Thermal-Aware Real-Time Scheduling using Timed Continuous Petri Nets

G. Desirena-López, CINVESTAV-IPN Unidad Guadalajara
A. Ramírez-Treviño, CINVESTAV-IPN Unidad Guadalajara
J.L. Briz, DIIS/I3A Universidad de Zaragoza
C.R. Vázquez, Tecnologico de Monterrey, Escuela de Ingeniería y Ciencias, Guadalajara.
D. Gómez-Gutiérrez, Tecnologico de Monterrey, Escuela de Ingeniería y Ciencias, Guadalajara.

We present a thermal-aware, hard real-time (HRT) global scheduler for a multiprocessor system designed upon three novel techniques. First, a modeling methodology based on Timed Continuous Petri nets (TCPN) which yields a complete state variable model, including job arrivals, CPU usage, power, and thermal behavior. The model is accurate and avoids the calibration stage of RC thermal models. Second, based on this model, a linear programming problem (LPP) determines the existence of a feasible HRT thermal-aware schedule. Last, a sliding-mode controller and an on-line discretization algorithm implement the global HRT scheduler, which is capable of managing thermal constraints, context switching, migrations and disturbances.

Additional Key Words and Phrases: Real-Time Systems, Thermal-Aware Scheduling, Multiprocessor, modeling, Feedback Control, Timed Continuous Petri Nets.

1. INTRODUCTION

Multi-core systems on-chip (MPSoCs) are becoming pervasive in embedded systems. They are considered as the next unavoidable step in hard real-time (HRT) mission-critical environments, traditionally based on single-core processors, because they provide long-term system flexibility. Flight management or on-board maintenance systems tend to expand, and single-cores impose strong limits on the available CPU time when the applications grow. Leveraging the computing power of MPSoCs is, however, quite a challenge. First, HRT task scheduling is far more complicated on multiprocessors than on single-core processors. Second, power consumption can be an issue in battery-powered systems. Last, inefficient thermal management can lead to unexpected failures or to a short chip lifespan. Also, handhelds and other embedded devices impose thermal constraints because of ergonomic reasons (typically 45^oC for plastic and 41^oC for aluminum enclosures [Berhe 2007]).

RT scheduling in multiprocessors has been mostly tackled under two different approaches: *partitioned* and *global* scheduling ([Baker 2005], [Davis and Burns 2011]), along with some hybrid methods [Casini et al. 2017]. In *partitioned* scheduling, tasks are statically allocated to CPUs, and then a uniprocessor scheme can be applied. Since

Authors' addresses: G. Desirena-López, and A. Ramírez-Treviño are with the CINVESTAV-IPN Unidad Guadalajara, Av. del Bosque 1145, CP 45019, Zapopan, Jalisco, Mexico {gdesirena}{art}@gdl.cinvestav.mx; C.R. Vázquez and D. Gómez-Gutiérrez are with Tecnologico de Monterrey Campus Guadalajara, Av. Ramón Corona 2514, CP 45201,Zapopan, Jalisco, Mexico {cr.vazquez}{david.gomez.g}@itesm.mx; J.L. Briz is with Dept. de Informática e Ing. de Sistemas - IA3 Universidad de Zaragoza, María de Luna 1 - 50018 Zaragoza, España. briz@unizar.es

DOI: http://dx.doi.org/10.1145/0000000.0000000

task allocation is fixed, there is little leeway to allocate new tasks. *Global* schedulers allocate tasks to any CPU, allowing migrations. There is a large list of recent multiprocessor scheduling algorithms considering RT or thermal constraints (e.g. [Kong et al. 2014; Murali et al. 2008; Shi et al. 2010; Fu et al. 2012; Fu et al. 2009]). This work proposes a HRT thermal-aware schedule for multicore systems with a global scheduling scheme. As is customary in the field, we assume independent and periodic tasks, which period, deadline, consumed energy and Worst Case Execution Time (WCET) are known beforehand [Baruah et al. 2015].

There are three novel contributions in our approach. First, we leverage a modeling methodology based on Timed Continuous Petri nets (TCPN). This leads to a complete state variable model, under a single formalism, including job arrivals, CPU usage, power, and thermal aspects. Second, upon this model we state a Linear Programming Problem (LPP) whose solution is a set of coefficients that are used to compute *fluid schedule functions*. These functions represent a feasible execution of a HRT task set on the available CPUs, warranting maximum utilization under thermal constraints. Third, a sliding mode feedback controller tracks the computed fluid schedule functions, yielding a HRT thermal-aware global scheduler able to deal with disturbances.

The advantages of the first contribution over common approaches such as RC thermal equivalent models [Skadron et al. 2010] are that calibrations are obviated, greatly improving accuracy. Also, the state space variable representation provided by the TCPN allows the prediction of future states, and the process of analyzing the system and designing the controller is easier, thanks to the structural and dynamic properties of the TCPN model. The benefit of our second contribution lies in that the LPP solution is computed off-line in polynomial time, simplifying the subsequent on-line scheduling and avoiding the heuristic algorithms reported in other approaches [Donald and Martonosi 2006] [Zanini et al. 2009]. Last, the virtue of our third contribution is that a sliding mode controller makes easier the implementation of on-off control laws, leading to low-overhead schedulers, capable of handling perturbations or CPU detentions in underloaded systems without rescheduling a job.

To the best of our knowledge, this is the first work on HRT thermal-aware scheduling leveraging TCPNs. We compare trough simulations three possible implementations of the algorithm to show the impact on two conflicting objectives: low context-switching and migration, and thermal control. A first quantum-based implementation sacrifices the first objective on behalf of a better thermal control. A second implementation, consisting in a bare deadline-partitioning approach, achieves just the opposite effect. A third opportunistic implementation balances these two objectives under specific restrictions. Finally, we show that the sliding mode feedback controller allows dealing with system disturbances as long as the system utilization keeps lower than 100%.

This paper is organized as follows. Section 2 discusses the related work. Section 3 briefly introduces TCPNs and defines the problem addressed. Section 4 presents the modeling methodology. Section 5 investigates the existence of a solution for the thermal-aware HRT scheduling problem. Section 6 designs a feedback controller that implements a global fluid schedule, and presents two possible discrete implementations. Sec. 7 presents some key examples. Finally, Section 8 summarizes conclusions and future work.

2. RELATED WORK

Thermal-aware scheduling has been widely studied for single core systems, scaling the processor frequency to reduce its power consumption and, accordingly, its temperature. In [Kong et al. 2014] a Dynamic Voltage and Frequency Scaling ($DVFS$) technique are used to control power consumption and temperature. In contrast, we assume a fixed frequency for every core in this work yet our underlying model is ready for applying $DVFS$ if required.

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. [Donald and Martonosi 2006] propose a general framework for dynamic thermal management for multicore. It consists of a hierarchical feedback control loop with PI controllers but does not guarantee HRT performance. The thermal problem is defined as a control theory problem with a state space representation in [Zanini et al. 2009]. They propose 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 soft RT systems ([Fu et al. 2009], [Fu et al. 2012]), allowing for a certain percentage of missed deadlines.

[Ahmed et al. 2016] tackle the problem of thermal constrained scheduling of periodic tasks, but they assume partitioned scheduling instead of global (migration) scheduling like we do. [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.

Fluid scheduling is a powerful theoretical instrument. However, since it instantaneously shares the CPU among all the active tasks, practical implementations seek to interleave the tasks trying to keep a *fair* CPU share within time periods. Proportionate Fairness (P-Fair) [Baruah et al. 1996], PD [Baruah et al. 1995] and PD^2[Anderson and Srinivasan 2001] are all well-known algorithms based on the notion of fairness that have been proved optimal when RT constraints are considered. They imply a high overhead due to context switching and migration. This overhead has been improved with *deadline partitioning* schemes ($B-Fair$, Boundary Fairness) [Zhu et al. 2003], ***DP-Fair*** [Chandra et al. 2001]). It was still reduced by relaxing the fairness requirement [Nelissen et al. 2011] with a loss of optimality in terms of CPU usage. Deadline partitioning splits time into slices demarcated by the deadlines of all tasks in the system, greatly reducing the scheduling points. However, these slices can be too long to cope with temperature variations. This problem can be solved by considering a quantum, which would compromise the effectiveness of the deadline partitioning approach, as an infinitesimal quantum would match the fluid scheduler, whereas the maximum quantum value should be limited by time constraints. This was resolved in [Desirena-Lopez et al. 2016], for a HRT fluid scheduler without thermal constraints, by defining the quantum as the greatest common divisor of the deadlines of all tasks. In this paper we leverage that solution to meet the thermal constraint, although the accuracy of thermal control is related to the amount of context-switching and migration. The latter can be reduced by a ***DP-Fair*** implementation at the cost of losing grip on temperature. Our methodology allows defining the circumstances under which an optimistic ***DP-Fair*** provides a precise thermal control while lowering context-switching and migrations.

3. BACKGROUND

This Section introduces basic definitions concerning $TCPNs$. An interested reader may also consult [David and Alla 2008], [Vázquez et al. 2014] to get a deeper insight in the field.

3.1. Timed continuous Petri nets

Definition 3.1. A Timed Continuous Petri net (***TCPN***) is a time-driven continuous-state system described by the tuple $(N, \boldsymbol{\lambda}, \boldsymbol{m}_0)$. N is a Petri net (graph) structure, and it is defined as a tuple $N = (P, T, \boldsymbol{Pre}, \boldsymbol{Post})$, where P and T are finite disjoint sets of places (circles representing local states) and transitions (rectangles representing events), respectively. $\boldsymbol{Pre}$ and $\boldsymbol{Post}$ are $|P| \times |T|$ $\boldsymbol{Pre}-$ and $\boldsymbol{Post}-$ incidence matrices, where $\boldsymbol{Pre}(i,j)$ (resp. $\boldsymbol{Post}(i,j)$) is the weight of the arc going from transition $t_j \in T$ to place $p_i \in P$ (resp. going from place $p_i \in P$ to transition $t_j \in T$), otherwise $\boldsymbol{Pre}(i,j) = 0$ (resp. $\boldsymbol{Post}(i,j) = 0$). The *incidence matrix of* N is defined as $\boldsymbol{C} = \boldsymbol{Post} - \boldsymbol{Pre}$. The column vector $\boldsymbol{m}_0 \in \{\mathbb{R}^+ \cup 0\}^{|P|}$ is the initial state (*initial marking*). Its i-th entry represents the marks in place $p_i \in P$ the initial state. In continuous Petri nets, the marking and transition firings (event occurrences) can take any value in $\mathbb{R}$. A transition t_i 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 vector $\boldsymbol{\lambda} \in \{\mathbb{R}^+ \cup 0\}^{|T|}$ represents the transitions rates (the speed at which transitions fire), i.e. the temporal evolution of the system.

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

The firing rate matrix is defined as $\boldsymbol{\Lambda} = diag(\lambda_1, ..., \lambda_{|T|})$. For the flow to be well defined, every continuous transition must have at least one input place, hence we assume $\forall t \in T, |^\bullet t| \geq 1$. The *min* in the definition above 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:

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

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

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

The Petri net (graph) structure determines the incidence matrix $\boldsymbol{C}$. The current marking (state) $\boldsymbol{m}$ determines the configuration matrix $\Pi(m)$, and the rates $\boldsymbol{\lambda}$ determine Λ. Thus, (2) can be used to simulate the TCPN, based on the knowledge of the graph, initial state, and rates.

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

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

Example 3.2. Consider the manufacturing system in Fig. 1b. Raw materials arrive to input buffer I. A resource of type R moves the material to the machine M, after which the resource is released. M performs some operations over the material to obtain a final product. A resource of type R unloads the machine and moves the product to the output buffer O. Then, the resource is released.

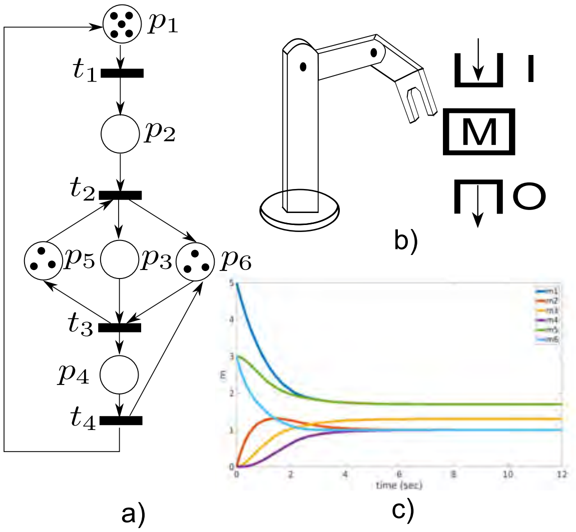


Fig. 1: a) TCPN model. b) Manufacturing Process. c) Marking evolution.

