization at F^* is equal to the number of processors, the solution to the LPP (4) is always integer and if each task τ_i is executed for exactly x_i^k cycles during the k-th interval, then an optimal schedule is obtained.

Zero Laxity policy

The clock frequencies F^*, F^+ and the workload X previously computed determine that task τ_i must be allocated x_i^k cycles at frequency F^* during the interval I_{SD}^k to satisfy the HRT and thermal constraints. This implies that the frequency can be throttled up to F^+ without violating the thermal restriction. However, the actual allocation of tasks to processors requires a scheduling algorithm. In this work, we leverage a ZL policy as posed in Algorithm 1, following the results from Prop. 1.

Algorithm 1 ZLH policy

```
1: Input $I_{SD}^k$ – Scheduling intervals; $X^k$ – CPU cycles per interval
   of each task; $ex_i^k$ – Current execution $P$ cycles in interval $t_0$ –
   Initial time $t_f$– Final time
2: Output A feasible schedule;
   $k = 0$,
3: for $t = t_0$ to $t_f$ do
4:     Compute the laxity of every active task
5:     if reach $I_{SD}^{k+1}$ then
6:         k=k+1;
7:         Compute task priorities as: Jobs with Zero laxity get higher
           priority, followed by jobs that are being executed
8:         Execute the $m$ tasks with higher priority
9:     else if reach a zero laxity then
10:        Compute task priorities
11:        Execute the $m$ tasks with higher priority
12:    end if
13: end for
```

Example 1. Suppose a task system $\mathcal{T} = \{(3,5),(6,10),(9,15),(6,10),(3,5)\}$ to be executed on $m = 3$ processors at $F^* = 1$. The system utilization is $U = 3$. $H = 30$ and the set of deadlines is $SD = \{0,5,10,15,20,25,30\}$. Applying the ZL policy (Alg.1) up to the hyperperiod, we find the target schedule in Fig. 2a.

[1] Please visit **URL** for the complete proof.

So far, we have found an off-line schedule that can be implemented on a real system, in the absence of disturbances. The following two control laws ensures the accomplishment of the schedule under CPU detentions, system time drifting and similar events.

4.2 Execution path generator

The target schedule computed in the previous section is the reference to calculate the *task execution paths* for the control laws to track them, in order to add robustness to the scheduler. Fig. 2b shows the τ_1 task execution paths on each processor for *Example 1.* The vertical solid lines link the scheduling points in the schedule a) and in the task execution paths in b).

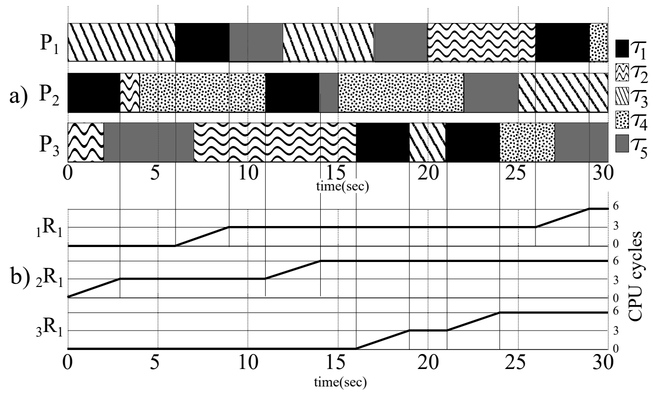

Fig. 2. a) Schedule of system $\mathcal{T} = \{(3,5),(6,10),(9,15),(6,10),(3,5)\}$ on 3 processors. b) Execution paths for τ_1 on each processor.

