

Received May 12, 2021, accepted May 26, 2021, date of publication June 4, 2021, date of current version June 16, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3086698

# Maximizing Utilization and Minimizing Migration in Thermal-Aware Energy-Efficient Real-Time Multiprocessor Scheduling

LAURA ELENA RUBIO-ANGUIANO<sup>1</sup>, ABEL CHILS TRABANCO<sup>2</sup>, JOSÉ LUIS BRIZ VELASCO<sup>2</sup>,  
AND ANTONIO RAMÍREZ-TREVIÑO<sup>1</sup>

<sup>1</sup>CINVESTAV-IPN Unidad Guadalajara, Zapopan 45019, Mexico

<sup>2</sup>DIIS/I3A, Universidad de Zaragoza, 50018 Zaragoza, Spain

Corresponding author: Laura Elena Rubio-Anguiano (laura.rubio@cinvestav.mx)

This work was supported by the MINECO/AEI/ERDF (EU) under Grant PID2019 105660RB C21/AEI/10.13039/501100011033, in part by the Aragón Government (T58\_20R Research Group), and in part by the Construyendo Europa desde Aragón under Project ERDF 2014-2020. The work of Laura Elena Rubio-Anguiano was supported by CONACYT for providing a scholarship.

**ABSTRACT** This work proposes CAIECs, a clustered scheduling system for MPSoCs subject to thermal and energy constraints. It calculates off-line a cyclic executive honoring temporal and thermal constraints, for a hard real-time (HRT) task set at minimum frequency to reduce consumed energy, minimizing context switches and migrations. It also provides an on-line controller able to manage system and task parametric variations and soft real-time (SRT) tasks, always meeting the HRT task set constraints and the system thermal bound. CAIECS maximizes CPU utilization to help avoid overprovisioning contributing to a low SWaP factor. Its modular design allows the utilization of different modeling and scheduling approaches, and makes the off-line and on-line components independent from each other to better suit the requirements of a specific system. We experimentally show that the cyclic executive provided by CAIECS for HRT task sets outperforms RUN, a reference off-line algorithm in terms of optimal number of context switches.

**INDEX TERMS** Real-time, scheduling, clustering algorithms, multiprocessors, Petri nets, control.

## I. INTRODUCTION

Multiprocessor architectures are increasingly common in embedded real-time (RT) systems because they help reduce the SWaP-C factor (Space, Weight, Power and Cost). Current cars encompass over 125 electronic control units (ECUs), whose number is expected to grow as more and more features are incorporated [1], [2]. The aerospace industry is also familiar to this problem. Along with the cellphone market and edge computing designs subject to RT constraints in IoT (Internet of Things) ecosystems, all of them can benefit from powerful, compact multiprocessor systems on a chip (MPSoCs).

MPSoCs can be effective for avoiding overprovisioning. They facilitate the design process in many aspects, but they also entail tough problems which are easier to solve on single-core chips. This happens with the optimal allocation

The associate editor coordinating the review of this manuscript and approving it for publication was Wentao Fan<sup>1</sup>.

and scheduling of RT task sets, for which there are two known approaches: *partitioning* and *global scheduling*. Partitioning allocates tasks statically to CPUs with no migration. The benefit of partitioning approaches is that we can resort to the solid uniprocessor scheduling theory for hard RT (HRT) schedulability analysis. The downside is a limit on the maximum 50% utilization bound [3], which heuristic approaches can improve [4]. Alternatively, global scheduling allows jobs to migrate among CPU cores, maximizing CPU utilization and consequently reducing the number of required processor units. The downside lies in the complexity of its implementation. It usually involves a high overhead in number of context switches and migrations, often difficult to characterize, which hampers the theoretical benefits of maximizing CPU utilization.

Two other techniques try to balance the pros and cons of partitioned and global scheduling. *Clustered scheduling* divides tasks and cores into disjoint clusters so that jobs can exclusively migrate within their cluster, reducing the size of

the global scheduling problem. *Semi-partitioned scheduling* allows some tasks to migrate across processors.

Finally, a more conservative approach is designing a cyclic executive upon static partitioning. The automotive and aerospace industry mostly rely in this scheme [5]–[7], which leads to poor CPU utilization. Difficulties exacerbate when considering thermal or energy constraints. Ad-hoc optimal solutions are possible [8] but there is a lack of feasible, usable helping tools to find a suitable schedule for a HRT task set on a multicore processor maximizing CPU utilization and honoring energy or thermal constraints.

### A. CONTRIBUTIONS

We introduce in this paper a scheduling framework named CAIECS (*Clustered Allocation and Execution Control Scheduler*) which encompasses two core contributions beyond the framework itself. The first contribution is a method to calculate a cyclic executive for a HRT task set on a multicore chip optimizing CPU utilization while ensuring temporal correctness at the lowest possible frequency, minimizing the number of context switches and migrations, and honoring a thermal bound. Following a clustered approach, we first try to partition the HRT task set to maximize utilization with zero migrations. To this end, we resort to a common binary packing (BPP) formulation. The novelty lies in that if a perfect full partition is not found, we leverage a fluid global scheduler to maximize CPU utilization, avoiding the drawbacks of global scheduling in two ways. On the one hand, the result is a cyclic executive, and therefore there are no calculations at runtime. On the other hand, we keep the number of context switches and migrations very low: in our experiments our solution outperforms RUN, a global scheduler which is considered a reference in this aspect [9].

The second contribution is a runtime controller that ensures the accomplishment of the HRT task set constraints while managing the admission of SRT aperiodic tasks, and the rejection of small disturbances such as those produced by parameter variations, for example. This controller takes as an input a set of references derived by CAIECS from the foregoing cyclic executive, leverages DVFS to throttle the frequency in order to ensure temporal correctness, and avoid the violation of a given thermal threshold, irrespective the upper TDP (*thermal design point*) established by the chip-maker. CAIECS constitutes a good starting point to design either a standalone cyclic executive for a HRT task set or a MIF (*Minor Frame*) in a collective hyperperiod.

### B. EXTENDED SUMMARY

We model the whole system (RT tasks, processors and thermal parameters) under a single formalism, namely Timed Continuous Petri Nets (TCPNs). Leveraging the TCPN model, we calculate the minimum CPU frequency ( $F^*$ ) required to meet the HRT constraints and the maximum frequency ( $F^+$ ) that guarantees the thermal requirement of the system. The ensuing stage packs HRT tasks into 1-size bins (*minor clusters*) or into an integer number of bins of

size greater than one (*major clusters*), leveraging a Binary Packing Problem (BPP) formulation, employing a *Best-Fit Descending* (BFD) heuristic.

HRT tasks in minor clusters can be scheduled on a single processor. We resort to EDF because of its optimality [10] but other options are possible. To schedule the major clusters we formulate a global scheduling problem over each cluster as an Integer Programming Problem (IPP). We prove that this IPP can be solved as an Linear Programming Problem (LPP), and that if the LPP is solvable, then the solution is optimal. The size of this LPP on a major cluster is far smaller and consequently faster to solve than an LPP considering the whole HRT task set. The resulting schedule consists in a cyclic executive over the hyperperiod.

A run-time control system tracks the off-line scheduling solution for the HRT task set controlling the admission of SRT aperiodic tasks and managing small disturbances. We compare CAIECS with RUN in the absence of SRT aperiodic tasks or disturbances (for the comparison to be fair). CAIECS results in fewer context switches and migrations and requires minimum run-time support.