Fig. 1a shows the TCPN model of the system. The quantity of raw material is modeled as tokens in place p_1, the resource is modeled by place p_6, and the capacity of the machine is the initial marking of the net. The resource allocation policy is modeled by the firing of transitions, and the number of products per unit time is the throughput of the net system. If the initial marking is set as $m_0 = [5\ 0\ 0\ 0\ 3\ 3]$ and the firing rate vector is $\lambda = [1\ 1\ 1\ 1]^T$, then $\Lambda = diag(\Lambda)$. Therefore, $\boldsymbol{f(m)} = \boldsymbol{\Lambda\Pi(m)m}$ and from Eq. (3), we have:

$$\dot{m} = \begin{bmatrix} -1 & 0 & 0 & 1 \\ 1 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & -1 & 1 & 0 \\ -1 & 1 & -1 & 1 \end{bmatrix} \begin{bmatrix} \lambda_1 \cdot m_1 \\ \lambda_2 \cdot min\{m_2, m_5\} \\ \lambda_3 \cdot min\{m_3, m_6\} \\ \lambda_4 \cdot m_4 \end{bmatrix} = \begin{bmatrix} m_4 - m_1 \\ m_1 - min\{m_2, m5\} \\ min\{m_2, m_5\} - min\{m_3, m_6\} \\ min\{m_3, m_6\} - m_4 \\ min\{m_3, m_6\} - min\{m_2, m_5\} \\ m_4 - m_1 + min\{m_2, m_5\} - min\{m_3, m_6\} \end{bmatrix}$$

Note that $enab(t_1, m_0) = m_0[p_1] = m_1$, $enab(t_2, m_0) = m_0[p_2] = m_2$, $enab(t_3, m_0) = m_0[p_3] = m_3$ and $enab(t_4, m_0) = m_0[p_4] = m_4$. In the initial state, the configuration is given by (p_1, t_1), (p_2, t_2), (p_3, t_3) and (p_4, t_4). The marking evolution is depicted on Fig. 1c. As the system evolves, the marking m_3 increases and the marking m_6 decreases. At time $\tau = 2$, $m_2 = m_6$ and the configuration switches to (p_1, t_1), (p_2, t_2), (p_6, t_3) and (p_4, t_4). The marking evolution graph shows the two different dynamics provided by the two different system configurations.

3.2. Problem definition

Definition 3.3. The system consists of a set of n periodic tasks $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_n\}$ that must be allocated and executed on a set of m identical processors $\mathcal{P} = \{CPU_1, \ldots, CPU_m\}$ with an homogeneous clock frequency F. A task τ_i is characterized by a 4-tuple $\tau_i = (cc_i, d_i, \omega_i, e_i)$, where cc_i are the CPU cycles required to complete an instance of the task or *job*, calculated as the Worst-Case Execution-Time (WCET); ω_i is the task's period, d_i is the associated implicit deadline ($d_i = \omega_i$), and e_i is the energy demanded by the task during execution. All tasks hold the same priority and are independent (with no resource sharing). A job τ_i^k of task $\tau_i \in \mathcal{T}$ must run to completion within the time period $[(k-1)\omega_i, (k-1)\omega_i + d_i]$, i.e. it executes for a fixed number of cc_i CPU cycles. Once a job τ_i^k is over, a new one (τ_i^{k+1}) becomes immediately ready for execution at time $k\omega_i$.

We define the hyperperiod as the period equal to the least common multiple $H = lcm(\omega_1, \omega_2, \ldots, \omega_n)$ of all tasks periods. A task τ_i executed on a processor at clock frequency F requires $c_i = \frac{cc_i}{F}$ 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: $U = \sum_{i=1}^{n} \frac{c_i}{\omega_i}$. This work only considers the case where $U \leq m$ [Baruah et al. 1996]. A schedule is a set of 3-tuples $(\tau_i^q, CPU_k, [\zeta_r, \zeta_s])$, where τ_i^q is the $q-th$ job of task τ_i that is allocated to CPU_k, starting execution at time ζ_r and completing or being preempted at ζ_s. A schedule is considered *feasible* if it can be repeated every hyperperiod while meeting the required deadlines [Liu and Layland 1973].

Feasible schedules can be computed before system operation as a set of 3-tuples, or can be computed online, determining which jobs must be allocated to which processors. In this work, we leverage on-line scheduling for a multiprocessor system with a fixed frequency.

Problem Definition 3.1. *Thermal-aware fluid scheduler. Given the system defined in Def. 3.3, design an algorithm to allocate within the hyperperiod the tasks in $\mathcal{T}$ to the m identical CPUs in $\mathcal{P}$ in such a way that the deadlines for $\mathcal{T}$ are always satisfied, and the CPU temperatures are always kept below a given temperature threshold $\boldsymbol{T_{max}}$.*

4. SYSTEM MODELING METHODOLOGY

A TCPN system model is an aggregation of three types of submodels, respectively for tasks, CPUs and thermal behavior.

4.1. TCPN submodel for tasks

The TCPN task model for the task set presented in Def. 3.3 is fully explained in [Desirena-Lopez et al. 2016] and summarized here. The top row in Fig. 2a (*TCPN module for τ_i*) depicts this model.

The task period ω_1 of task τ_1 implies that $\frac{1}{\omega_1}$ jobs arrive per second on average (task arriving frequency). This is modeled as the firing rate $\lambda_1^{\omega} = \frac{1}{\omega_1}$ of transition t_1^{ω} in the figure. The arc from transition t_1^{ω} (jobs arrival) to place p_1^{cc} (arrived job cycles) stands for the task WCET in CPU cycles. The weight cc_i of the arc represents the marking of place p_1^{cc} in CPU cycle units. Tasks relative deadlines are captured as tokens inside places (p_1^d for τ_1). We assume implicit deadlines($d_i = \omega_i$) so we use ω_i in computations throughput this work instead of d_i.

4.2. TCPN submodel for CPUs

The TCPN module CPU_j of Fig. 2a models a CPU. It consists of transitions $t_{1,j}^{alloc}$, ... , $t_{n,j}^{alloc}$, transitions $t_{1,j}^{exec}$, ... , $t_{n,j}^{exec}$, places $p_{1,j}^{busy}$, ... , $p_{n,j}^{busy}$ and the place p_j^{idle}. Transitions $t_{1,j}^{alloc}$, ... ,$t_{n,j}^{alloc}$ (*allocation transitions*) model the allocation of the jobs of tasks τ_1, ... ,τ_n to processor CPU_j. The transition rates $\lambda_{1,j}^{alloc}$, ... , $\lambda_{n,j}^{alloc}$ match the allocation rate of jobs to the processor.

Places $p_{1,j}^{busy}$, ... , $p_{n,j}^{busy}$ represent the busy state of the processor, once a job of task τ_i is allocated to CPU_j. Transitions $t_{1,j}^{exec}$, ... , $t_{n,j}^{exec}$ represent the execution of the corre-

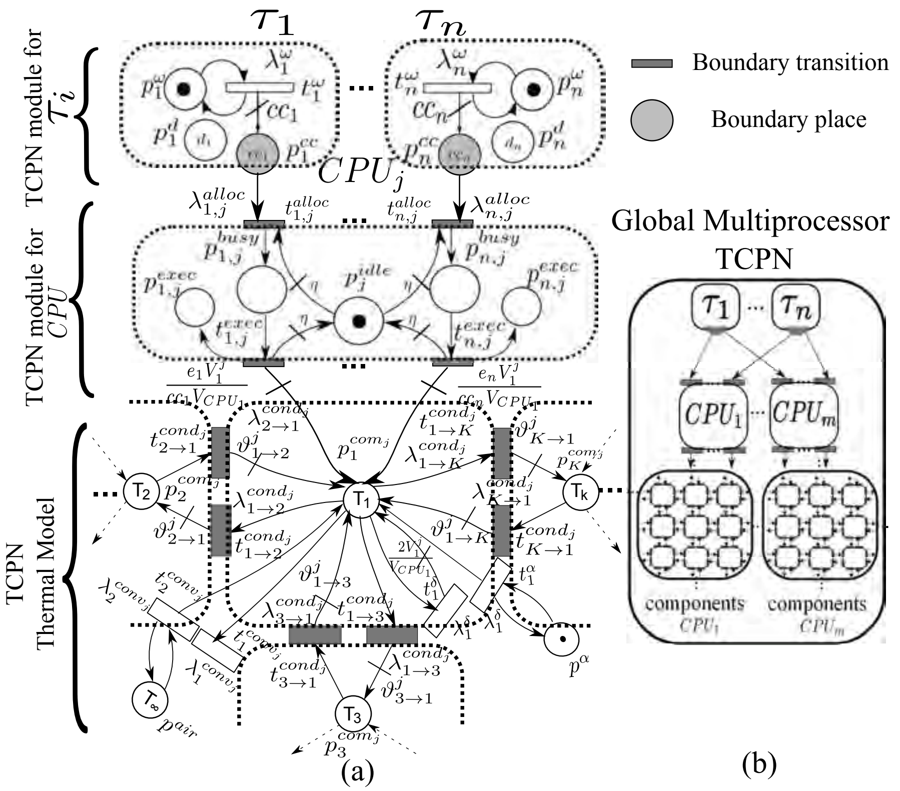


Fig. 2: (a) Detailed TCPN global model of a monoprocessor system. It includes the task, CPUs and thermal models. (b) Global multiprocessor TCPN schematic representation. Weights $\vartheta^j_{q\to q+1} = \frac{\lambda^{cond_j}_{q\to q+1}}{\lambda^{cond_j}_{q+1\to q}}$, where $q = 1, \ldots, K-1$

sponding task jobs on CPU_j. Transition rates $\lambda^{exec}_{1,j}$, ... , $\lambda^{exec}_{n,j}$ represent the execution rate. The model considers that jobs start to run immediately after the allocation. [1].

The marking of a place p^{idle}_j models the availability of CPU_j (*throughput capacity*). The initial marking at p^{idle}_j is set to 1, meaning that the CPU_j is idle.

The arcs from transitions $t^{exec}_{1,j}$, ... , $t^{exec}_{n,j}$ to place p^{idle}_j and from place p^{idle}_j to transitions $t^{alloc}_{1,j}$, ... , $t^{alloc}_{n,j}$ are weighted by a constant value η, to ensure that the flow in transitions $t^{alloc}_{i,j}$ is limited by the CPU throughput capacity modeled by place p^{idle}_j.

The power consumed by a CPU_j has two components: the dynamic power due to computational activities of tasks P_{dyn}, and the static power due to leakage currents P_{leak}. The latter can be modeled as a linear function of temperature for the limited operating range ([Liu et al. 2007]). Thus,

$$P_{CPU_j} = P_{dyn_j} + P_{leak_j} \tag{4}$$

[1] For simulation reasons, the execution rates of CPU_j can be set as $\lambda^{exec}_{i,j} = \eta F$, where F is the frequency of CPU_j and η is a modeling parameter to ensure that p^{idle}_j constraints $t^{alloc}_{i,j}$ (a suitable value is given by $\eta > 10$). The firing rate of transition $t^{alloc}_{i,j}$ can be set as $\lambda^{alloc}_{i,j} = \eta\lambda^{exec}_{i,j}$.