The execution path per task and processor is defined as:

$$ {}_j\dot{R}_i(t) = F_n[{}_jW_i(t)] \tag{5}$$

where F_n is the operating frequency, $i = 1,2,...,|\mathcal{T}|$, $j = 1,2,...,|\mathcal{P}|$ and

$$ {}_jW_i(t) = \begin{cases} 1 & \text{if } \tau_i \text{ is executed on } P_j \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

Note that ${}_jW_i$ remains constant in the interval of execution $[t_0, t_f)$ of τ_i. Thus, function ${}_2W_1$ in Fig. 2b remains constant during $[0,3)$, as ${}_1W_1$ does in $[6,9)$. Hence, we define the *executing interval* as the time interval where τ_i is being executed. The function ${}_jR_i(t)$ will be used as a set-point for the control stage, to allocate and execute tasks.

4.3 Control law

The allocation and execution of the HRT tasks is modeled in Eq.(3a)-(3c). Specifically, it is determined by the flow of transitions $t_{i,j}^{alloc}$ and $t_{i,j}^{exec}$, respectively. Accordingly, the marking at $m_{i,j}^{exec}$ and $m_{i,j}^{alloc}$, represents which tasks are being allocated and executed, respectively. The controllers for allocation and execution read these measures either from the TCPN equations, when simulating a TCPN model of the system, or from actual counters, when running in a real system.

Allocation control

To allocate jobs to processors according to the task execution paths, we will define an *allocation error* per *executing* *interval* $[t_0, t_f)$, where the desired allocation cycles are obtained by solving Eq. (5) at tf

$$ {}_jR_i(t_f) = {}_jR_i(t_0) + F_n[{}_jW_i(t_0)](t_f - t_0) \tag{7}$$

Let define the *vector allocation error* $\mathcal{E}^{alloc}(t) = [\mathcal{E}^{alloc}_{1,1}, ..., \mathcal{E}^{alloc}_{n,1}, ..., \mathcal{E}^{alloc}_{n,m}]^T$, where each $\mathcal{E}^{alloc}_{i,j}(t)$ is computed as:

$$\mathcal{E}^{alloc}_{i,j}(t) = m^{exec}_{i,j}(t) + m^{busy}_{i,j}(t) - {}_jR_i(t_f) \tag{8}$$

where $t_0 \leq t < t_f$. Taking the time derivative of Eq. 8, the dynamics of the error is given by:

$$\dot{\mathcal{E}}^{alloc}_{i,j}(t) = \dot{m}^{exec}_{i,j} + \dot{m}^{busy}_{i,j} = f^{alloc}_{i,j} - u^{alloc}_{i,j} = w^{alloc}_{i,j} \tag{9}$$

where $w^{alloc}_{i,j}$ is the controlled flow through transition $t^{alloc}_{i,j}$ and $f^{alloc}_{i,j}$ is

$$f^{alloc}_{i,j} = \frac{\lambda^{alloc}_{i,j}}{\eta} m^{idle}_{i,j}. \tag{10}$$

Proposition 4.2. [2] Let $u^{alloc}_{i,j}$ be an ON/OFF control for system (9), such that

$$u^{alloc}_{i,j} = \begin{cases} \dfrac{\lambda^{alloc}_{i,j}}{\eta} m^{idle}_{i,j} & \text{if } \mathcal{E}^{alloc}_{i,j}(t) \geq 0 \\ 0 & \text{if } \mathcal{E}^{alloc}_{i,j}(t) < 0 \end{cases} \tag{11}$$

Then system (9) is stable and each $\mathcal{E}^{alloc}_{i,j}$ remains bounded for all $t_0 \leq t < t_f$.

Execution control

The allocation control is in charge of assigning tasks to processors, whereas the execution control is only concerned with the rate of execution, i.e, the frequency at which the processor should operate to comply with the task execution paths. The frequency is a parameter on the execution module of the TCPN model in Sec. 3.1, specifically on the firing rate $\lambda^{exec}_{i,j} = \eta F^+$ of transitions $t^{exec}_{i,j}$ (the rate at which the processor can consume cycles). In accordance with the off-line calculations, the operating frequency can vary on the interval $[F^*, F^+]$. Our controller acts on the flow trough $t^{exec}_{i,j}$ according to the operating frequency.