The whole process from the modeling stage to the simulation and analysis of results is fully automated and can be customized using Tertimuss, a simulation framework publicly available, which includes a number of schedulers available out-of-the-box [11]. The modular design of CAIECS allows for the adoption of different thermal models, BPP solvers or minor and major cluster schedulers.

Sec. II discuss the related work and the originality of the proposal. Sec. III states the scheduling problem and notation. Sec. III-A introduces a background and notation used for the TCPN model in this paper. Sec. IV provides an overview of CAIECS. In Sec. V the conditions for energy minimization and thermal bound compliance are explained. Sec. VI details the clustering process. Sec. VII to Sec. IX present the runtime controller. Sec. X shows an example of the scheme demonstrating the operation of the feedback controller. The experimental comparison between RUN and CAIECS is found in Sec. XI. Sec. XII summarizes the computational complexity of the different algorithms. Finally, the conclusions for this work are drawn in Sec. XIII.

## II. RELATED WORK

The problem of finding a HRT multiprocessor scheduler which accounts for energy and thermal constraints was recently studied in [12]. The authors try to maximize workload-per-joule (WPJ) subject to a thermal constraint. The allocation of tasks to processors relies on an (enhanced) variable-size bin packing algorithm (En-VSBP): the size of a bin depends on the capability (frequency) of each core. The number of cores running at a given frequency according to the thermal-compliant WPJ maximization problem determines the available number of bins of a given size. When there is no bin size capable to fit a given task because of the high utilization of the latter (named a *heavy task*), the task is allocated to some available idle core, if such core exists. The

purpose is to turn off other cores so that the frequency of the newly allocated core can be increased without violating the thermal constraint. Otherwise, the allocation fails.

The downside of the heuristic approach in [12] is that it cannot ensure schedulability, and only explores a very small portion of the solution space, yielding sub-optimal results. Also, authors in [12] apply an abstraction of an RC thermal and power model at the chip level, whereas we can fine-tune thermal properties and dynamics by discretizing the MPSoC or the relevant part of it with prisms as small as desired [13], [14]. We provide a holistic formal model of the system, whereas RC power and thermal models do not provide a true integrated state model.

The core of our research resides in the exploitation of *fluid global scheduling*. The latter relies on the idea of *proportionate fairness (Pfair)* [15], *PD* [16], *PD<sup>2</sup>* [17]. This type of schedulers constitute the only known solution for multiprocessor scheduling which has been proved HRT and SRT optimal for implicit deadline task sets, whereas no optimal scheduler exists for constrained or arbitrary deadlines [18]. Simpler global schedulers like G-EDF are easier to implement but can only provide SRT optimality [19].

Pfair algorithms discretize time so that tasks can only run for an integer number of quanta  $Q$ , distributing time quanta among tasks so that the difference between the execution time of every task and the fluid schedule (*fluid error*) is smaller than the unit at any time. This usually leads to a high number of context switches and migrations. *Deadline partitioning (DP)* approaches such as *DP-Fair* [20] can lower this overhead by limiting the scheduling points to the set of all task deadlines. In this way, scheduling actions are only taken at variable time intervals instead of at a fixed quantum. U-EDF achieves even better results at the price of a loss of optimality in CPU usage [21].

Authors in [22] also rely on DP to find an energy-efficient scheduler based on mathematical optimization, in this case 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. After assuming some system constraints, a linear programming problem (LPP) solves the task workload and a hetero-wrap algorithm solves the task ordering problem. The optimization problem is solved in runtime, entailing a non-negligible overhead even for a discrete set of frequencies.

A recent approach leveraging mathematical optimization and open to energy implications is [23]. The proposal targets heterogeneous IoT systems. The authors pose a Mixed Integer Linear Programming problem (MILP) similar to ours. The formulation is interesting in its generality and in that it encompasses security constraints besides energy, but it obviates temperature and does not focus on maximizing the utilization, actually resorting to a partitioning scheme.

A way to reduce the size of the LPP upon a DP basis appears in [24] (which corrects [25]). The proposal is thermal and energy-agnostic but provides a necessary and sufficient schedulability condition. They apply DP and solve the LPP

on a time window of size  $0 \leq l \leq 1$ . With arbitrarily small sizes, the algorithm approaches the fluid concept. Another recent work on energy-aware RT multiprocessor scheduling based on DP is HEART [26]. The proposal does not take temperature into account and leverages a series of heuristics which entails an exponential complexity and can only explore a limited subset of scheduling solutions. Other recent research exploiting DP is LAA [27], in the context of dynamic tasks systems, extended to SRT aperiodic tasks in [28]. They do not deal with energy or thermal issues.

RUN (*Reduction to Uniprocessor*) is a non-fluid global scheduler which achieves a notably low number of context switches and migrations with an optimal utilization [9], [29], [30]. We take it as an *optimal reference (target)* concerning the number of context switches and migrations. It considers feasible systems composed of independent implicit-deadline periodic (not sporadic) tasks on homogeneous processors, and transforms the multiprocessor RT scheduling problem into a set of uniprocessor scheduling problems. RUN considers the *utilization* of a task (proportion of the period required to execute each task instance or *job*, see Sec. III) and its *dual*. Thus, if 0.6 is the utilization of a task, the utilization of its dual is 0.4. During an off-line stage, RUN applies a *pack* operation to find task groups with an aggregated utilization less or equal to 1 (i.e. with maximum CPU utilization) which can be allocated to a virtual processor. Then, the *dual* operation follows after each successful *pack* operation.

The *pack-dual* operations continue until a single utilization system is found. The algorithm guarantees convergence. Then, RUN uses EDF to schedule each task group on-line, reversing the *pack-dual* operations performed off-line. The resulting schedule generates a small number of preemptions and migrations.

QPS (*Quasi-Partitioned Scheduling*) [31] partitions the system tasks into *minor* and *major* execution sets, depending on whether they require one or multiple processors. QPS boils down to a partitioned EDF in the case of minor sets, whereas major sets are scheduled either by a set of QPS servers on multiple processors, or by local EDF scheduler on a single processor depending on execution requirements. QPS adapts to the system load by monitoring major execution sets in run-time. Like RUN, QPS partitions the system off-line and generates the schedule on-line. If aperiodic tasks arrive to the system, QPS needs to recompute the servers, unlike fluid schedulers, which are more amenable to apply feedback control techniques, which is our case in this research. RUN and QPS behave similarly in the absence of aperiodic tasks [30], [32], and therefore we compare our cyclic executive with RUN.

A close example of research formulating a BPP to solve the RT multiprocessor scheduling problem is [33]. They solve the BPP for allocating tasks to processors according to clusters of a given size. Instead, we solve the BPP iteratively to optimize the size (utilization) of the clusters.

Feedback methods from control theory have been often used for RT scheduling with different purposes. A recent

example focusing on reliability in a RT system is [34]. The proposed technique involves instrumenting the code of the RT tasks, which influences on the WCET and utilization. Instead, we rely on code-oblivious feedback control. This approach has been mostly proposed for non-HRT systems. Thus, authors in [35] dynamically allocate firm RT dynamic tasks leveraging a PID (proportional-integral-derivative) controller. First, they select the least utilized processor to dispatch a task from a common task queue. Then, a per-CPU admission controller performs a task schedulability analysis and applies DVFS to the selected core. Once the computational demand drops, the processor is brought to a lower frequency state. In our case, the workload is known *a priori*. This makes possible to generate a pre-schedule which is tracked by a controller which modifies the frequency. We introduced a similar approach in [13]. Although it was capable of managing thermal and temporal constraints of HRT tasks, it did not accounted for energy minimization and incurred in a great number of context switches.