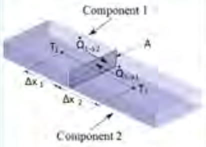
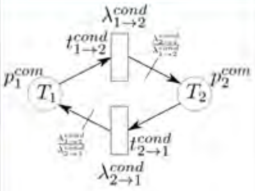
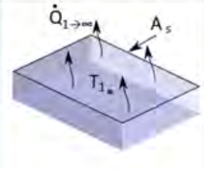
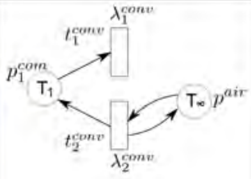
	Components Subject to Heat Transfer Mechanisms	TCPN module of each Heat Transfer Mechanisms	TCPN module parameters
Thermal Conduction	 Component 1 $T_{1\infty}$ $\dot{Q}_{1\to2}$ A $\dot{Q}_{2\to1}$ T_2 Δx_1 Δx_2 Component 2	 $\lambda_{1\to2}^{cond}$ $t_{1\to2}^{cond}$ p_1^{com} T_1 T_2 p_2^{com} $t_{2\to1}^{cond}$ $\lambda_{2\to1}^{cond}$	$\lambda^{cond} = \begin{bmatrix}\lambda_{1\to2}^{cond}\\ \lambda_{2\to1}^{cond}\end{bmatrix} = \begin{bmatrix}\frac{1}{V_1\rho_1 cp_1}\cdot\frac{k_1k_2A}{k_2\Delta x_1 + k_1\Delta x_2}\\ \frac{1}{V_2\rho_2 cp_2}\cdot\frac{k_1k_2A}{k_2\Delta x_1 + k_1\Delta x_2}\end{bmatrix}$ $Pre = \begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix},\ Post = \begin{bmatrix}0 & \frac{\lambda_{1\to2}^{cond}}{\lambda_{2\to1}^{cond}}\\ \frac{\lambda_{2\to1}^{cond}}{\lambda_{1\to2}^{cond}} & 0\end{bmatrix}$
Thermal Convection		 λ_1^{conv} t_1^{conv} p_1^{com} T_1 T_∞ p^{air} t_2^{conv} λ_2^{conv}	$\lambda^{conv} = \begin{bmatrix}\lambda_1^{conv}\\ \lambda_2^{conv}\end{bmatrix} = \begin{bmatrix}\frac{hA_s}{V_1\rho_1 cp_1}\\ \frac{hA_s}{V_1\rho_1 cp_1}\end{bmatrix}$ $Pre = \begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix},\ Post = \begin{bmatrix}0 & 1\\ 0 & 1\end{bmatrix}$

Fig. 3: Thermal conduction and convection mechanisms and their TCPN models.

P_{dyn_j} is the dynamic power due to computational activities. The summation $P_{leak_j} = \delta T_j + \rho$ approximates leakage power consumption, where T_j is the $j-th$ processor temperature and δ, ρ are modeling constants [Liu et al. 2007].

The average energy consumed by task τ_i in one cycle running on CPU_j at frequency F is defined as $e_i = P_{CPU_j}/F$. In Fig. 2a, module CPU_j, the execution flow $f_{i,j}^{exec}$ of transitions $t_{i,j}^{exec}$ stands for the number of CPU cycles of task τ_i executed on CPU_j. Therefore the dynamic power consumed by CPU_j when task τ_i runs cc_i cycles can be stated as:

$$P_{dyn_j} = \sum_{\tau_i \text{executed in } CPU_j} f_{i,j}^{exec}\frac{e_i}{cc_i} \tag{5}$$

4.3. TCPN Thermal submodel

We leverage the thermal model presented in [Desirena-Lopez et al. 2014]. It divides a multiprocessor into prismatic solid components capable of heat generation, thermal conduction and thermal convection, yielding a general thermal equation and a state space representation of the system.

4.3.1. Thermal Conduction and convection. The thermal conduction of two adjacent prismatic components is modeled according to the equation of the TCPN module together with the parameters shown in the row *Thermal Conduction* of Fig. 3. These components (prisms) are assumed to hold isotropic properties (thermal conductivity coefficients k_1, k_2; volumes V_1, V_2; densities ρ_1, ρ_2 and specific heat capacities cp_1, cp_2). The marking in the TCPN places (p_1^{com} and p_2^{com}) represents the average temperature of component 1 and component 2 respectively.

The thermal convection in a prismatic component with a convection coefficient h and temperature T_1 is modeled by the TCPN module shown at the row *Thermal Convection*

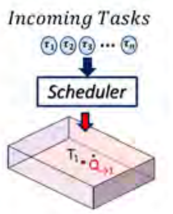
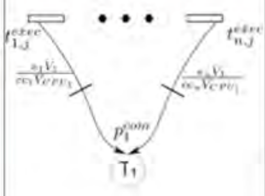
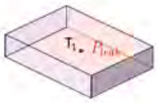
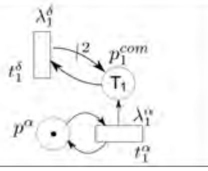
	Components Subject to Heat Transfer Mechanisms	TCPN module of each Heat Transfer Mechanisms	TCPN module parameters
Heat Generation due to dynamic Power	 Incoming Tasks τ_1 τ_2 τ_3 ... τ_n Scheduler T_1 $\dot{Q}_{\tau 1}$	 $t^{exec}_{1,j}$ $\frac{e_1V_1}{cc_1V_{CPU_1}}$ $t^{exec}_{n,j}$ $\frac{e_nV_1}{cc_nV_{CPU_1}}$ p_1^{com} T_1	$\lambda_{\mathcal{P}}^{T} = \begin{bmatrix} \lambda^{exec}_{1,1} \\ \vdots \\ \lambda^{exec}_{n,m} \end{bmatrix}$ $Pre = [0 \ \ \ldots \ \ 0]$, $Post = \begin{bmatrix} \frac{e_1V_1}{cc_1V_{CPU_1}} & \ldots & \frac{e_nV_k}{cc_nV_{CPU_m}} \end{bmatrix}$
Leakage Power		 λ_1^{δ} t_1^{δ} 2 p_1^{com} T_1 p^{α} λ_1^{α} t_1^{α}	$\lambda^{leak} = \begin{bmatrix} \lambda_1^{\delta} \\ \lambda_1^{\alpha} \end{bmatrix} = \begin{bmatrix} \delta \\ \alpha \end{bmatrix}$ $Pre = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, $Post = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}$

Fig. 4: Heat generation mechanisms and their TCPN models.

of Fig. 3. The marking in places p_1^{com} and p^{air} represents the average temperatures of the component and the ambient temperature, respectively.

4.3.2. Heat generation. A prismatic component increments its thermal energy due to the execution of tasks' jobs. This is modeled by Eq. (5), corresponds to the TCPN module shown at the first row of Fig. 4.

The thermal energy due to the leakage of a prismatic component with temperature T_1 is represented by the TCPN module at the second row of Fig. 4. The leakage coefficients δ and α depend on the technology node and the type of subcircuit (SRAM, logic) modeled by each element. Our model allows setting both coefficients for each element separately, honoring a precise floorplan if required. However, since our scheduling technique is unaffected by the magnitude of their values, we assume in this paper an average leakage behavior for every element, setting $\delta = 0.1$ and $\alpha = 0.001$ [Ahmed et al. 2016]. The marking of place p_1^{com} represents the average temperature of the solid component.

4.3.3. Integrating conduction, convection and heat generation. The module *Thermal model* in the TCPN, at the bottom of Fig. 2a, depicts the thermal model of a prism component which generates heat due to the dynamic power of job execution and transfers heat by conduction and convection to its surrounding elements. The model is composed of the basic models of Figs. 3 and 4, by merging the corresponding places of the basic TCPNs that represent the temperature of the component. The component in Fig. 2a is linked to three other surrounding components (partially represented in the figure) through shadowed (boundary) transitions. By the principle of superposition of a thermal system, the merged model is dynamically equivalent to merging the Petri net places in the TCPN [Desirena-Lopez et al. 2014]. The number of prismatic thermal components depends on the desired balance between accuracy and computing time.

4.4. Global TCPN model

The TCPN submodels for tasks, CPUs and thermal dynamics, combine into a global model by merging specific shared places and transitions (*boundary nodes*) and defining arcs accordingly. Fig. 2a shows a detailed part of the TCPN global model, whereas Fig. 2b outlines the integration of the submodels into the single global TCPN model. Each CPU in (b) links to a set of nine components (modeling nine prisms), by way of example. The *TCPN Thermal Model* in (a) shows the net of each component enclosed in a dotted rounded box. The arcs from places p_i^{cc} to transitions $t_{i,j}^{alloc}$ represent the jobs of τ_i allocated to processor CPU_j. Places $p_{i,j}^{exec}$ and arcs going from $t_{i,j}^{exec}$ to $p_{i,j}^{exec}$ are added in order to merge the models. The marking of place $p_{i,j}^{exec}$ stands for the total amount of jobs of τ_i that have been executed in CPU_j from the initial time.

Modeling the power consumption due to task execution implies adding weighting arcs from transitions $t_{i,j}^{exec}$ to places $p_{1,\ldots,k}^{com_j}$ denoting the temperatures of the CPU_j's prismatic components. The flow of transitions $t_{i,j}^{exec}$ (execution flow) stands for the number of CPU cycles by time unit demanded by task τ_i while running on CPU_j. The execution flow ($f_{i,j}^{exec}$) multiplied by the weight $\frac{e_i \times V_1^j}{cc_i \times V_{CPU_j}}$ equals the power generation at each prismatic component corresponding to CPU_j when the task τ_i is running.

The global TCPN model scales up easily by adding a new task or CPU submodule per every additional task or CPU, with the calculations being limited by the computational complexity. The thermal model is unaffected by the number of tasks or CPUs: it only depends on the degree of required detail, translated into more or fewer components (prisms).

4.5. Fundamental equation of the global model

The global model of Fig. 2 is fully described by its Petri net (graph) structure, firing rates and initial marking. Upon this information, the system can be simulated by using the fundamental equation (2). For convenience, we assume that the places are numbered in such a way that $\boldsymbol{m} = [\boldsymbol{m}_T^T, \boldsymbol{m}_a^T, \boldsymbol{m}_\tau^T, \boldsymbol{m}_\mathcal{P}^T, \boldsymbol{m}_{exec}^T]^T$, where $\boldsymbol{m}_T$, $\boldsymbol{m}_a$, $\boldsymbol{m}_\tau$, $\boldsymbol{m}_\mathcal{P}$ and $\boldsymbol{m}_{exec}$ denote the marking at the places representing temperature at components ($p_k^{com_j}$), air temperature (p^{air}), places belonging to the task submodel (p_i^ω, p_i^{cc}, p_i^d), processor sub-model ($p_{i,j}^{busy}$, p_j^{idle}) and places representing accumulated executed jobs ($p_{i,j}^{exec}$), respectively. Thus, the fundamental equation (2) can be represented by blocks as

$$\dot{\boldsymbol{m}}_{\boldsymbol{T}} = \boldsymbol{C}_{\boldsymbol{T}} \boldsymbol{\Lambda}_{\boldsymbol{T}} \boldsymbol{\Pi}_{\boldsymbol{T}}(\boldsymbol{m}) \boldsymbol{m}_{\boldsymbol{T}} + \boldsymbol{C}_{\boldsymbol{a}} \boldsymbol{\Lambda}_{\boldsymbol{a}} \boldsymbol{\Pi}_{\boldsymbol{a}}(\boldsymbol{m}) \boldsymbol{m}_{\boldsymbol{a}} + \boldsymbol{C}_{\boldsymbol{\mathcal{P}}}^{\boldsymbol{exec}} \boldsymbol{f}^{\boldsymbol{exec}} \tag{6a}$$

$$\dot{\boldsymbol{m}}_{\boldsymbol{a}} = 0 \tag{6b}$$

$$\dot{\boldsymbol{m}}_{\boldsymbol{\mathcal{T}}} = \boldsymbol{C}_{\boldsymbol{\mathcal{T}}} \boldsymbol{\Lambda}_{\boldsymbol{\mathcal{T}}} \boldsymbol{\Pi}_{\boldsymbol{\mathcal{T}}}(\boldsymbol{m}) \boldsymbol{m}_{\boldsymbol{\mathcal{T}}} - \boldsymbol{C}_{\boldsymbol{\mathcal{T}}}^{\boldsymbol{alloc}} \boldsymbol{w}^{\boldsymbol{alloc}} \tag{6c}$$

$$\dot{\boldsymbol{m}}_{\boldsymbol{\mathcal{P}}} = \boldsymbol{C}_{\boldsymbol{\mathcal{P}}} \boldsymbol{\Lambda}_{\boldsymbol{\mathcal{P}}} \boldsymbol{\Pi}_{\boldsymbol{\mathcal{P}}}(\boldsymbol{m}) \boldsymbol{m}_{\boldsymbol{\mathcal{P}}} + \boldsymbol{C}_{\boldsymbol{\mathcal{P}}}^{\boldsymbol{alloc}} \boldsymbol{w}^{\boldsymbol{alloc}} \tag{6d}$$

$$\dot{\boldsymbol{m}}_{\boldsymbol{exec}} = \boldsymbol{C}_{\boldsymbol{\mathcal{P}}}^{\boldsymbol{exec}} \boldsymbol{f}^{\boldsymbol{exec}} \tag{6e}$$

The differential equations underlying the Thermal TCPN submodel are represented in Eqs. (6a) and (6b).