The purpose of this controller is to keep the *execution error* $\mathcal{E}^{exec}_{i,j}(t)$ equal to zero. We define the *vector execution error* $\mathcal{E}^{exec}(t) = [\mathcal{E}^{exec}_{1,1}, ..., \mathcal{E}^{exec}_{n,1}, ..., \mathcal{E}^{exec}_{n,m}]^T$, where each $\mathcal{E}^{exec}_{i,j}(t)$ is computed as:

$$\mathcal{E}^{exec}_{i,j}(t) = m^{exec}_{i,j} - {}_jR_i(t). \tag{12}$$

Then the dynamic system of the *execution error* is:

$$\dot{\mathcal{E}}^{exec}_{i,j}(t) = w^{exec}_{i,j} - {}_j\dot{R}_i(t) \tag{13}$$

where $w^{exec}_{i,j}$ is the controlled flow through transition $t^{exec}_{i,j}$:

$$w^{exec}_{i,j} = f^{exec}_{i,j} - u^{exec}$$

such that $f^{exec}_{i,j} = \lambda^{exec} m^{busy}_{i,j}$ and $\lambda^{exec} = \eta F^+$.

Proposition 4.3. [2] Let u^{exec} be a control law for system (13), such that

$$u^{exec} = \begin{cases} 0 & \text{if } \phi \geq \eta F^+ m^{busy}_E \\ \eta(F^+ - F^*) m^{busy}_E & \text{if } \phi \leq \eta F^* m^{busy}_E \\ \eta F^+ m^{busy}_E - \dot{R}_E + \alpha E & \text{otherwise} \end{cases} \tag{14}$$

$$\phi = \dot{R}_E - \alpha E$$

where α is a positive constant, E is the element $\mathcal{E}^{exec}_{i,j}$ such that $|\mathcal{E}^{exec}_{i,j}| = ||\mathcal{E}^{exec}||_\infty$, and m^{busy}_E, $\dot{R}_E$ are the elements $m^{busy}_{i,j}$ and ${}_j\dot{R}_i$ respectively associated to E. Then the *execution error* is locally exponentially stable and the controlled flow is bounded $\eta F^* m^{busy}_E \leq w^{exec}_E \leq \eta F^+ m^{busy}_E$ for all $t_0 \leq t < t_f$.

To prove the effectiveness of the control law, suppose there is an overhead in the system at time $[15, 16]$, and none of the tasks where executed on that interval. Then, the system must increase the frequency to compensate for the delay. Fig. 3a shows the target functions and Fig. 3b shows the output of the system and how the execution accelerates to reach the execution paths.

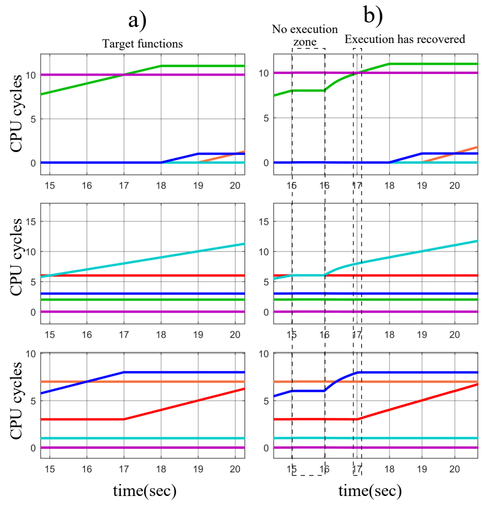

Fig. 3. a) Task execution paths and b) System output recovering from a system overhead in the interval $[15, 16]$

4.4 Online Aperiodic Manager

Now, we endow our scheduler with a third component, the Online Aperiodic Manager (OnAM), which is able to manage SRT aperiodic tasks. Upon arrival of a SRT aperiodic task, the OnAM will determine if such task can be executed without compromising the constraints of the HRT periodic task set. If so, it will re-compute the task execution paths ${}_jR_i$ so that they include the execution of τ^a_i, rejecting the aperiodic task otherwise.