The CAIECS scheduling scheme stems from preliminary results in [36], where the AIECS fluid global scheduler is introduced. We now take an entirely different clustering approach nonetheless. Global scheduling only applies to major clusters, thus ensuring zero migrations on minor clusters. Additionally, we now provide full proofs and development of the LPP solution presented in [36], which is a conference paper with limited extension. The TCPN model of the system we use in this research is based on [13], [14], and [36] with changes on the flow of the execution transitions. The thermal module within this model was validated with Ansys® in [37].

### III. PROBLEM AND SYSTEM DEFINITION

*Definition 1:* Let  $\mathcal{T} = \{\tau_1, \dots, \tau_n\}$  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) which takes to complete any instance of the task (job),  $\omega_i$  the period and  $d_i$  is the relative implicit deadline ( $d_i = \omega_i$ ) [38].

Let  $\mathcal{P} = \{CPU_1, \dots, CPU_m\}$  be a set of  $m$  identical processors, with a discrete clock frequency  $F \in \{F_1, \dots, F_{max}\}$ .

Without loss of generality, 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 set of periodic tasks is scheduled up to the hyperperiod, which is defined as the period equal to the least common multiple of periods  $H = lcm(\omega_1, \omega_2, \dots, \omega_n)$  of the  $n$  periodic tasks. A task  $\tau_i$  executed on a processor  $CPU_j$  at its maximum frequency  $F_{max}$ , requires  $c_i = \frac{cc_i}{F_{max}}$  processor time at every  $\omega_i$  interval.

As in deadline partitioning approaches ([20]), 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  $\tau$  must execute during a given time interval.

TABLE 1. System notation.

Symbol	Description
$n$	Number of tasks
$m$	Number of processors
$\mathcal{T}$	The task set
$\mathcal{P}$	The Processor set
$\tau_i$	The $i^{th}$ task
$\tau_i^a$	The $i^{th}$ aperiodic task
$cc_i$	The WCET of $\tau_i$ in CPU cycles
$c_i$	The WCET of $\tau_i$
$d_i$	The relative deadline of $\tau_i$
$\omega_i$	The period of $\tau_i$
$u_i$	The utilization of task $\tau_i$
$U$	System utilization
$H$	Hyperperiod
$\mathcal{F}$	Set of discrete frequencies
$F^*$	The minimum frequency
$F_c$	The solution of Eq.(15)
$F^+$	The maximum thermal frequency
$F_n$	Frequency when aperiodic tasks are accepted
$SD$	The set of ordered deadlines $sd_k$
$sd_k$	The $k - th$ deadline in set $SD$
$I_{SD}^k$	The $k - th$ scheduling interval $(sd_{k-1}, sd_k)$
$x_i^k$	Cycles of $\tau_i$ to execute during $I_{SD}^k$
$k_q$	Thermal conductivity of component $q$
$V_q$	Volume of component $q$
$V_{CPU_j}$	Volume of CPU $j$
$\rho_q$	Density of component $q$
$cpq$	Specific heat capacity of component $q$
$h$	Convection coefficient
$T_q$	Temperature of component $q$

*Definition 2:* A task set is HRT schedulable under a given scheduling algorithm iff the temporal constraints are always satisfied, and SRT schedulable iff tardiness is bounded.

*Definition 3:* A task set is feasible iff there is at least a scheduling algorithm under which the task set is schedulable.

A task set with implicit deadlines is feasible if the following (sufficient) condition holds [15]:

$$U = \sum_{i=1}^n \frac{c_i}{\omega_i} \leq m \tag{1}$$

where  $U$  is the system utilization and  $m$  is the number of processors. Condition 1 is also necessary in the case of HRT task sets [39].

In Sec. IX we also consider the arrival of asynchronous aperiodic tasks that need to be executed on the system.

*Definition 4:* Let  $\mathcal{T}_a = \{\tau_1^a, \dots, \tau_p^a\}$  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.

### A. BACKGROUND ON PETRI NETS

This section aims to provide basic definitions and concepts on Timed Continuous Petri Nets (TCPN), the formal model used in this work to represent tasks, CPUs, energy consumption, temporal and thermal behavior. For a deeper insight on Petri Nets see [40]–[42].

*Definition 5:* A Petri Net structure (PN) is a 4-tuple  $N = (P, T, \mathbf{Pre}, \mathbf{Post})$  where  $P = \{p_1, \dots, p_{|P|}\}$  and  $T = \{t_1, \dots, t_{|T|}\}$  are finite disjoint sets of places and transitions.  $\mathbf{Pre}$  and  $\mathbf{Post}$  are  $|P| \times |T|$   $\mathbf{Pre}$ – and  $\mathbf{Post}$ – incidence

TABLE 2. Notation for the TCPN model.

TCPN Module for task $\tau_i$	
$p_i^w$	Period place of $\tau_i$
$p_i^d$	Deadline place of $\tau_i$
$p_i^{cc}$	Cpu cycles place of $\tau_i$
$t_i^w$	period transition of $\tau_i$
$\lambda_i^w$	Fire rate of transition $t_i^w$
TCPN Module for $CPU_j$	
$p_j^{idle}$	Idle state place of $CPU_j$
$p_j^{busy}$	Busy state place of $\tau_i$ in $CPU_j$
$t_{i,j}^{alloc}$	Allocation transition of $\tau_i$ in $CPU_j$
$t_{i,j}^{exec}$	Execution transition of $\tau_i$ in $CPU_j$
$\lambda_{i,j}^{alloc}$	Fire rate transition of $t_{i,j}^{alloc}$
$\lambda_{i,j}^{exec}$	Fire rate transition of $t_{i,j}^{exec}$
$\eta$	CPU modeling parameter
TCPN Thermal model	
$p_q^{com}$	Place of component $q$
$p_q^{air}$	Place of component $q$
$p_q^\alpha$	Place for leakage power of component $q$
$t_{q \rightarrow r}^{cond}$	Conduction transition from component $q$ to component $r$
$t_{q \rightarrow \infty}^{conv}$	Convection transition of component $q$
$t_{q \rightarrow \infty}^\alpha$	Convection transition of component $q$ to air
$t_q^\delta$	Leakage power transition for $\delta$ of component $q$
$t_q$	Leakage power transition for $\delta$ of component $q$
$\lambda_{q \rightarrow r}^{cond}$	Fire rate transition of $t_{q \rightarrow r}^{cond}$
$\lambda_{q \rightarrow \infty}^{conv}$	Fire rate transition of $t_{q \rightarrow \infty}^{conv}$
$\lambda_{q \rightarrow \infty}^\alpha$	Fire rate transition of $t_{q \rightarrow \infty}^\alpha$
$\lambda_q^\delta$	Fire rate transition of $t_q^\delta$
$\lambda_q$	Fire rate transition of $t_q$

matrices, where  $\mathbf{Pre}[i, j] > 0$  (resp.  $\mathbf{Post}[i, j] > 0$ ) if there is an arc going from  $p_i$  to  $t_j$  (or going from  $t_j$  to  $p_i$ ),  $\mathbf{Pre}[i, j] = 0$  (or  $\mathbf{Post}[i, j] = 0$ ) otherwise.