The matrices $\boldsymbol{C_T}$, $\boldsymbol{\Lambda_T}$, and $\boldsymbol{\Pi_T}(\boldsymbol{m})$ are the incidence matrix, the firing rate transitions and the configuration matrix respectively of the thermal module in Fig. 2, corresponding to the prismatic components of the floorplan. The matrices $\boldsymbol{C_a}$, $\boldsymbol{\Lambda_a}$, and $\boldsymbol{\Pi_a}(\boldsymbol{m})$ represent the incidence matrix, the firing rate transitions and the configuration matrix respectively, for the ambient temperature of the thermal subnet.

Task arrival is given by Eq. (6c). The matrices $\boldsymbol{C}_{\boldsymbol{\mathcal{T}}}$, $\boldsymbol{\Lambda}_{\boldsymbol{\mathcal{T}}}$, and $\boldsymbol{\Pi}_{\boldsymbol{\mathcal{T}}}(\boldsymbol{m})$ are the incidence matrix, the firing rate transitions and the configuration matrix corresponding to the subnet of the task model in Fig. 2 modules task model.

The behavior of a CPU is modeled by Eq. (6d). The matrices $\boldsymbol{C_{\mathcal{P}}}$, $\boldsymbol{\Lambda_{\mathcal{P}}}$, and $\boldsymbol{\Pi_{\mathcal{P}}(m)}$ represent the incidence matrix, the firing rate transitions and the configuration matrix corresponding to the CPU models.

The matrices $\boldsymbol{C_{\mathcal{P}}^{exec}}$, $\boldsymbol{C_{\mathcal{T}}^{alloc}}$ and $\boldsymbol{C_{\mathcal{P}}^{alloc}}$ represent the connections of transitions $t_{i,j}^{exec}$ and $t_{i,j}^{alloc}$ from (to) places in the thermal model, task arrival model and CPU model, respectively. Fig. 2b shows the connection of the aforementioned models, by means of the allocation transitions $t_{i,j}^{alloc}$ and the execution transitions $t_{i,j}^{exec}$. The task model does not depend on the temperature model or the CPU model but on $\boldsymbol{w^{alloc}}$ (Eq. 6c). The vector $\boldsymbol{w^{alloc}}$ represents the jobs allocated per time unit. These jobs are removed from the task submodels and allocated to the processors for execution. In addition, the CPU model evolves independently from the thermal model (Eq. 6a). The marking $\boldsymbol{m_{exec}}$, i.e., the accumulated executed jobs, is the integral of $\boldsymbol{f^{exec}}$ (Eq. 6e).

Note that Eq. (6) is in a state space form, thus being able to design controllers. The actual input control is the vector $\boldsymbol{w^{alloc}}$. It represents the necessary flow going through transitions $\boldsymbol{t^{alloc}}$, i.e. the allocation of tasks to CPUs to meet temporal constraints. The controlled variables are $\boldsymbol{m_{exec}}$ and $\boldsymbol{m_{T}}$. The former represents the accumulated task execution, it must be equal to the required task execution over time, the latter represents the CPU's temperature.

The proposed scheduling algorithms will act on the model by determining when transitions $t_{i,j}^{alloc}$ must be fired. In particular, we will define a *controlled flow vector* $\boldsymbol{w^{alloc}} = [{}_1w_1^{alloc}, \ldots, {}_1w_n^{alloc}, \ldots, {}_mw_n^{alloc}]$ (where each ${}_jw_i^{alloc}$ represents the controlled flow of transition $t_{i,j}^{alloc}$), it will define the allocation of jobs to CPUs, influencing the dynamical behavior of the global model, and causing temperature to raise or decrease accordingly.

5. HRT THERMAL-AWARE FLUID SCHEDULE FEASIBILITY

We prove that if a feasible schedule exists, then a set-point for the controlled variable $\boldsymbol{m_{exec}}$ exists that satisfies simultaneously the thermal and temporal constraints.

Definition 5.1. A schedule is *thermal feasible* if it satisfies the required deadlines every hyperperiod and the temperature of the processors do not exceed a maximum operating value $\boldsymbol{T_{max}} = [\boldsymbol{T_{max_1}}, \ldots, \boldsymbol{T_{max_m}}]$. Moreover, since the schedule is periodic, the temperature of the processors must satisfy that temperatures at the start and at the end of the hyperperiod are equal.

The feasibility analysis is accomplished in three parts. First, we present the temporal and CPU utilization restrictions. Second, we derive the thermal constraints. Both restrictions are described as linear functions of certain coefficients ${}_j\beta_i$ denoting fractions of task executions. Third, we pose a LPP to computate the fraction of each task job to be run at each CPU, subject to the constraints mentioned above, whose solution provides the values of the coefficients ${}_j\beta_i$. The continuous controller in Section 6 uses these coefficients.

5.1. Computation of the temporal fluid-schedule functions

The task *fluid* schedule function is computed as:

$$FSC_{\tau_i}(\zeta) = \frac{c_i}{\omega_i}\zeta \qquad (7)$$

where ζ is the current time. This function represents the optimal *fluid* execution of task τ_i at time ζ [Baruah et al. 1996] [Zhu et al. 2003]. Eq. (7) is defined as optimal in the literature, although it just provides a feasible schedule for task τ_i. We obtain a *fluid* schedule function for each pair CPU_j and task τ_i as follows.

Assume that $s_\tau = \tau_i^1 \ldots \tau_h^q$ is a sequence of jobs, and $\mathcal{T}'$ the set of tasks in the sequence s_τ, i.e., $\mathcal{T}' = \{\tau_a \in \mathcal{T} \mid \exists\ \tau_a^b \in s_\tau\}$. Now, suppose that the jobs in s_τ can be executed in any CPU. Since migration is possible, a single job $\tau_a^b \in s_\tau$ (the $b - th$ execution of τ_a) can be executed in several CPU. If we denote by ${}_jcc_a^b$ the CPU cycles that job τ_a^b runs in CPU_j, the total number of cycles that the $b - th$ execution of τ_a takes running in all the CPUs is $cc_a = \sum_{j=1}^{m} {}_jcc_a^b$.

Moreover, if the sequence s_τ is a periodic schedule with period H, then the number of instances of task τ_a in s_τ is $i_a = \frac{H}{\omega_a}$. The number of CPU cycles that task τ_a executes in CPU_j during the hyperperiod is ${}_jcc_a = \sum_{r=1}^{i_a} {}_jcc_a^r = {}_j\beta_a \times cc_a$. Thus ${}_jcc_a$ is a proportion of cc_a, where ${}_j\beta_a$ is the number of times that τ_a is executed on CPU_j until the hyperperiod (always a real positive). Hence, the number of jobs of τ_a executed during the hyperperiod of a periodic sequence s_τ can be computed as:

$$i_a = \sum_{j=1}^{m} {}_j\beta_n = \frac{H}{w_a} \qquad \forall\ \tau_a\ \in \mathcal{T} \qquad\qquad [\textit{Temporal Constraint}] \tag{8}$$

In order to fulfill task temporal requirements, the CPU_j utilization must not exceed the processors capacity:

$$\sum_{\tau_i \in \mathcal{T}} \frac{c_i \times {}_j\beta_i}{H} \leq 1 \qquad \forall\ CPU_j\ \in \mathcal{P} \qquad\qquad [\textit{CPU utilization Constraint}] \tag{9}$$

where $c_i = \frac{cc_a}{F}$ is the respective total execution time of τ_i.

The fluid execution of a task (FSC_{τ_i}) can be derived from the previous equations. Since $cc_a = \sum_{j=1}^{m} {}_jcc_a^b$, then $cc_a \times i_a = \sum_{j=1}^{m} \sum_{r=1}^{i_a} {}_jcc_a^r$ represents the CPU cycles during the hyperperiod. Thus, at the hyperperiod H, the total execution time $c_a \times i_a = c_a \times \sum_{j=1}^{m} {}_j\beta_a$, and it follows that $c_a \times \left(\frac{H}{\omega_a}\right) = \sum_{j=1}^{m} {}_j\beta_a \times c_a$. By dividing the last expression by H, we obtain $\frac{c_a}{\omega_a} = \sum_{j=1}^{m} \frac{{}_j\beta_a \times c_a}{H}$. Generalizing, for a task τ_i, $FSC_{\tau_i}(\zeta)$ can be expressed as:

$$FSC_{\tau_i}(\zeta) = {}_1FSC_{\tau_i}(\zeta) + \ldots + {}_mFSC_{\tau_i}(\zeta) = \frac{{}_1\beta_i \times c_i}{H}\zeta + \ldots + \frac{{}_m\beta_i \times c_i}{H}\zeta \tag{10}$$

where ${}_jFSC_{\tau_i}(\zeta) = \frac{{}_j\beta_i \times c_i}{H}\zeta$ stands for the fluid schedule function of τ_i at time ζ in CPU_j. Eqs. (8) and (9) provide the temporal restrictions for computing the ${}_j\beta_a$ coefficients.

5.2. Thermal Analysis

The relationship between task allocation and temperature was formulated in Section 4.5 using Eqs. (6a) and (6d). Eq. (6d) can be rewritten by applying the following change of variable. Let $\boldsymbol{M_T^1 = m_T}$ and $\boldsymbol{M_T^2 = m_{\mathcal{P}}}$, thus $\boldsymbol{M_T = [M_T^1\ M_T^2]^T}$ denotes the state variables corresponding to the thermal behavior of processors and the dynamic task allocation respectively. Hence, the part of the model that represents the relationship between the temperature and the power consumption due to task alloca-tion is:

$$\begin{array}{l} \dot{\boldsymbol{M}}_{\boldsymbol{T}} = \boldsymbol{A}\boldsymbol{M}_{\boldsymbol{T}} + \boldsymbol{B}\boldsymbol{w}^{\boldsymbol{alloc}} + \boldsymbol{B}'\boldsymbol{m}_{\boldsymbol{a}} \\ \boldsymbol{Y}_{\boldsymbol{T}} = \boldsymbol{S}'\boldsymbol{M}_{\boldsymbol{T}} \end{array} \tag{11}$$

where $\boldsymbol{A}$ corresponds to the system matrix, $\boldsymbol{B}$ is the input matrix, and $\boldsymbol{B}'$ conforms the matrix associated to ambient temperature (m_a which is considered constant). These matrices are:

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{C}_{\boldsymbol{T}}\boldsymbol{\Lambda}_{\boldsymbol{T}}\boldsymbol{\Pi}_{\boldsymbol{T}} & \boldsymbol{C}^{\boldsymbol{exec}}_{\boldsymbol{\mathcal{P}}}\boldsymbol{\Lambda}^{\boldsymbol{exec}}\boldsymbol{\Pi}^{\boldsymbol{exec}} \\ \boldsymbol{0} & \boldsymbol{C}_{\boldsymbol{\mathcal{P}}}\boldsymbol{\Lambda}_{\boldsymbol{\mathcal{P}}}\boldsymbol{\Pi}_{\boldsymbol{\mathcal{P}}} \end{bmatrix} \boldsymbol{B}' = \begin{bmatrix} \boldsymbol{C}_{\boldsymbol{a}}\boldsymbol{\Lambda}_{\boldsymbol{a}}\boldsymbol{\Pi}_{\boldsymbol{a}}\boldsymbol{m}_{\boldsymbol{a}} \\ \boldsymbol{0} \end{bmatrix} \boldsymbol{B} = \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{C}^{\boldsymbol{alloc}}_{\boldsymbol{\mathcal{P}}} \end{bmatrix} \tag{12}$$

As stated before, vector $\boldsymbol{w}^{\boldsymbol{alloc}}$ is the controlled flow of the *allocation transitions* (it stands for the task allocation rate to CPUs). The task allocated to CPU ($m_{\mathcal{P}}$) times $\boldsymbol{C}^{\boldsymbol{exec}}_{\boldsymbol{\mathcal{P}}}\boldsymbol{\Lambda}^{\boldsymbol{exec}}\boldsymbol{\Pi}^{\boldsymbol{exec}}(\boldsymbol{m})$ represents the dynamic computational power of the tasks under execution. The matrix $\boldsymbol{S}' = [\boldsymbol{S}\ \boldsymbol{0}]$ is the output matrix which selects the components representing the temperature of the CPUs, and $\boldsymbol{Y}_{\boldsymbol{T}}$ stands for the temperature of the components of the processor. The schedule must be periodic from a temporal and thermal point of view. Thus, the initial temperature must be equal to the final temperature (evaluated at the hyperperiod H) to meet the thermal feasibility condition, i.e.: $\boldsymbol{Y}_{\boldsymbol{T}}(\boldsymbol{H}) = \boldsymbol{S}'\boldsymbol{M}_{\boldsymbol{T}}(0)$.