First, the OnAM determines the *scheduling intervals* where τ^a_i will be active (I^r_{SD} to I^f_{SD}). Second, it computes the processor cycles C_u to satisfy the rest of the active tasks until the *deadline* of τ^a_i. Third, With this information, the algorithm computes a frequency such that there is enough time left for executing the new task: $F_n = max\left\{\frac{C_u + cc^a_i}{m \times d^a_i}, \frac{cc^a_i}{d^a_i}\right\}$ Last, if $F_n \leq F^+$, the OnAM accepts the task into the system and assigns the workload xa^k_i (per I^k_{SD}) for τ^a_i proportionally to the *scheduling interval* duration: $xa^k_i = \frac{|I^k_{SD}|}{d^a_i} cc^a_i$ If the arrival or deadline $(r^a_i + d^a_i)$ of the aperiodic task does not match any current deadline, the current interval duration is different from I^k_{SD}, thus xa^k_i is slightly modified. If the mismatch occurred at arrival time, instead of $|I^k_{SD}|$, we use $sd^k - r^a_i$. However, if it occurred at the deadline, then we use $(r^a_i + d^a_i) - sd^{k-1}$. The frequency F_n along with the new workload that accounts

[2] Please visit **URL** for the complete proof.

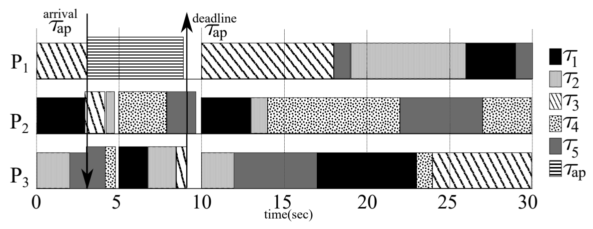

Fig. 4. Schedule of system $\mathcal{T} = \{(3,5),(6,10),(9,15),(6,10),(3,5)\}$ serving an aperiodic task $\tau_{ap} = (10,6,3)$

for τ_i^a serves as input for Alg. 1, which now computes online ${}_jW_i$ and ${}_jR_i$ for the interval $[ra, sd_f)$, as in Eq. (5) and Eq. (6). When task τ_i^a reaches its deadline, the execution paths are computed for the rest of active tasks under the previous frequency. Algorithm 2 details the workflow of the OnAM.

Algorithm 2 OnAM

```
1:  Input $I_{SD}^k$ – Scheduling intervals; $X^k$ – tasks $CPU$ cycles per
    interval; $ex_i^r$ – current execution $P$ cycles in interval $cc_i^a, d_i^a$–
    Aperiodic tasks parameters;
2:  Output ${}_jR_i$
3:  if periodic task arrives then
4:      Determine intervals $I_{SD}^r$ to $I_{SD}^f$ , where $\tau_i^a$ is active
5:      Compute required CPU cycles for active tasks
6:      if $F_n \le F^+$ then
7:          Accept task $\tau_i^a$
8:          Assign workload $xa_i^k$ from $I_{SD}^r$ to $I_{SD}^f$
9:          Execute Alg.1 from $[ra, sd_f)$
10:         Compute task execution paths ${}_jR_i$
11:     else
12:         Reject task
13:     end if
14: end if
15: if aperiodic task deadline then
16:     Execute Alg.1 from $[sd_f, H)$
17:     Compute task execution paths ${}_jR_i$
18: end if
```

Example 2. Recall *Example 1* and now suppose there is an aperiodic task $\tau_1^a = (10,6,3)$. The interval of admissible frequencies is $F = [1,3]$. Fig. 4 shows the target schedule. Fig. 5 shows the new execution paths computed to accommodate the aperiodic task. On CPU 1, ${}_1R_{ap}$ shows a ramp until $t = 9$ whereas ${}_2R_{as}$ and ${}_3R_{as}$ are flat for the whole interval in CPU 2 and 3, meaning that the incoming aperiodic task can be entirely accommodated in CPU 1 without disturbing the correct execution of tasks 2 and 3 now scheduled on CPU 2 and 3.