**Definition 6:** A continuous Petri net (ContPN) is a pair  $\text{ContPN} = (N, \mathbf{m}_0)$  where  $N = (P, T, \mathbf{Pre}, \mathbf{Post})$  is a PN and  $\mathbf{m}_0 \in \{\mathbb{R}^+ \cup 0\}^{|P|}$  is the initial marking.

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

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

**Definition 7:** A timed continuous PN (TCPN) is a time-driven continuous-state system described by the tuple  $(N, \lambda, \mathbf{m}_0)$  where  $(N, \mathbf{m}_0)$  is a continuous PN and the vector  $\lambda \in \{\mathbb{R}^+ \cup 0\}^{|T|}$  represents the transitions firing rates determining the temporal evolution of the system. Transitions fire according to a certain speed, which is generally a function of the transition firing rates and the current marking. Such function depends on the semantics associated to the transitions. Under infinite server semantics, the flow through a transition  $t$  (or  $t$  firing speed, denoted as  $\mathbf{f}(\mathbf{m})$ ) is the product of the firing rate,  $\lambda_i$ , and  $\text{enab}(t_i, \mathbf{m})$ , the instantaneous enabling of the transition, i.e.,  $\mathbf{f}_i(\mathbf{m}) = \lambda_i \text{enab}(t_i, \mathbf{m})$ .

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

to the concept of configuration. A configuration of a TCPN at  $\mathbf{m}$  is a set of  $(p, t)$  arcs describing the effective flow of each transition, and indicates that  $p_i$  constrains  $t_j$  for each arc  $(p_i, t_j)$  in the configuration. A configuration matrix is defined for each configuration as follows:

$$\mathbf{\Pi}_{j,i}(\mathbf{m}) = \begin{cases} \frac{1}{\mathbf{Pre}[i, j]} & \text{if } p_i \text{ is constraining } t_j \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

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

$$\dot{\mathbf{m}} = \mathbf{C} \mathbf{f}(\mathbf{m}) = \mathbf{C} \mathbf{\Lambda} \mathbf{\Pi}(\mathbf{m}) \mathbf{m} \quad (3)$$

A control action can be applied to (3) by adding a term  $\mathbf{u}$  to every transition  $t_i$  such that  $0 \leq u_i \leq f_i$ , indicating that its flow can be reduced. Thus, the controlled flow of transition  $t_i$  becomes  $w_i = f_i - u_i$  and the forced state equation is  $\dot{\mathbf{m}} = \mathbf{C}[\mathbf{f} - \mathbf{u}] = \mathbf{C} \mathbf{w}$ .

## B. TCPN SYSTEM MODEL

This subsection introduces and updates the TCPN model detailed in [13] encompassing processor cores (CPUs), task arrival, thermal activity and power consumption, and allowing the use of differential equations in control design. Fig. 1a depicts a section of the model corresponding to a system with  $n$  tasks ( $\tau_i$ ) and one processor ( $CPU_j$ ). The model has three main sub-nets: a task arrival module ①, a processor module per processor in the system ②, and the thermal model ③. The arrival module models the fluid arrival of a periodic task to the system, resorting to a transition whose firing rate represents the period  $\omega$  of the task.

Each processor module describes how tasks are allocated to a processor and executed. The boundary transitions  $t_{i,j}^{alloc}$  between a task and CPU modules are fired when the scheduler allocates task  $\tau_i$  to processor  $CPU_j$ . The task execution rate is expressed as the flow  $f_{i,j}^{exec}$  through transitions  $t_{i,j}^{exec}$ , when  $\tau_i$  is being executed at processor frequency  $F$ .

To model the thermal activity, the MPSoC board is divided into prismatic elements, as detailed in [13] and [37]. Pointer ③ from Fig. 1a shows the TCPN model of one of these elements, which models conduction, convection and heat generation in a specific chip area.

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

$$\dot{\mathbf{m}}_{\mathcal{T}} = \mathbf{C}_{\mathcal{T}} \mathbf{\Lambda}_{\mathcal{T}} \mathbf{\Pi}_{\mathcal{T}}(\mathbf{m}) \mathbf{m}_{\mathcal{T}} + \mathbf{C}_{\mathcal{T}}^{alloc} \mathbf{w}^{alloc} \quad (4a)$$

$$\dot{\mathbf{m}}_{\mathcal{P}} = \mathbf{C}_{\mathcal{P}} \mathbf{\Lambda}_{\mathcal{P}} \mathbf{\Pi}_{\mathcal{P}}(\mathbf{m}) \mathbf{m}_{\mathcal{P}} + \mathbf{C}_{\mathcal{P}}^{alloc} \mathbf{w}^{alloc} \quad (4b)$$

$$\dot{\mathbf{m}}_{exec} = \mathbf{f}^{exec} \quad (4c)$$

$$\dot{\mathbf{m}}_{\mathcal{T}} = \mathbf{C}_{\mathcal{T}} \mathbf{\Lambda}_{\mathcal{T}} \mathbf{\Pi}_{\mathcal{T}}(\mathbf{m}) \mathbf{m}_{\mathcal{T}} + \mathbf{C}_a \mathbf{\Lambda}_a \mathbf{\Pi}_a(\mathbf{m}) \mathbf{m}_a + \mathbf{C}_{\mathcal{P}}^{exec} \mathbf{f}^{exec} \quad (4d)$$

$$\dot{\mathbf{m}}_a = 0 \quad (4e)$$

$\mathbf{C}_x$ ,  $\mathbf{\Lambda}_x$ , and  $\mathbf{\Pi}_x(\mathbf{m})$  are the incidence matrix, the firing rate transitions and the configuration matrix ( $x = \{\mathcal{T}, \mathcal{P}\}$ )