Now, we assume that the processor is running the periodic schedule at a constant rate, then its temperature is a non-decreasing function, and it reaches a steady state condition. In a thermal steady state $\dot{\boldsymbol{M}}_{\boldsymbol{T}} = \boldsymbol{0}$, the steady state temperature ($\boldsymbol{M}_{\boldsymbol{T}_{ss}}$) and the steady state CPUs temperature ($\boldsymbol{Y}_{\boldsymbol{T}_{ss}}$) are computed as:

$$\begin{array}{l} \boldsymbol{M}_{\boldsymbol{T}_{ss}} = -\boldsymbol{A}^{-1}(\boldsymbol{B}\boldsymbol{w}^{\boldsymbol{alloc}} + \boldsymbol{B}'\boldsymbol{m}_{\boldsymbol{a}}) \\ \boldsymbol{Y}_{\boldsymbol{T}_{ss}} = \boldsymbol{S}'\boldsymbol{M}_{\boldsymbol{T}_{ss}} \end{array} \tag{13}$$

The steady state temperature $Y_{T_{ss}}[k]$ of CPU_k must be less than or equal to its maximum temperature level i.e., $Y_{T_{ss}}[k] \leq T_{max_k}$ so as not to violate the thermal constraint. In vectorial form:

$$\boldsymbol{S}'\boldsymbol{M}_{\boldsymbol{T}_{ss}} \leq \boldsymbol{T}_{\boldsymbol{max}} \tag{14}$$

Combining Eqs. (13) and (14),

$$-\boldsymbol{S}'\boldsymbol{A}^{-1}\boldsymbol{B}\boldsymbol{w}^{\boldsymbol{alloc}} \leq \boldsymbol{T}_{\boldsymbol{max}} + \boldsymbol{S}'\boldsymbol{A}^{-1}\boldsymbol{B}'\boldsymbol{m}_{\boldsymbol{a}} \qquad [\textit{Thermal constraint}] \tag{15}$$

The previous equation provides the thermal constraints that the allocation of tasks to the processors ($\boldsymbol{w}^{\boldsymbol{alloc}}$) must fulfill. Now, $\boldsymbol{w}^{\boldsymbol{alloc}}$ must be represented as a function of ${}_j\beta_i$ parameters. In steady state, the flows in transitions $\boldsymbol{t}^{\boldsymbol{alloc}}$ and $\boldsymbol{t}^{\boldsymbol{exec}}$ are equal. According to the temporal restrictions in Eq. (8), the flow required in $t^{alloc}_{i,j}$ must be $\frac{{}_j\beta_i \times c_i}{H}$, matching the number of jobs of task τ_i assigned to CPU_j per time unit. The next subsection computes the ${}_j\beta_i$ coefficients.

5.3. Computation of coefficients ${}_j\beta_i$

The computation of the fluid execution of tasks can be formulated as a LPP, to deal with the steady state temperature and the *maximum temperature* of each CPU. The problem consists in finding a solution for the ${}_j\beta_i$ subject to thermal, temporal and CPU utilization constraints, which translates into Eq. (16):

$$
\begin{array}{ll}
min \sum_{j=1}^{m} \sum_{\tau_a \in \mathcal{T}} {}_j\beta_a & \\
s.t. & \\
\quad \boldsymbol{-SA^{-1}B} \left[\sum_{\tau_a} \frac{{}_1\beta_a c_a}{H}, \; \cdots \;, \sum_{\tau_a} \frac{{}_m\beta_a c_a}{H} \right]^T \leq \boldsymbol{T_{max}} + \boldsymbol{SA^{-1}B'm_a} \left.\vphantom{\sum}\right\} & \textit{Thermal Constraint} \\
\quad \sum_{j=1}^{m} {}_j\beta_n = \frac{H}{\omega_n} \quad \forall i = 1, \ldots, n \left.\vphantom{\sum}\right\} & \textit{Temporal Constraint} \\
\quad \sum_{\tau_a \in \mathcal{T}} \frac{c_a \times {}_m\beta_a}{H} \leq 1 \quad \forall j = 1, \ldots, m \left.\vphantom{\sum}\right\} & \textit{CPU util. Constraint}
\end{array}
\tag{16}
$$

The thermal constraint in Eq. (15) states that the temperature of the processors due to task execution must not violate the maximum allowed temperature. The estimated temporal constraints, Eqs. (8) and (9) ensure that the number of tasks' jobs within the hyperperiod allows a feasible schedule. The last constraint ensures that the computation utilization of each CPU is ≤ 1.

PROPOSITION 5.2. *Consider a system as defined in 3.3 and the thermal-aware fluid scheduler problem 3.1 for this system. If the linear programming problem LPP (16) defined for the system parameters has a feasible solution, then there exists a thermal-aware HRT feasible schedule.*

If a solution exists, then the coefficients ${}_j\beta_i$ can be found. Hence the ${}_jFSC_{\tau_i}(\zeta) = \frac{{}_j\beta_i \times c_i}{H}\zeta$ functions are completely known, and any scheduler capable of tracking these functions will obtain a schedule that fulfills thermal and temporal task constraints since temporal and thermal restrictions are met.

To show the existence of such a scheduler (and complete the proof), we propose in the next section a continuous scheduler whose result is discretized in a later stage. It uses the functions ${}_jFSC_{\tau_i}(\zeta)$ (target function) and $m_{i,j}^{exec}(\zeta)$ (actual task execution) to define an error as the difference between these two quantities. By controlling task allocation with $\boldsymbol{w^{alloc}}$ the proposed scheduler brings the error down to zero. This is proved in proposition 6.1, where the Lyapunov functions guarantee that the error reaches a zero value.

6. THERMAL-AWARE HRT FLUID SCHEDULER CONTROL

If the LPP has a feasible solution, then each task τ_i in each CPU_j must be executed at the *fluid* execution rate (${}_jFSC_{\tau_i}(\zeta)$) to honor the HRT thermal fluid schedule. Considering $\varphi_{i,j} = \frac{{}_j\beta_i \times c_i}{H}$ the fluid schedule function becomes ${}_jFSC_{\tau_i}(\zeta) = \varphi_{i,j}\zeta$. This function ${}_jFSC_{\tau_i}(\zeta)$ will be used as a set-point for the control stage.

We leverage a sliding mode feedback controller to manage workload execution [Utkin et al. 2009]. The purpose of the controller is to keep the *RT thermal fluid execution error* $\mathcal{E}_{T_{i,j}}(\zeta)$ equal to zero. This error is defined as the difference between the task fluid execution ${}_jFSC_{\tau_i}(\zeta)$ of a task τ_i in CPU_j and its actual execution percentage ($m_{i,j}^{exec}(\zeta)$ in Fig. 2a):

$$\mathcal{E}_{T_{i,j}}(\zeta) = {}_jFSC_{\tau_i}(\zeta) - m_{i,j}^{exec}(\zeta) \tag{17}$$

6.1. RT Thermal Sliding surface

In the sliding mode technique, a sliding surface $\mathcal{S}$ is first designed as a function of the system's state in such way that, if the system is controlled so that $\mathcal{S}$ is null then the error

converges to zero. In order to construct the sliding surface, let $x^1{}_{i,j} = \mathcal{E}_{T_{i,j}}(\zeta)$ and $x^2{}_{i,j} = m^{busy}_{i,j}$. Then, the following system holds:

$$\begin{aligned} \dot{x}^1{}_{i,j} &= \varphi_{i,j} - \lambda^{exec}_{i,j} x^2_{i,j} \\ \dot{x}^2{}_{i,j} &= {}_j w^{alloc}_i - \lambda^{exec}_{i,j} x^2_{i,j} \end{aligned} \tag{18}$$

For this system, the sliding surface may then be set to be of the form:

$$\mathcal{S}_{i,j}(\zeta) = \frac{K_1}{\lambda^{exec}_{i,j}} x^1_{i,j} + \frac{\varphi_{i,j}}{\lambda^{exec}_{i,j}} - x^2_{i,j} \tag{19}$$

where K_1 is a real positive number.

Since the aim is to force the system states to the sliding surface, the control strategy must guarantee that the system trajectory moves toward and stays on the sliding surface from any initial condition. Such control law is described in the following subsection. Once the system slides on the surface $\mathcal{S}_{i,j}(\zeta) = 0$, then

$$x^2_{i,j} = -\frac{K_1}{\lambda^{exec}_{i,j}} x^1_{i,j} - \frac{\varphi_{i,j}}{\lambda^{exec}_{i,j}} \tag{20}$$

Therefore

$$\dot{x}^1{}_{i,j} = -K_1 x^1_{i,j} \tag{21}$$

In other words, the *RT thermal error fluid execution* tends to zero asymptotically when the system slides on the surface.

6.2. Control law computation

Now, a control law is designed to force the system to slide on the surface, which will lead to a null error and thus the system will track the *fluid* schedule function of each task τ_i and CPU_j, meeting both temporal and thermal requirements. The control law is proposed as:

$${}_j w^{alloc}_i(\zeta) = {}_j \hat{w}^{alloc}_i(\zeta) + \tfrac{K_1}{\lambda^{exec}_{i,j}} \varphi_{i,j} \tag{22}$$

where ${}_j \hat{w}^{alloc}_i(\zeta) = K_2 sign(\mathcal{S}_{i,j}(\zeta))$ and $sign(x) = 1$ if $x \geq 0$; 0 otherwise.

PROPOSITION 6.1. *Let $\mathcal{T}$ and $\mathcal{P}$ be the sets of n tasks and m processors, respectively. Let ${}_j FSC_{\tau_i}$ be fluid schedule function of task τ_i and CPU_j, obtained by solving the linear programming problem of Eq. (16). If the control law given by Eq. (22) is applied to the global system with $K_1 = \lambda^{exec}_{i,j}$ and $0 < K_2 < \varphi_{i,j}$ then each RT thermal fluid execution error $\mathcal{E}_{T_{i,j}}(\zeta)$ converges to zero.*

PROOF. The controlled flow of a transition $t^{alloc}_{i,j}$ is given by ${}_j w^{alloc}_i(\zeta)$ in Eq. (22), where ${}_j \hat{w}^{alloc}_i(\zeta)$ is the control action. Note that when ${}_j w^{alloc}_i > 0$, transition $t^{alloc}_{i,j}$ is fired, i.e., jobs of τ_i are being allocated to CPU_j.

In order to prove the asymptotic stability of Eq. (17), a Lyapunov function can be defined, satisfying $V(0) = 0$, $V(x) > 0$ and $\dot{V}(x) < 0 \; \forall x \neq 0$ ([Khalil and Grizzle 1996]). Let us consider the following quadratic candidate Lyapunov function V:

$$V(\mathcal{S}_{1,1}, \ldots, \mathcal{S}_{n,m}) \;=\; \tfrac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{m} \mathcal{S}_{i,j}^{2} \tag{23}$$

$V = 0$ ***iff*** each $\mathcal{S}_{i,j} = 0$, and $V > 0$ if any $\mathcal{S}_{i,j} \neq 0$. Therefore, V can be considered a Lyapunov function (and Eq. (17) is asymptotic stable) ***iff*** $\overset{\bullet}{V} < 0$ for any $\mathcal{S}_{i,j} \neq 0$. To prove this, we first compute the derivative of V:

$$\begin{aligned}
\overset{\bullet}{V} \;&=\; \sum_{i=1}^{n} \sum_{j=1}^{m} \mathcal{S}_{i,j}\overset{\bullet}{\mathcal{S}}_{i,j} = \sum_{i=1}^{n} \sum_{j=1}^{m} \mathcal{S}_{i,j}\left(-{}_j\hat{w}_i^{alloc} + x_{i,j}^{2}(\lambda_{i,j}^{exec} - K_1)\right) \\
&\leq\; \sum_{i=1}^{n} \sum_{j=1}^{m} -K_2\mathcal{S}_{i,j}\,sign(\mathcal{S}_{i,j}) + |\mathcal{S}_{i,j}||x_{i,j}^{2}||\lambda_{i,j}^{exec} - K_1| \\
&\leq\; \sum_{i=1}^{n} \sum_{j=1}^{m} -|\mathcal{S}_{i,j}|\left(K_2 - |x_{i,j}^{2}||\lambda_{i,j}^{exec} - K_1|\right)
\end{aligned} \tag{24}$$

Next, let us prove that following holds for each term in the sum (i.e. for each task and CPU) $-|\mathcal{S}_{i,j}|\left(K_2 - |x_{i,j}^{2}||\lambda_{i,j}^{exec} - K_1|\right) < 0$. The term is negative if K_2 satisfies $K_2 > |x_{i,j}^{2}||\lambda_{i,j}^{exec} - K_1|$. Since the controlled flow is always positive (i.e., ${}_jw_i^{alloc} = K_2 sign(\mathcal{S}_{i,j}) + \frac{K_1}{\lambda_{i,j}^{exec}}\varphi_{i,j} > 0$), then if $\mathcal{S}_{i,j}$ is positive, K_2 must satisfy $K_2 > -\frac{K_1}{\lambda_{i,j}^{exec}}\varphi_{i,j}$, otherwise (when $\mathcal{S}_{i,j}$ is negative) K_2 must satisfy $K_2 < \frac{K_1}{\lambda_{i,j}^{exec}}\varphi_{i,j}$.