5. CONTEXT SWITCHING AND MIGRATION: COMPARISON WITH RUN

We perform a comparison of context switch and migration overhead with the off-line schedule obtained from Sec. 4.1 and RUN (Regnier et al. (2011)) using Tertimuss, an in-house simulation framework publicly available (Desirena et al. (2019a)). Tertimuss allows the parameterization of processors and task sets, automatically generates the TCPN model and state equations, solves the systems, and eases the simulation workflow. It includes a number of schedulers available out-of-the-box. RUN does not support

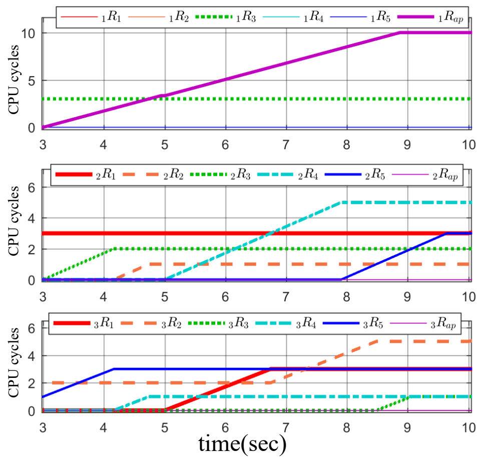

Fig. 5. Task execution paths computed to satisfy the aperiodic task

DVFS, nor thermal or disturbance management. Therefore, for the comparison to be fair, the frequency of every core is fixed, we obviate power consumption and thermal behavior, and we assume no disturbances. We consider two processor cores with cache memories and speculative mechanisms non-existent or turned off. We produced task sets with 10 tasks per set and deadlines of 2, 5 and 10 s with a utilization randomly generated under a uniform distribution by using UUnifast (Bini and Buttazzo (2005)), integrated in Tertimuss.

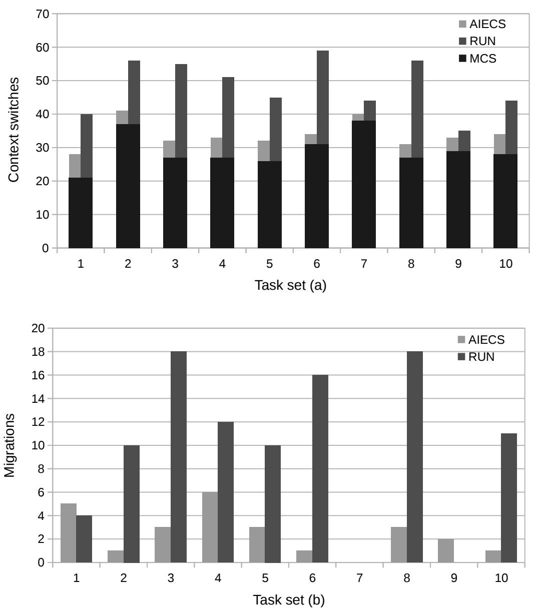

Fig. 6. (a) Breakdown of context switches generated by our proposal vs RUN along the ten task sets. The bottom black bars stand for the MCS and the upper stacked bars represent the CCS of the proposal (grey) and RUN (dark grey). (b) Number of job migrations produced by our proposal (grey) vs. RUN (dark grey)

We measured separately the number of *Mandatory Context Switches* (MCS) and *Coerced Context Switches* (CCS). MCS are given by job activation and termination, and therefore are independent from the scheduler, unlike CCS. Fig. 6 (a) displays two stacked bars per experimental task set, the bottom black bar of which represents the MCS, amounting to the same value in both schedulers as expected. RUN triggers between 6% (set 7) and up to 80% (set 6) more CCS than our proposal, averaging 44%. Fig. 6 (b) shows that our proposal cuts by almost a quarter the number of job migrations yielded by RUN, which reaches zero migrations against the two produced by our proposal in set 9 nonetheless.