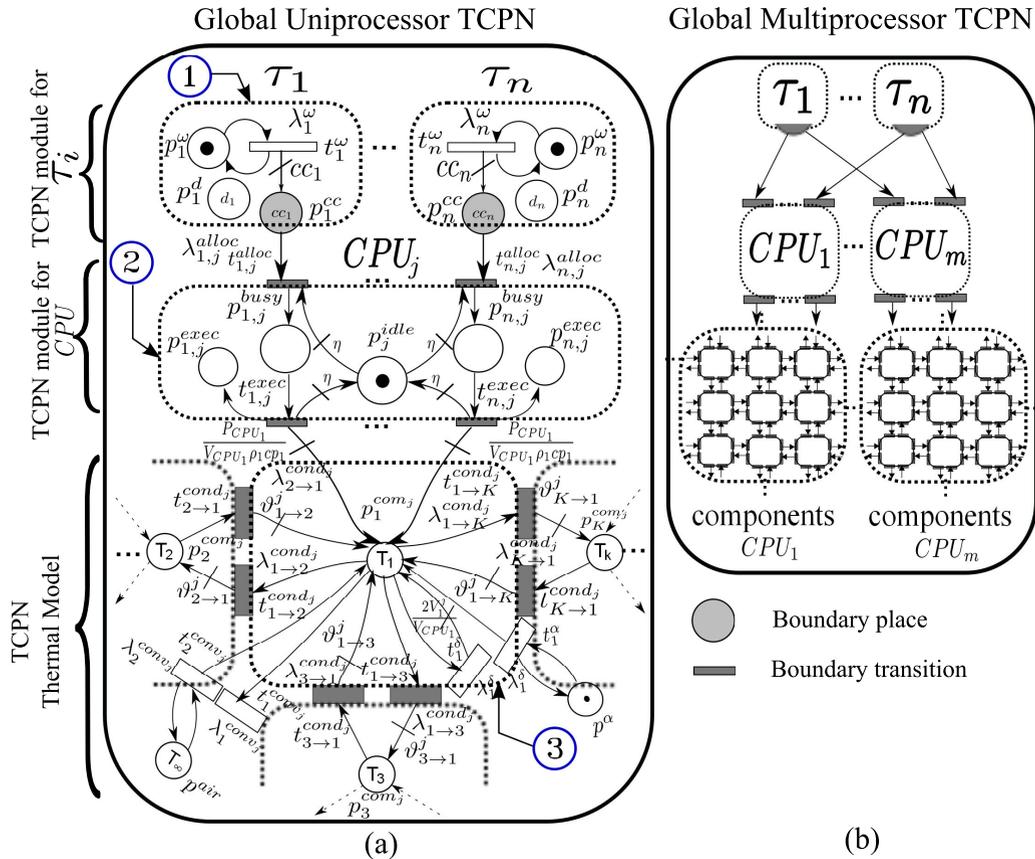


FIGURE 1. TCPN model integrating task ①, processor ② and thermal modeling ③. (a) details the case for a single processor, and (b) zooms out and extends the model for  $m$  CPUs.

of the thermal, task, and processor sub-nets respectively. Each equation from system (4) represents a module from the TCPN representation on Fig. 1. Eq. (4a) describes the periodic arrival of each task, Eq. (4b) the processors behavior, and Eq. (4c) the processors execution rate (i.e frequency). Finally, Eq. (4d) represents the thermal evolution of the system due to task execution, with Eq. (4e) indicating that the environmental temperature keeps constant during observation time (its derivative is neglected).

### C. EQUATIONS IN SCALAR FORM

The dynamics of the marking of each place of the TCPN model can be represented as the difference between the input and output flows. Specifically the dynamics of markings  $m_{i,j}^{busy}$ , from places  $p_{i,j}^{busy}$ , are given by (Sec. III-A):

$$\dot{m}_{i,j}^{busy} = \frac{\lambda_{i,j}^{alloc}}{\eta} m_{i,j}^{idle} - u_{i,j}^{alloc}, \quad (5)$$

where  $0 \leq u_{i,j}^{alloc} \leq \frac{\lambda_{i,j}^{alloc}}{\eta} m_{i,j}^{idle}$ . On the other hand, for the design and simulation of the execution control the dynamics of markings  $m_{i,j}^{exec}$ , from places  $p_{i,j}^{exec}$ , are conceived such that the flow is constant and equal to the processors frequency  $F$  when task  $\tau_i$  is active on processor  $CPU_j$ . Thus:

$$\dot{m}_{i,j}^{exec} = \lambda_{i,j}^{exec} m_{i,j}^{busy} - v_{i,j}^{exec} = \sigma_{i,j} F, \quad (6)$$

where  $v_{i,j}^{exec}$  stands for a corrective parameter that can take any real value and:

$$\sigma_{i,j} = \begin{cases} 1 & \text{if } m_{i,j}^{busy} > 0 \\ 0 & \text{if } m_{i,j}^{busy} = 0. \end{cases} \quad (7)$$

### D. THERMAL CONSTRAINT

The thermal dynamic behavior from the TCPN model on Eq. (4d) can be written separately as:

$$\begin{aligned} \dot{m}_T &= A m_T + B' m_a + P_{cpu} B f^{exec} \\ Y_T &= S \vec{m}_T \end{aligned} \quad (8)$$

where  $A = C_T \Lambda_T \Pi_T(m)$ ,  $B' = C_a \Lambda_a \Pi_a(m)$ ,  $Y_T$  is the temperature at the center of each CPU and  $P_{cpu} B = C_P^{exec}$ , such that  $P_{cpu}$  stands for the CPU power consumption.

The dominant component of power consumption in CMOS technology is the dynamic power  $P_{dyn}$  given by  $P_{dyn} = C_{eff} V_{dd}^2 F$ , where  $C_{eff}$  is the effective switching capacitance,  $V_{dd}$  the supply voltage and  $F$  is the frequency of the clock. Given that  $V_{dd} \propto F$  and  $k$  is a modeling constant,  $P_{dyn}$  can be stated as:

$$P_{dyn} = k F^3 \quad (9)$$

Eq. (8) with  $P_{cpu} = P_{dyn}$  is used to derive the thermal constraints, in order to keep the system temperature below a given value  $T_{max}$ . Since the schedule is periodic and it repeats

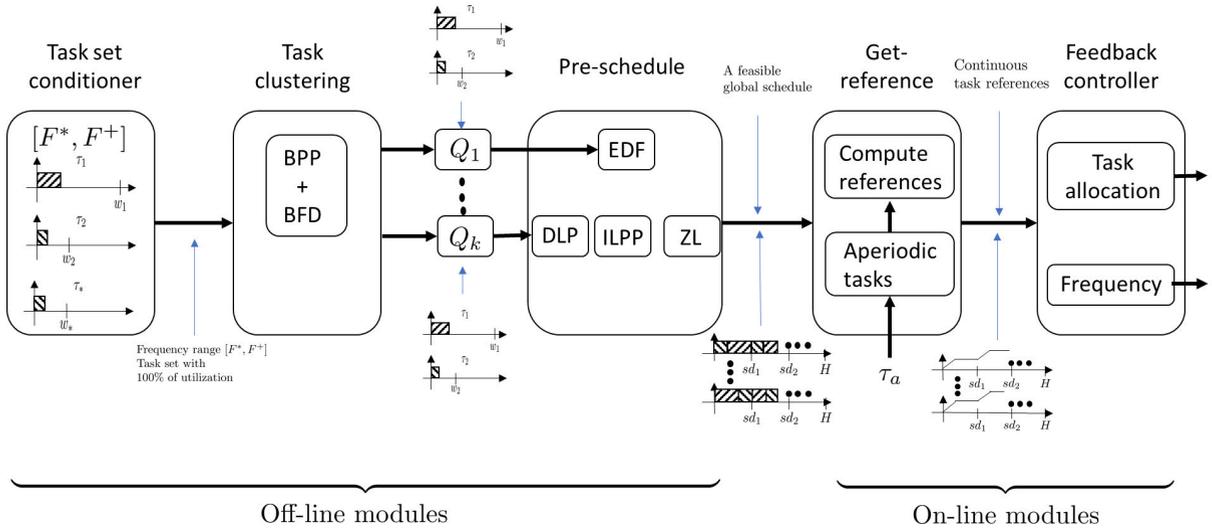


FIGURE 2. Scheduler scheme.

every hyperperiod, so must be the thermal solution. For this reason, we consider the temperature steady state ( $m_{T_{ss}}$ ), i.e. with no change in temperature  $\dot{m}_T = 0$ . Hence,

$$m_{T_{ss}} = -A^{-1}(kF^3Bw^{alloc} + B'm_a). \quad (10)$$

In order to comply with the thermal constraint of the system, the temperature in steady state must be less than or equal to a maximum threshold (i.e.  $Sm_{T_{ss}} \leq T_{max}$ ). Therefore:

$$-SA^{-1}kF^3Bw^{alloc} \leq T_{max} + SA^{-1}B'm_a \quad (11)$$

Eq. 11 provides the thermal constraint that the allocation of tasks to processors ( $w^{alloc}$ ) must fulfill, i.e.  $w^{alloc}$  represents a control variable. This equation includes the clock frequency and the temperature bounds along with the allocated tasks, which in the steady state are equivalent to the executed tasks. Eq. 11 is used to compute the range of feasible operating frequencies in Sec. V.