Therefore, in order to satisfy the stability condition ($\overset{\bullet}{V} < 0$), K_2 must satisfy $|x_{i,j}^{2}||\lambda_{i,j}^{exec} - K_1| < K_2 < \frac{K_1}{\lambda_{i,j}^{exec}}\varphi_{i,j}$. For simplicity, we assume that $K_1 = \lambda_{i,j}^{exec}$, hence $0 < K_2 < \varphi_{i,j}$. Thus, for each τ_i and CPU_j the ***RT thermal fluid execution error*** $\mathcal{E}_{T_{i,j}}(\zeta)$ converges to zero asymptotically. □

The correct selection of the gains K_1 and K_2 for the derived control law provided by Eq. (22) allows tracking the optimal fluid schedule for each τ_i and CPU_j, which is thermally feasible. Therefore, the fluid execution $\boldsymbol{m_{exec}}$ follows a fluid schedule.

6.3. On-line discretization of the Thermal-Aware fluid schedule

The Alg. 1 (*OLDTFS*) implements our thermal-aware RT fluid scheduler, aiming to balance a good thermal control and a suitable context-switch overhead. We leverage the approach taken in [Desirena-Lopez et al. 2016], which limits the fluid schedule computation to the set of deadlines, but places the scheduling points on a quantum basis. Later we will show that under precise circumstances it is possible to limit the scheduling points to the set of deadlines as well, and still accomplish the RT and thermal constraints without requiring a fixed quantum, all the more lowering the overhead. *OLDTFS* yields a discrete schedule that closely tracks the fluid one ($\boldsymbol{m_{exec}}$) by computing a schedule up to the hyperperiod (from time zero to time H). We first define the set of deadlines SD and quantum Q as in [Desirena-Lopez et al. 2016].

For every time interval $[sd_k,\ sd_{k+1}]$ ($sd_{k+1} \leq H$ and $k =, 0, 1, \ldots$), at time ζ, if ${}_jFSC_{\tau_i}(sd_k) > M_{i,j}^{exec}(\zeta)$, τ_i must be allocated to CPU_j so that it runs to the point required by the fluid scheduler, to warrant that the k-th job of τ_i completes before its k-th deadline. The thermal-compliant HRT fluid schedule (${}_jFSC_{\tau_i}(sd_k)$) was computed in the off-line stage. We define the *remaining jobs execution* until the next deadline in the whole set of deadlines as $RE_{i,j}(\zeta) = {}_jFSC_{\tau_i}(sd_k) - M_{i,j}^{exec}(\zeta)$ for each task τ_i and CPU_j (line 4), and the *task priority function* as $PR_{i,j}(\zeta) = m_{i,j}^{exec}(\zeta) - M_{i,j}^{exec}(\zeta)$ (line 6). All tasks τ_i such that $RE_{i,j}(\zeta) > 0$ must be allocated to CPU_j before the next quantum.

ALGORITHM 1: On-line discretization of Thermal fluid schedule (OLDTFS)

```
   Input: The TCPN of the set of tasks $\mathcal{T}$, the ordered set $SD$ where any $sd_k \in SD$ is lower or equal than
          $H$. The quantum $Q$. The fluid schedule ${}_jw_i^{alloc}$. The number of processors $m$
   Output: The discrete schedule ${}_jW_i^{alloc}$
1  Initialize $i = 1$, $sd = sd_i$, $\zeta = 0$, $M_{i,j}^{exec}(\zeta) = 0\ \forall \tau_i \in \mathcal{T}$ and $\forall CPU_j \in \mathcal{P}$;

2  for $\zeta \leq H$ do
3      All tasks are preempted from the processors;
4      $RE_{i,j}(\zeta) = {}_jFSC_{\tau_i}(sd) - M_{i,j}^{exec}(\zeta)$ ;                        /* Compute remaining jobs */
5      $ET_j(\zeta) = \{\tau_i | RE_{i,j}(\zeta) > 0 \forall CPU_j \in \mathcal{P}\}$ ;      /* Compute the set of tasks to be executed */
6      $PR_{i,j}(\zeta) = m_{i,j}^{exec}(\zeta) - M_{i,j}^{exec}(\zeta)$ ;  /* Compute the priority for every task $\tau_i$ in $ET_j(\zeta)$ */
7      for $j = 1$ to $m$ do
8          ${}_jW_i^{alloc} = 0, 1 \leq j \leq m,\ 1 \leq i \leq n$ ;
9          $\tau_a$ = task with the highest priority value in $ET_j(\zeta)$;      /* $ET_j(\zeta)$: task queue of $CPU_j$ */
10         ${}_jW_a^{alloc} = 1$ ;                                                /* Set $\tau_a$ to run on $CPU_j$ */
11         ;/* Now remove $\tau_a$ from all tasks queues but $ET_j(\zeta)$:                                  */
12         Remove $\tau_a$ from $ET_k(\zeta)$ for all $1 \leq k \leq m$ and $k \neq j$ ;
13         $M_{a,j}^{exec}(\zeta + Q) = M_{a,j}^{exec}(\zeta) + Q \times {}_jW_a^{alloc}$ ;  /* Compute the discrete execution of $\tau_a$ */
14         Remove $\tau_a$ from $ET_j$;
15         Switch to $\tau_a$ in $CPU_j$ ;      /* Only if scheduling tasks in a real, physical system */
16     end
17     Simulate the CPU TCPN model from $\zeta$ to $\zeta + Q$;      /* Solve Eqs. (6.e) to compute $m^{exec}$ */
18     $\zeta = \zeta + Q$;                                                                  /* Update time */
19     if $\zeta == sd$ then
20         $i = i + 1, sd = sd_i$
21     end
22 end
```

The discretized time that task τ_i must run on CPU_j starting at time $\zeta + Q$ is given by the following equation (line 13):

$$M_{i,j}^{exec}(\zeta + Q) = M_{i,j}^{exec}(\zeta) + Q \times {}_jW_i^{alloc} \tag{25}$$

where

$${}_jW_i^{alloc} = \begin{cases} 1 & \text{if } \tau_i \text{ is allocated for execution in } CPU_j \text{ at time } \zeta \\ 0 & \text{otherwise} \end{cases}$$

Thus, Eq. (25) yields the execution time slice of each task job in a CPUs when dispatched for the next interval. We can leverage the TCPN to entirely simulate a physical system, or to exclusively compute the fluid schedule, discretized on a quantum basis in Alg. 1. In the first case, besides accounting for the runtime, we have to actually dispatch the task on a specific CPU (statement in line 15).

6.4. Overview of the Thermal-Aware Real-Time scheduler

Fig. 5 summarizes the three parts of the proposed scheduler higlighting the system control signals. The dotted boxes above and below (A, B) exemplify the evolution of the principal system's input and output signals during normal operation (A) or under disturbance (such as a CPU detention, B). During the off-line stage, the scheduler is build up as described in Sec. 4, from the TCPN model, according to the thermal parameters of the materials of the system, the number of prisms, and the parameters of the CPUs and HRT task set, obtaining its state equation. Then, the constraints of the LPP are stated from this model, and the LPP solution provides the coefficients for the fluid scheduling functions ${}_jFSC_{\tau_i}(\zeta)$ (S1), which guarantee the accomplishment of the thermal and HRT constraints in H (hyperperiod). All this process has been fully

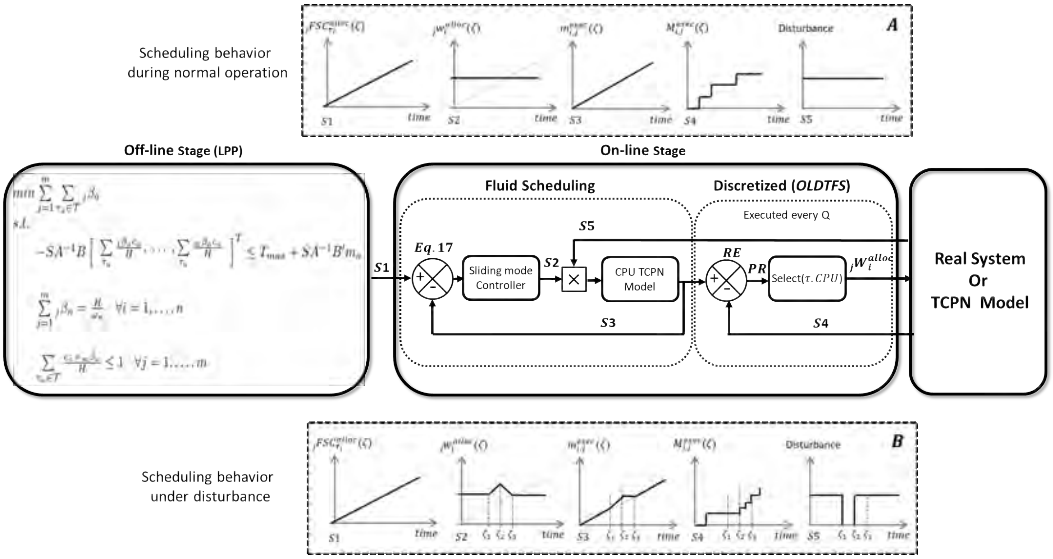


Fig. 5: Scheduler overview. To simplify the view we respectively name ${}_jFSC_{\tau_i}(\zeta)$, ${}_jw_i^{alloc}$, $m_{i,j}^{exec}$ and $M_{i,j}^{exec}$ as the signal arrays $S1, S2, S3, S4$. $S5$ provides the feedback to capture disturbances.

automated, as part of a publicly available simulation framework [Desirena et al. 2019] that includes a number of schedulers that can be simulated out-of-the-box, including the one presented in this paper. Also, this package provides tools to manually or automatically generate RT task sets for simulation, and plots scheduling results and temperature variations.

The on-line stage starts with the sliding mode controller allocating tasks to CPUs in the TCPN model. The controller throttles the fluid marking ${}_jw_i^{alloc}$ (S2), which is proportional to the error $\mathcal{E}_{T_{i,j}}(\zeta)$ (Eq. (17)), to make S3 (the current fluid schedule, $m_{i,j}^{exec}$ in the TCPN) follow S1 (the fluid schedule calculated off-line), closing $\mathcal{E}_{T_{i,j}}(\zeta)$. During normal operation S3 will always follow S1 (upper dotted box A in Fig. 5). Alg. 1 iterates every quantum Q. It computes the error as the difference between the fluid execution time S3 (provided by the TCPN model) and the actual discrete execution time S4 (Alg. 1 line 4), dynamically adjusts priorities accordingly (line 6) and selects, allocates and dispatches the jobs until the next scheduling point (line 18).

The flow of transitions $t_{i,j}^{exec}$ (representing the active execution of task τ_i on CPU_j, Fig. 2) is computed as $f_{i,j}^{exec} = (\lambda_{i,j}^{exec} m_{i,j}^{busy}) \times S5$. Under disturbance (e.g.Fig. 5 box B, at time ζ_0) S5 will be set to 0. This will halt the simulated CPU_j in the TCPN model. As a consequence, the controller will increase S2 so that S3 ($m_{i,j}^{exec}$) becomes higher than S1 (${}_jFSC_{\tau_i}(\zeta)$) until ζ_2. By ζ_3, S1 reaches its normal level, and S3 catches up with the S1 line. Then, *OLDTFS* discretizes S3 by increasing the ratio at which the tasks smitten by the CPU halt are dispatched in successive quanta (compare S4 in box A and B). A scheduler lacking a continuous controller like this are not capable of recovering from disturbances. A sliding mode controller is specially suitable when dealing with off-line and on-line signals.