6. CONCLUSIONS

With this contribution we show the power of Timed Continuous Petri Nets (TCPN) to model a RT multiprocessor system and to ease the design and test of the complex RT multiprocessor schedulers demanded by today's MPSoCs. We present a scheduling scheme which provides entirely off-line a schedule that meets the HRT, thermal and power constraints of a periodic task set. We reformulate the ILP solved in Rubio-Anguiano et al. (2019) to calculate the share of each task job per time interval. We greatly improve the schedulability of the former approach by posing and solving the ILP taking into account the whole hyperperiod. Besides, we apply a ZL task policy off-line to allocate jobs to processors, in contrast with Rubio-Anguiano et al. (2019) where the allocation is performed on-line. We lower the context switches and migrations yielded by Rubio-Anguiano et al. (2019), outperforming RUN, which can be considered a reference in this point, in a preliminary comparison. We add robustness to the system by designing a second on-line component where an allocation controller and an execution controller respectively compare actual or simulated allocation and execution data with the values (execution paths) provided by the previous off-line stage, taking action to bring the error to zero. As a bonus, the execution controller allows the design of an aperiodic task manager (onAM) simpler and lighter than the one presented in Rubio-Anguiano et al. (2019).Immediate future work include controlling possible thermal disturbances in a real environment. Next, we aim to adapt our model to an heterogeneous architecture, leveraging our underlying TCPN model.

REFERENCES

Baruah, S., Bertogna, M., and Butazzo, G. (2015). *Multiprocessor Scheduling for Real-Time Systems.* Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Baruah, S.K., Cohen, N.K., Plaxton, C.G., and Varvel, D.A. (1996). Proportionate progress: A notion of fairness in resource allocation. *Algorithmica*, 15(6), 600–625.

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

David, R. and Alla, H. (2008). Discrete, continuous and hybrid Petri nets (david, r. and alla, h.; 2004). *Control Systems, IEEE*, 28(3), 81–84.

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

Desirena, G., Rubio, L., Ramirez, A., and Briz, J. (2019a). Thermal-Aware HRT Scheduling simulation framework. https://webdiis.unizar.es/gaz/repositories/tertimuss.

Desirena-Lopez, G., Ramírez-Treviño, A., Briz, J., Vázquez, C., and Gómez-Gutiérrez, D. (2019b). Thermal-aware real-time scheduling using Timed Continuous Petri nets. *ACM Transactions on Embedded Computing Systems (TECS)*, 18(4), 36.

Funk, S., Levin, G., Sadowski, C., Pye, I., and Brandt, S. (2011). Dp-fair: a unifying theory for optimal hard real-time multiprocessor scheduling. *Real-Time Systems*, 47(5), 389–429.

Regnier, P., Lima, G., Massa, E., Levin, G., and Brandt, S. (2011). Run: Optimal multiprocessor real-time scheduling via reduction to uniprocessor. In *Proceedings of the 2011 IEEE 32Nd Real-Time Systems Symposium*, RTSS '11, 104–115. IEEE Computer Society, Washington, DC, USA. doi:10.1109/RTSS.2011.17. URL http://dx.doi.org/10.1109/RTSS.2011.17.

Rubio-Anguiano, L., Desirena-López, G., Ramírez-Treviño, A., and Briz, J. (2019). Energy-efficient thermal-aware multiprocessor scheduling for real-time tasks using TCPN. *Discrete Event Dynamic Systems*, 1–28.

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

Silva, M. and Recalde, L. (2007). 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.

Thammawichai, M. and Kerrigan, E.C. (2018). Energy-efficient real-time scheduling for two-type heterogeneous multiprocessors. *Real-Time Systems*, 54.