#### IV. OVERVIEW OF THE CAIECS SCHEDULING SYSTEM

This section will help the reader identify the elements of CAIECS as we extend on them in further sections. The CAIECS architecture appears in Fig. 2. The modules *task set conditioner*, *clustering*, and *pre-schedule* process the HRT task set, to yield an optimal schedule entirely off-line, ensuring minimum energy consumption and honoring a thermal bound. The modules *get-reference* and *feedback controller* act on-line, at runtime. They guarantee the accomplishment of the HRT and thermal constraints upon the arrival of SRT aperiodic tasks or in the presence of disturbances.

The *task set conditioner* determines the schedulability of the HRT task set on the available CPUs. It assumes that if a schedule exists, it must be periodic, producing a periodic thermal evolution of the system that reaches a stationary state when the CPU clock frequency is adequately chosen. Hence, it computes a frequency range taking advantage of the steady state of a formal system model consisting in a *Timed*

*Continuous Petri Net* (TCPN). The solution is in the range  $[F^*, F^+]$  of feasible clock frequencies. At these frequencies the tasks are executed in the MPSoC without violating the temporal and thermal constraints. This module also adds a dummy task  $\tau_*$  to ensure that the integer CPU utilization is equal to the number of processors  $m$  (i.e. the utilization is 100%) when the lowest possible frequency is selected.

Then, the *clustering* module partitions the task set into  $k$  clusters  $Q_1, \dots, Q_k$  in such a way that the task utilization  $s_j$  in cluster  $Q_j$  is an integer and  $\sum_j^k s_j = m$ . This module uses a bin packing problem (BPP) algorithm based on the Best Fit Descending (BFD) heuristic. Its computational complexity is  $(m^2 \times \log(n) \times n)$ , so it always finds a solution in polynomial time. At worst, this module returns a single set  $Q_1$  and  $s_1 = m$ , even if there exists another integer partition with more clusters.

The goal of the *pre-schedule* module is to find a feasible schedule. It allocates a cluster  $Q_j$  to  $s_j$  processors. When  $s_j = 1$  (*minor cluster*),  $Q_j$  is assigned to a single CPU and we resort to an EDF scheduling policy to find a feasible schedule. If  $s_j > 1$  (*major cluster*), then  $Q_j$  is assigned to  $s_j$  processors and the hyperperiod of the task cluster  $Q_j$  is partitioned according to the deadlines of tasks in  $Q_j$ . The task load per deadline interval is computed by an Integer Linear Programming Problem (ILP). The constraint set is described by a unimodular matrix, therefore the ILP can be solved as a linear programming problem (LPP), saving computational time. Then, a Zero-Laxity policy finds a feasible schedule from the computed task load. The set of all the schedules of all clusters  $Q_j$  yields a single feasible schedule for the HRT task set, that could be implemented straightforward, as a simple cyclic executive with minimum overhead, in the absence of additional SRT sporadic tasks or system disturbances.

At runtime, the on-line module *get-reference* uses this feasible off-line schedule to generate task-allocation references for the next module. The *Aperiodic manager* in *Get-reference*

is capable of accepting or rejecting SRT aperiodic tasks leveraging a smart ZL policy, so that the temporal and thermal constraints of the system are always met.

The *feedback controller* allocates and executes tasks. It throttles the frequency to ensure that tasks are executed according to their references. Moreover, it is able to reject small disturbances consisting in short CPU detentions or parameters deviations.

### V. TASK SET CONDITIONER

The task set conditioner module checks that the HRT task set  $\mathcal{T}$  is schedulable on the available cores (Sec. III). This module yields a set of operating frequencies  $\mathcal{F} = \{F^*, \dots, F^+\}$ .  $F^*$  minimizes the energy consumption, and  $F^+$  is the maximum permissible frequency at maximum utilization. This frequency range ensures the accomplishment of the thermal bound  $T_{max}$ .

We leverage DVFS to vary processor frequency by selecting one from a finite set of a preset values, i.e.  $\{F_1, \dots, F_{max}\}$ , and as a result minimize the system dynamic energy. The average consumed energy in a processor is modeled by

$$E_j = \int_{\xi_1}^{\xi_2} P_{CPU_j}(F) d\xi \quad (12)$$

where  $P_{CPU_j}(F)$  represents the power consumed by  $CPU_j$ . It depends on the dynamic power  $P_{dyn}$  due to computational activities of tasks (Eq. (9)), and on the static power  $P_{leak}$  due to leakage. It is computed as:  $P_{CPU_j}(F) = P_{dyn_j}(F) + P_{leak_j}$ , where  $P_{leak_j}$  can be modeled as a linear function of temperature ([43]):  $P_{leak} = \delta T + \rho$ , with  $T$  as the CPUs temperature and  $\delta$  and  $\alpha$  are modeling constants.

The CPU frequency that minimizes energy consumption while meeting temporal constraints is (Proposition 1 in [14]):

$$F^* = \max\{F_1, \frac{1}{m} \sum_{i=1}^n \frac{cc_i}{\omega_i}\}. \quad (13)$$

provided that  $\forall i u_i \leq 1$ . At frequency  $F^*$  the system utilization is  $U = \sum_{i=1}^n \frac{cc_i}{\omega_i F^*} = m$  and the processor frequency is,

$$F^* = \min\{F \in \mathcal{F} | F \geq F^*\} \quad (14)$$

given the nature of the discrete set of frequencies of the CPUs.

When computing  $F^*$  (Eq. 13) we assume a fully utilized system, but actual  $F^*$  in Eq. 14 can make the execution faster, causing the utilization to become below 100%, in those cases we can introduce an idle task  $\tau_{idle}$ , to ensure that system utilization is 100%, discussed later in sec.VI-A.

To guarantee that the thermal constraint is also fulfilled, we must comply with Eq. (11) at  $F = F^*$ . Otherwise, the problem does not have a solution.

The *maximum thermal frequency*  $F^+ \in \mathcal{F}$  is the greatest frequency at which all CPUs can operate at 100% of utilization so that temperature meets the thermal constraint. To compute  $F^+$ , we first solve the optimization problem in

Eq. (15) to find the frequency upper bound  $F_c$  that satisfies the constraints.

$$\begin{aligned} & \max F_c \\ & \text{s.t. } -\mathbf{SA}^{-1} kF_c^3 \mathbf{B} \left[ \frac{CC_1}{F_c H}, \dots, \frac{CC_m}{F_c H} \right]^T \\ & \leq \mathbf{T}_{max} + \mathbf{SA}^{-1} \mathbf{B}' \mathbf{m}_a \\ & \frac{CC_j}{F_c H} = 1 \quad \forall j = 1, \dots, m \\ & F^* \leq F_c \leq F_{max} \end{aligned} \quad (15)$$

The first constraint establishes the thermal requirements as in Eq. (11).  $CC_j$  represents the cycles that  $CPU_j$  must execute per hyperperiod. We assume that the workload is equally distributed among CPUs. All CPUs must work at their maximum capacity, which is implied by the second constraint. The last constraint bounds  $F_c$  to the actual clock frequency range of the CPUs. Finally, the solution for  $F^+$  must be in the set  $\mathcal{F}$  of discrete frequencies. Thus, the processor frequency  $F^+$  is calculated as:

$$F^+ = \max\{F \in \mathcal{F} | F \leq F_c\}. \quad (16)$$

With the minimum frequency  $F^*$  and the maximum thermal frequency  $F^+$  we define the set of operating frequencies that meet the thermal constraint as follows:

$$\mathcal{F}^s = \{F \in \mathcal{F} | F^* \leq F \leq F^+\}, \quad (17)$$

### VI. TASK CLUSTERING

The *clustering module* aims to reduce the number of job migrations, constraining migration within clusters of processors. Clustering also downsizes the global scheduling problem, now reduced to each cluster. A desirable effect of the algorithm in this module is that it usually reduces the number of context switches as well. The worst case appears when only one cluster, containing all the processors, can be obtained. In such case the global scheduling algorithm must deal with the whole set of processors and tasks. The clustering problem can be formally stated as follows.