6.4.1. Complexity. The algorithm executes $I = H/Q$ times (H is the hyperperiod), thus the outer loop in Alg. 1 runs I times. The instructions inside this loop run in polynomial time in the size of the number of tasks and CPUs. All the instructions in the inner loop (lines $7-14$) run in polynomial time. Moreover, Eq. (6.e) (Alg. 1 line 15) can be rewritten in a discrete state space representation. Hence the TCPN is simulated in polynomial time, and therefore the algorithm is polynomial too. Note that Eqs. (6a) and (6b) are not computed in the algorithm because they are not required to solve m^{exec}.

6.5. Discretization of the Thermal-Aware fluid schedule by opportunistic *DP-Fair*

Our methodology allows establishing the circumstances under with we can implement the thermal-aware RT scheduler without a fixed quantum, limiting the scheduling points (i.e. possible context-switches and migrations) to the set of all tasks' deadlines, still ensuring a proper thermal control.

If the linear programming problem in Eq. (16) has a solution, it means that the coefficients ${}_j\beta_i$ of the solution meet the thermal, temporal and CPU utilization constraints in a steady-state temperature. These coefficients can now be used to develop an algorithm based on *DP-Fair*. However, in contrast with the deadline partitioning technique, based on the task local utilization, we leverage the coefficients ${}_j\beta_i$ as the task share that must run on each CPU for each time slice, defined by all the deadlines of all tasks in the system. The algorithm is opportunistic in that it prioritizes the thermal constraint and makes the processors to become idle when processors' temperature approaches the temperature limit. When resuming execution, the algorithm catches up and meet the temporal constraints.

7. SIMULATION RESULTS

The following experiments show the ability of the proposed scheduler to meet HRT and thermal constraints and to deal with disturbances. Also, we compare *OLDTFS* with a straightforward implementation of a *DP-Fair* scheduler [Chandra et al. 2001]

7.1. Experimental setup

We consider a package with two homogeneous $1cm \times 1cm$ microprocessors mounted over a $5cm \times 5cm$ copper heat spreader. The microprocessors and the copper heat spreader are respectively $0.5mm$ and $1mm$ thick. The temperature of the surrounding air is fixed and set to 35^oC, in the range of an environment type C according to [ASHRAE 2012]. The convection coefficient of the heat spreader is $h = 0.001\frac{W}{mm^{2o}C}$. The isotropic thermal properties of the materials in the package are shown in Table I. We assume CPUs with caches and speculative mechanisms non-existent or turned-off, and tasks running at a fixed frequency $F = 1GHz$. Thus, WCET, deadline and period time bounds can be stated more accurately. We include scheduling and context switch overhead in the WCET. We have used for the thermal model a mesh of 25×25 prisms for the heat spreader, 5×5 prisms per CPU, totaling 675 prisms.

The off-line and online stages are implemented in MatLab$^{®}$. For the off-line stage, we use the lingprog (*simplex* algorithm) function to solve the LPP (16). We obtain the predicted task execution ($\dot{\boldsymbol{m}}_{\boldsymbol{exec}}$) during the on-line stage with the ODE45 solver. We provide a package publicly available that automates the whole modeling and simulation process [Desirena et al. 2019].

7.2. *OLDTFS* results

The first experiment considers a thermo-hydraulic system. Both the water level in a tank and its temperature are controlled by a certain algorithm, resulting in a periodic task set $\mathcal{T} = \{\tau_1, \tau_2, \tau_3\}$ running on a MPSoC with two cores; the maximum operating temperature of both cores is $T_{max_{1,2}} = 100^oC$. Tasks τ_2 and τ_3 control the level and temperature respectively. Task τ_1 (acquisition task) is used to read the sensors every sample period. We have computed the sample period according to Shannon's theorem.

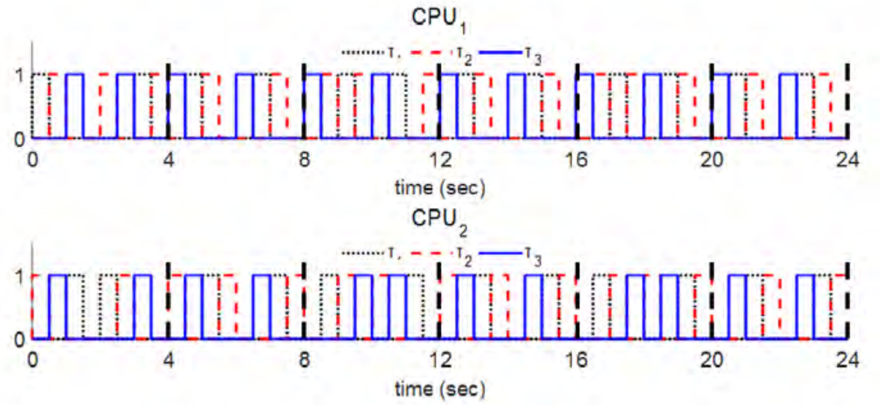


Fig. 6: Schedule computed by *OLDTFS* algorithm for the example of subsection 7.2. Its execution produce a rising in the CPUs temperature depicted in Fig. 7.

The relevant task parameters are WCET (in CPU cycles), sample time ω and deadline d (with $\omega = d$ in this case), resulting in $\tau_1 = (2 \times 10^9, 4, 4, 6.4)$, $\tau_2 = (5 \times 10^9, 8, 8, 8)$, $\tau_3 = (6 \times 10^9, 12, 12, 9.6)$, and the consumed energy e. Task independence is achieved by the correct computation of the sample time. If ω_1 is two or more times shorter than ω_2 and ω_3, then the level and temperature variables are known before the execution of τ_2 and τ_3, i.e. tasks are independent of each other.

Fig. 6 depicts the schedule obtained by the on-line discretization of the fluid temporal scheduler. Fig. 7 shows the evolution of temperature in both CPUs until the hyperperiod, using two different values for the quantum. Temporal and thermal constraints are met with both values, but temperature variations are much narrower with $Q = 0.05$ than with the computed quantum $Q = 0.5$. A theoretical infinitesimal quantum would match the optimal thermal solution with infinitesimal context switches.

A second experiment shows the sensitivity of *OLDTFS* to computation utilization and its ability to cope with a significant number of tasks pushing temperature well over the limit. We leverage the algorithm UUnifast [Bini and Buttazzo 2005] to generate 10^3 task sets with computation utilization varying from 1 to 2 to evaluate the thermal feasibility (only considering task sets which are computationally feasible).

Fig. 8 shows the maximum temperature per CPU reached during the first hyperperiod, while varying the computation utilization (x-axis). The maximum temperature does not only depend on the computational utilization, but on the power consumption of each task set too (see LPP in Eq. (16). *OLDTFS* keeps the maximum temperature under control along the whole range of utilization values. The smaller the quantum, the finer the thermal control at the cost of a higher overhead.

7.3. *OLDTFS* vs. *DP-Fair*

We have seen that *OLDTFS* allows a great control on temperature, but finding a balance to keep a low overhead requires experimenting with different quanta. *DP-Fair* only schedules tasks on the set of deadlines, which generally yelds fewer scheduling points than when using a quantum. However, a baseline *DP-Fair* can fail to keep the temperature inside a safe region. We consider two identical processors and three tasks such that $\mathcal{T} = \{\tau_1, \tau_2, \tau_3\}$, where $\tau_1 = (2 \times 10^9, 4, 4, 6.4)$, $\tau_2 = (5 \times 10, 8, 8, 8)$, $\tau_3 = (6 \times 10, 12, 12, 9.6)$. The CPU utilization is $U = 1.625$, and therefore the scheduling problem has a solution on a two processor platform.

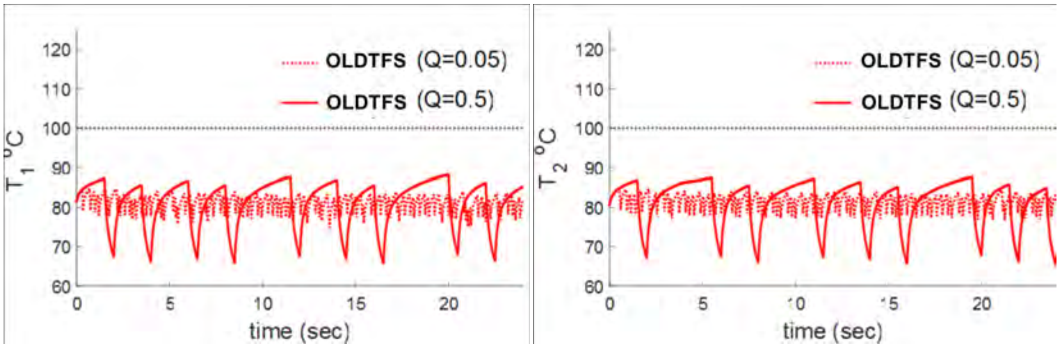


Fig. 7: CPUs temperature evolution due to the execution of the computed schedule in the example of subsection 7.2, for different quantums. The initial conditions of CPU_1 and CPU_2 are $m_T(0)[1] = 81.5664^oC$, $m_T(0)[2] = 80.7352^oC$ respectively. Note that the temperature does not exceed the bound $T_{max} = 100^oC$. As expected, a smaller quantum means a lower temperature variation.

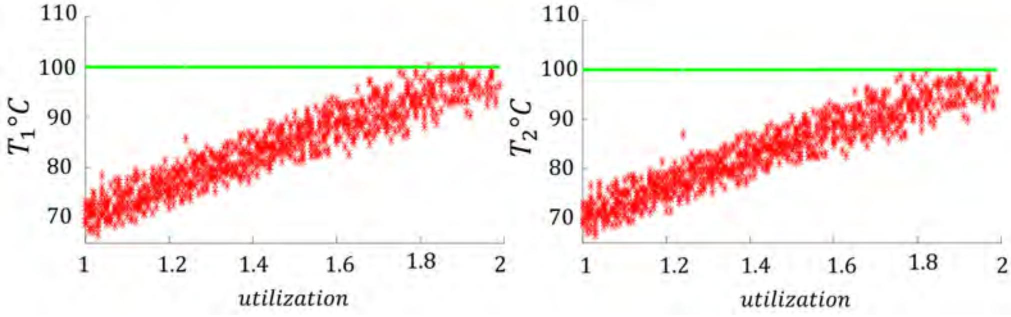


Fig. 8: Maximum temperature of processors considering 10^3 tasks sets, generated by the UniFast algorithm, with utilization varying 1-2. Each marker represents the maximum temperature of a task set with the corresponding utilization.

Both *DP-Fair* and *OLDTFS* compute feasible schedules. Fig. 9 shows the temperature evolution for both schedules. *OLDTFS* evenly distributes the execution and idle time of the CPUs over the scheduling points, keeping temperature under the 90°C bound. In contrast, *DP-Fair* runs all the tasks as soon as possible during the scheduling points, with accrued idle time at the end of each scheduled point, which leads to a temperature violation. The number of context switches and migrations in *DP-Fair* is much lower than in *OLDTFS* nonetheless.

7.4. Thermal-aware Opportunistic *DP-Fair*

Fig. 10 compares the temperature variations yielded by *OLDTFS* with a small quantum and by the thermal-aware opportunistic *DP-Fair* presented in Sec. 6.5. The temperature bound is set to $T_{max} = 100^oC$ for a set of task $\mathcal{T} = \{\tau_1, \tau_2, \tau_3\}$, where $\tau_1 = (2\times10^9, 4, 4, 9.6)$, $\tau_2 = (5\times10^9, 8, 8, 9.6)$, $\tau_3 = (6\times10^9, 12, 12, 9.6)$, and $\mathcal{P} = \{CPU_1, CPU_2\}$. The hyperperiod is $H = 24$ and the CPU utilization is $U = 1.62$. Both algorithms meet the thermal constraint, but *OLDTFS* reaches a lower temperature.