*Problem 1: Task Clustering.* Given a task set  $\mathcal{T}$  with  $n$  tasks HRT-schedulable on  $m$  processors, find a partition  $\mathcal{Q} = \{Q_1, \dots, Q_k\}$  of  $\mathcal{T}$  into  $Q_1, \dots, Q_k$  subsets (clusters) such that they are pairwise disjoint and  $\bigcup_{j=1}^k Q_j = \mathcal{T}$ , the utilization  $s_j$  of tasks in cluster  $Q_j$  (herein named the size of  $Q_j$ ) is an integer, and  $\sum_j s_j = m$ .

A cluster  $Q_j$  is a *minor cluster* if  $s_j = 1$ , and a *major cluster* if  $s_j \geq 2$ . The tasks in a minor cluster can be allocated and scheduled on a single processor.

A task set  $\mathcal{T}$  which is HRT feasible before the partition, will also be so after the partition, because the algorithm used in the module ensures that the total utilization of each cluster is equal to the number of processors allocated to it (Sec. III). The algorithm can only handle systems with full utilization, for which we resort to the scheme further described in this section.

We pose the following constraint in order to find a combination of clusters  $\mathcal{Q}$  of capacities  $s$  that ensures the maximum

utilization for each cluster:

$$\left( \sum_{i=1}^n u_i * x_{i,s,j} \right) - x_{s,j} * s = 0 \quad (18)$$

where,  $x_{i,s,j}$  is a decision variable which is one if task  $\tau_i$  is assigned to cluster  $j$  of capacity  $s$ , and  $x_{s,j}$  is also a boolean variable which is set to 1 when the  $j$ -th cluster of size  $s$  is selected. Eq. (18) considers all possible clustering scenarios, hence there could be at most  $\lfloor \frac{m}{s} \rfloor$  clusters of size  $s$ , for  $s = 1, \dots, m$ . Constants  $n$  and  $m$  represent the total number of tasks and the total number of processors respectively.

The next constraint guarantees that each task belongs to a single cluster,

$$\sum_{s=1}^m \sum_{j=1}^{\lfloor \frac{m}{s} \rfloor} x_{i,s,j} = 1 \quad (19)$$

Finally, the summation of cluster sizes should be equal to the number of processors:

$$\sum_{s=1}^m \sum_{j=1}^{\lfloor \frac{m}{s} \rfloor} x_{s,j} * s = m \quad (20)$$

Eqs. (18,19, 20) define a space of possible solutions that can be used to obtain a clustering with the lowest number of migrations. However, finding that optimal point requires the computation of the number of migrations generated by each scheduler, rocketing the complexity of the problem. Nonetheless, we can formulate an integer linear problem upon the intuition that the number of migrations decreases with the size of a cluster:

$$\begin{aligned} \min & \sum_{s=1}^m \sum_{j=1}^{\lfloor \frac{m}{s} \rfloor} x_{s,j} * m^{s-1} \\ \text{s.t. } & \forall s \in \{1..m\} \forall j \in \{1..\lfloor \frac{m}{s} \rfloor\} \\ & \left( \sum_{i=1}^n u_i * x_{i,s,j} \right) - x_{s,j} * s = 0 \\ & \forall i \in \{1..n\} \sum_{s=1}^m \sum_{j=1}^{\lfloor \frac{m}{s} \rfloor} x_{i,s,j} = 1 \\ & \sum_{s=1}^m \sum_{j=1}^{\lfloor \frac{m}{s} \rfloor} x_{s,j} * s = m \\ & x_{i,s,j} \in \{0, 1\}, x_{s,j} \in \{0, 1\} \end{aligned} \quad (21)$$

The objective function from Eq. 21 maximizes the number of clusters of the smallest possible size (starting with size 1) subject to the same constraints we had. This intuition proved to be coherent with the experimental results presented in Sec. XI.

We can approximate a solution to Eq. 21, i.e. to Prob. 1, by posing a Bin Packing Problem (BPP) [44]. We first solve the BPP for bins of size 1 (minor clusters). If there are bins with a utilization lower than one, we solve the BPP again

considering bins of size 2 (i.e. clusters with  $k = 2$ ), then 3 and so on until no unallocated tasks are left.

We resort to a BFD (*Best-Fit Descending*) heuristic [44] to solve the BPP but other solutions are possible (Alg. 1). Actually, the problem can also be undertaken by leveraging heuristics that solve the Knapsack Problem or the Cutting Stock problem. We made some preliminary calculations and found that the BPP formulation with the BFD heuristics provided the most promising results for us. Our implementation has the added benefit of providing a small number of processors required to execute the HRT task set  $\mathcal{T}$ , which aids to avoid overprovisioning, and opens up the chance for using the extra cores for redundancy or for allocating non-RT tasks in a mixed-criticality system.

---

#### Algorithm 1 Clustering Algorithm

---

```

1: Input  $\mathcal{T}$ : Task set;  $m$ : Number of CPUs
2: Output A set of clusters;
3: Aux. functions
   · SolveBPP(task set, binVolume) – Solves BPP for bins
   with volume binVolume so that each bin has maximum
   utilization;
   · utilization(bin) – Returns the sum of the utilizations of
   the tasks in the bin
4:  $Q = \emptyset$ ,
5:  $binVolume = 1$ ,
6:  $cpusToAssign = m$ ,
7:  $tasksWithoutCluster = T$ 
8:  $\mathcal{T} = \{T \cup \tau_{idle} | u_{\tau_{idle}} = m - \sum u_i\}$ 
9: while  $binVolume \leq cpusToAssign$  do
10:   SolveBPP(tasksWithoutCluster, binVolume)
11:   for all  $bin \in bins$  do
12:     if  $utilization(bin) == binVolume$  then
13:       // Each cluster has (number of CPUs in cluster,
       tasks in the cluster)
14:        $Q = Q \cup (binVolume, bin)$ 
15:        $tasksWithoutCluster = tasksWithoutCluster \setminus bin$ 
16:        $cpusToAssign = cpusToAssign - binVolume$ 
17:     end if
18:   end for
19:    $binVolume = binVolume + 1$ 
20: end while
21: // Ensure that all tasks and CPUs are in a cluster
22: if  $cpusToAssign! = 0$  then
23:    $Q = Q \cup (cpusToAssign, tasksWithoutCluster)$ 
24: end if

```

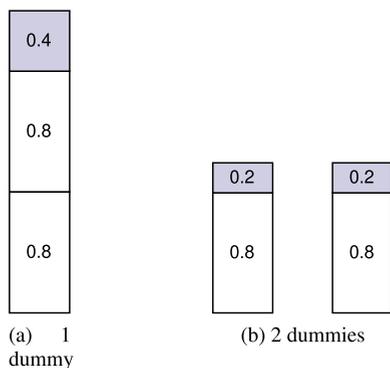
---

Finding an optimal solution to Eq. 21 requires that  $\sum u_i = m$ . Therefore, Alg. 1 starts by adding an idle task  $\tau_{idle}$  to  $\mathcal{T}$  such that its utilization is  $u_{\tau_{idle}} = m - \sum u_i$  (line 8). We make the deadline of this task be equal to the hyperperiod, so that it has always the lowest priority. Actually, it will never be scheduled. Then, the procedure consists in solving a BPP of size 1. While there are unassigned tasks left, the size of the bin is increased by one and the BPP is solved over the

remaining task subset. A group of tasks is assigned to a cluster if the utilization of the bin containing them is maximum. The variables *cpusToAssign* and *tasksWithoutCluster* represent, respectively, the number of CPUs and tasks unassigned to any cluster yet. Since this heuristic does not guarantee an optimal solution of the BPP, the condition in line 22 from Alg. 1 ensures that the last cluster contains any possible group of tasks with total utilization  $y < m$  that failed to conform a bin when  $binVolume = y$ .