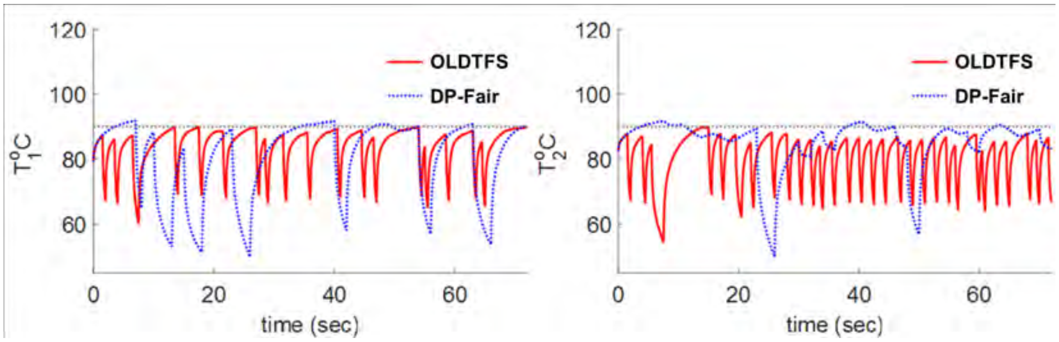


Fig. 9: Temperature evolution in both CPUs for the example in Sec. 7.3. *DP-Fair* produce a temporal feasible schedule, but unlike *OLDTFS* it violates the thermal bound.

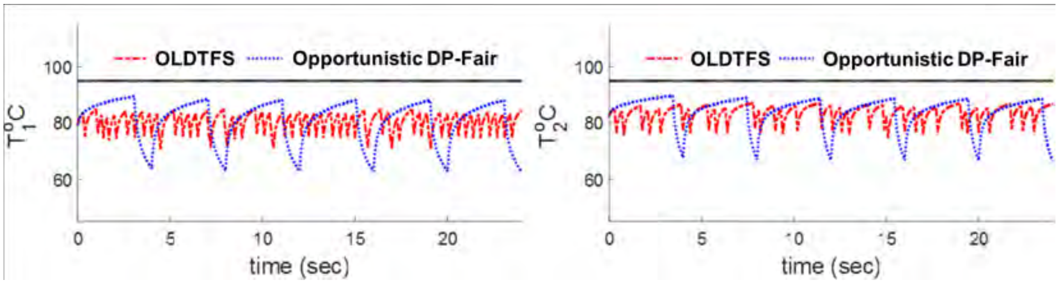


Fig. 10: Temperature evolution for the example in Sect. 7.4 by applying the *OLDTFS* and the opportunist *DP-Fair* algorithm, wich is a blend of the DP-Fair algorithm and the proposed fluid schedules functions. Both algorithms produce a thermal and temporal feasible schedule, but context switches are higher in *OLDTFS*.

7.5. Disturbance recovering with *OLDTFS*

The feedback controller embedded in the scheduler can recover the system from disturbances such as CPU detentions due to hazardous environmental conditions, energy interruptions and others, as we discussed when describing the overall structure and operation of the scheduler (Sec. 6.4, Fig. 5 box B). As a proof of concept, we consider three tasks $\mathcal{T} = \{\tau_1, \tau_2, \tau_3\}$, where $\tau_1 = (2 \times 10^9, 4, 4, 6.4), \tau_2 = (5 \times 10^9, 8, 8, 8), \tau_3 = (6 \times 10^9, 12, 12, 9.6)$ running on two processors. A disturbance causes CPU_1 to halt during the time interval $[5, 7]$, resuming task execution in $\zeta > 7$. Fig. 11 represents the task execution according to our scheduler (Fig. 5). The three plots on the left correspond to the allocation of the three tasks on CPU_1. During the time interval $[5, 7]$, $m^{exec}_{1,i}(\zeta)$ (S3) is unable to track the fluid schedule (${}_1FSC_i(\zeta)$, S3), and appears constant (flat) in the plot. When $\zeta > 7$, the controller starts increasing the task execution rate (without never exceeding a 100% CPU utilization): $m^{exec}_{1,i}(\zeta)$ increases continuously and the discretized schedule ($M^{exec}_{1,i}(\zeta)$, red dots) follows up on a quantum basis.

8. CONCLUSIONS

We have presented a global on-line fluid scheduler that meets both thermal and RT constraints, resorting to a feedback control technique, which offers a number of advantages. First, we can track state variables by leveraging an underlying TCPN that models a HRT set of tasks with implicit deadlines, along with the features of the CPUs in a MPSoC, encompassing power consumption, thermal and HRT behavior. The mod-

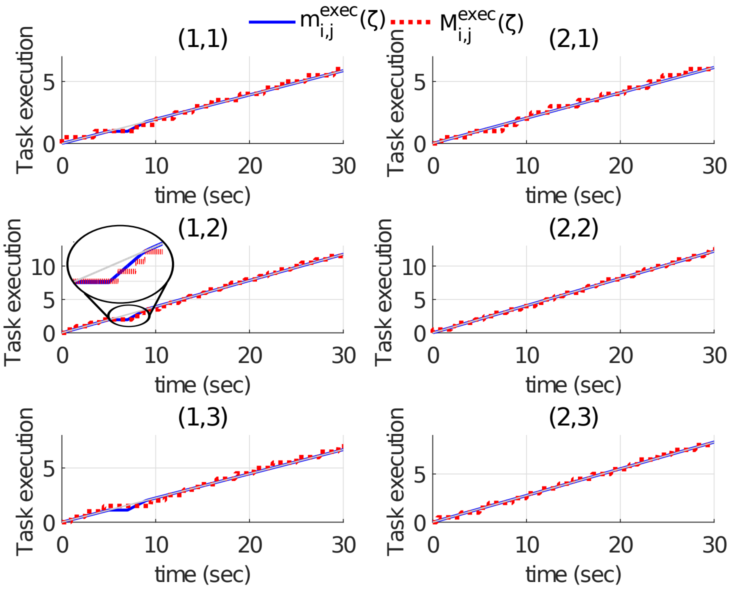


Fig. 11: Task execution of the example presented in Sec. 7.5. A disturbance is introduced during 5-7 s, when the CPU_1 stops task execution due to an external and uncontrollable interruption. Note that when the interruption is released, CPU_1 increases task execution by using the idle spaces; when the fluid schedule function is reached (its reference), then the CPU_1 continues with its normal task execution.

eling methodology is automated by a software application. The thermal schedule feasibility is proved by a LPP that captures HRT and thermal requirements as linear constraints. If there exists a feasible solution, then the LPP solution represents the correct execution of tasks as a continuous linear functions over time (*fluid schedule functions*), honoring thermal constraints. Then, the fluid scheduler is discretized allowing control on context switching and migrations. Second, the feedback controller that implements the proposed global scheduler allows the system to recover from disturbances such as CPU detentions due to environmental hazards causing energy interruptions or thermal peaks. Last, the number of scheduling points can be tuned to react to thermal changes accurately. When it comes to discretize the fluid schedule, deadline partitioning is a good start, and we present a thermal-aware opportunistic *DP-Fair* which can solve the problem as long as the system can support wide temperature variations, always under the limit. However, time slices in *DP-Fair* can be too long for a more accurate thermal control. Introducing a quantum in the implementation algorithm, calculated as the greatest common divisor of the set of deadlines, provides a way of properly tracking temperature, upon the reference set by a deadline partitioning scheme.

Future work includes frequency control to reduce the MPSoC power consumption in thermal-aware schedulers, further heuristics to minimize the number of migrations and context switches, and implementing the proposed scheduler in a RT kernel. Other feedback control characteristics are to be exploited yet, to include aperiodic tasks and slight variations in task parameters.

Acknowledgements

This work was supported by grants TIN2016-76635-C2-1-R (AEI/ERDF, EU), Aragón Government (T58_17R research group) and ERDF 2014-2020 *Construyendo Europa desde Aragon*.

REFERENCES

Rehan Ahmed, Parameswaran Ramanathan, and Kewal K Saluja. 2016. 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 (2016), 49.

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

ASHRAE. 2012. *Thermal guidelines for data Processing Environments*. ASHRAE Datacom Series.

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

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

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

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

Mulugeta K Berhe. 2007. Ergonomic Temperature Limits for Handheld Electronic Devices. In *ASME 2007 InterPACK Conference collocated with the ASME/JSME 2007 Thermal Engineering Heat Transfer Summer Conference*. ASME.

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

Daniel Casini, Alessandro Biondi, and Giorgio Buttazzo. 2017. Semi-Partitioned Scheduling of Dynamic Real-Time Workload: A Practical Approach Based on Analysis-Driven Load Balancing. In *29th Euromicro Conference on Real-Time Systems (ECRTS 2017) (Leibniz International Proceedings in Informatics (LIPIcs))*, Marko Bertogna (Ed.), Vol. 76. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 13:1–13:23. DOI: http://dx.doi.org/10.4230/LIPIcs.ECRTS.2017.13

Abhishek Chandra, Micah Adler, and Prashant Shenoy. 2001. Deadline fair scheduling: bridging the theory and practice of proportionate pair scheduling in multiprocessor systems. In *Real-Time Technology and Applications Symposium, 2001. Proceedings. Seventh IEEE*. IEEE, 3–14.

Thidapat Chantem, X. Sharon Hu, and Robert P. Dick. 2011. Temperature-Aware Scheduling and Assignment for Hard Real-Time Applications on MPSoCs. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* 19, 10 (Oct 2011), 1884–1897.

R. David and H. Alla. 2008. Discrete, Continuous and Hybrid Petri Nets (David, R. and Alla, H.; 2004). *Control Systems, IEEE* 28, 3 (June 2008), 81–84.

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

G. Desirena, L. Rubio, A. Ramirez, and J.L. Briz. 2019. Thermal-Aware HRT Scheduling simulation framework. (2019). https://www.gdl.cinvestav.mx/art/uploads/TCPN-Thermal-Aware_Real-Time_Scheduling.zip

Gaddiel Desirena-Lopez, José Luis Briz, Carlos Renato Vázquez, Antonio Ramírez-Treviño, and David Gómez-Gutiérrez. 2016. On-line Scheduling in Multiprocessor Systems based on continuous control using Timed Continuous Petri Nets. In *13th International Workshop on Discrete Event Systems*. 278–283.

Gaddiel Desirena-Lopez, Carlos Renato Vázquez, Antonio Ramírez-Treviño, and David Gómez-Gutiérrez. 2014. 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*.

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

Xing Fu, Xiaorui Wang, and Eric Puster. 2009. 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*. IEEE, 403–412.

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

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

Hassan K Khalil and JW Grizzle. 1996. *Nonlinear systems*. Vol. 3. Prentice hall New Jersey.

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

Chung Laung Liu and James W Layland. 1973. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM)* 20, 1 (1973), 46–61.

Yongpan Liu, Robert P Dick, Li Shang, and Huazhong Yang. 2007. Accurate temperature-dependent integrated circuit leakage power estimation is easy. In *Proceedings of the conference on Design, automation and test in Europe*. EDA Consortium, 1526–1531.

Srinivasan Murali, Almir Mutapcic, David Atienza, Rajesh Gupta, Stephen Boyd, Luca Benini, and Giovanni De Micheli. 2008. Temperature Control of High-Performance Multi-core Platforms Using Convex Optimization. In *Design, Automation and Test in Europe*. 110–115.

Geoffrey Nelissen, Vandy Berten, Jöel Goossens, and Dragomir Milojevic. 2011. Reducing Preemptions and Migrations in Real-Time Multiprocessor Scheduling Algorithms by Releasing the Fairness. 1 (Aug 2011), 15–24.

Bing Shi, Yufu Zhang, and Ankur Srivastava. 2010. Dynamic thermal management for single and multicore processors under soft thermal constraints. In *Proceedings of the 16th ACM/IEEE international symposium on Low power electronics and design*. ACM, 165–170.

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

Kevin Skadron, Mircea Stan, and Wei Huang. 2010. Thermal Modeling for Processors and Systems-on-Chip. In *Processor and System-on-Chip Simulation*. Springer, 243–257.

Vadim Utkin, Jürgen Guldner, and Jingxin Shi. 2009. *Sliding mode control in electro-mechanical systems*. Vol. 34. CRC press.

Carlos Renato Vázquez, Antonio Ramírez, and Manuel Silva. 2014. Controllability of timed continuous Petri nets with uncontrollable transitions. *Internat. J. Control* 87, 3 (2014), 537–552.

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

Dakai Zhu, Daniel Mossé, and Rami Melhem. 2003. Multiple-resource periodic scheduling problem: how much fairness is necessary?. In *Real-Time Systems Symposium, 2003. RTSS 2003. 24th IEEE*. IEEE, 142–151.