**A. IDLE TASK: OTHER OPTIONS**

Adding a single dummy idle task  $\tau_{idle}$  as in Alg. 1 line 8 is just a simple solution, well suited to our scheduling scheme, which in later stages can deal with aperiodic tasks and disturbances. However, it does not always help minimize migrations. Let us consider two tasks such that  $u_1 = u_2 = 0.8$ , to be scheduled on two processors. Adding an idle task with  $u_{idle} = 0.4$  leads to the solution in Fig. 3 (a), with a single cluster holding all the three tasks  $\tau_1, \tau_2$  and  $\tau_{idle}$ . A global scheduler could yield migrations over the two processors. Alternatively, if we add two idle tasks with utilization equal to 0.2, we reach solution (b) in Fig. 3, with two clusters of size 1, that will be scheduled separately using EDF, with no migrations.



**FIGURE 3.** Differences in clustering between the usage of one dummy (forces one cluster) and two dummies (dummies are in gray).

**VII. PRE-SCHEDULE**

This section presents the scheduling of *major clusters*  $Q_j$  of size  $s_j > 1$ .

**A. WORKLOAD ASSIGNMENT**

To solve the workload assignment for the HRT task set  $Q_j$ , we formulate a linear programming problem (LPP), where each constraint captures a desired behavior for our schedule. The solution to the LPP is a set  $X$  of  $x_i^k$  that represents the workload that every task  $\tau_i$  in  $Q_j$  should execute at every scheduling interval  $I_{SD}^k, k = 1, \dots, \alpha$ , as in DP-fair approaches. Each scheduling interval is defined as the time between consecutive task deadlines ( $sd$ ),  $I_{SD}^k = [sd_{k-1}, sd_k)$ . Each task deadline  $sd_k$  is in the ordered set of all job deadlines  $SD = \{sd_0, \dots, sd_\alpha\}$ , where  $sd_0 = 0$  and  $sd_\alpha$  corresponds to the hyperperiod.

We define the *absolute laxity* of  $\tau_i$  in cycles as  $cc_i^* = (\omega_i F^*) - cc_i$ , i.e., the cycles that task  $\tau_i$  can remain idle before compromising its deadline. The number of processor cycles up to time  $sd_k$  is computed as  $(sd_k F^*) = q_i(\omega_i F^*) + r_i$ , where  $0 \leq r_i < (\omega_i F^*)$  and  $q_i \in \mathbb{Z}$ , such that  $q_i$  represents the instances of  $\tau_i$  up to  $sd_k$ , and  $r_i$  the amount of cycles that task  $\tau_i$  has been active since its last deadline ( $q_i \omega_i$ ) and time  $sd_k$ . If  $r_i = 0$ , then  $sd_k$  is a deadline for  $\tau_i$ .

The LPP is formulated in Eq. (22). Each constraint is defined per scheduling interval and per task, up to the hyperperiod. The *Maximum utilization constraint* (M.c) ensures that the system utilization per  $I_{SD}^k$  is 100%. The *Execution constraint* (E.c) forces the individual task  $\tau_i$  workload to complete  $cc_i$  before its deadline. If  $\tau_i$  has not reached its deadline, the *Laxity constraint* (L.c) guarantees that the sum of the workloads from its last deadline to the current  $I_{SD}^k$  should be greater than  $r_i - cc_i^*$ , such that  $\tau_i$  is not idle for more cycles than its absolute laxity. And the parallelism of a task is avoided due to the *Sequential constraint* (S.c).

*Proposition 1:* Given a task set  $\mathcal{T}$  as in Def. 1, where task utilization at  $F^*$  is equal to the number of processors, the solution of the LPP (22) is always a vector of integer numbers, and if each task  $\tau_i$  is executed for exactly  $x_i^k$  cycles during the  $k$ -th interval, then an optimal schedule is obtained.

$$\begin{aligned} & \max \sum_{i=1}^n \sum_{k=1}^{\alpha} x_i^k \\ & s.t \forall k \sum_{i=1}^n x_i^k = s_j \times |I_{SD}^k| \times F^* \text{ M.c} \\ & \text{if } r_i = 0 \sum_{j=\gamma}^k x_i^j = cc_i \text{ E.c} \\ & \text{if } r_i \neq 0 \sum_{j=\gamma}^k x_i^j \geq \max\{0, r_i - cc_i^*\} \text{ L.c} \\ & \text{where } \gamma \text{ is 1 or the last deadline interval} \\ & \forall i \ x_i^k \leq |I_{SD}^k| \times F^* \text{ S.c} \end{aligned} \tag{22}$$

*Proof:* The execution constraint in LPP (22) ensures that every task meets its deadline. Hence, if there is a solution, then the workload allocation leads to a feasible schedule. Furthermore, by Theorem 4 from [20], the schedule is also optimal.

To prove that the solution for the LPP (22) is always integer, we will show that the restriction matrix is unimodular. Let  $M \cdot y = b$  be the constraints from LPP (22), where  $y = [x \ h]$ ,  $x$  is the solution vector from the LPP,  $h$  represents the vector of slack variables and  $M$  is the restriction matrix. By construction,  $M$  has the form:

$$M = \left[ \begin{array}{c|c} A & \emptyset \\ B & \\ \hline I_{sc} & I_h \end{array} \right] \tag{23}$$

$A$  represents the equality constraints,  $B$  the execution constraints that resulted on inequalities,  $I_{sc}$  the sequential

$$\begin{bmatrix} A \\ B \end{bmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ \hline 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 0 \end{pmatrix}$$

FIGURE 4. Form of submatrix [A B]<sup>T</sup>.

constraints (one for each  $x_i^k$ ) and  $I_h$  corresponds to the slack variables. By construction all constraints are linearly independent among them, hence  $M$  is full row rank with  $rank(M) = \alpha(2n + 1) - 1$ , where  $\alpha$  is the number of scheduling intervals.

From the properties of totally unimodular matrices (TUM), it is well known that the property of total unimodularity (TU) holds under the adjoining of unit vectors [45]. Thus,  $M$  is TUM if submatrix  $[A B]^T$  is TUM.

Submatrix  $[A B]^T$  has the particular form showed in Fig. 4. It contains a special structure of  $-1$ s in stair. If we remove all unit vectors from  $[A B]^T$  but those in the stairs the TU property still holds. Let this new matrix be  $M'$ . The TU property is also preserved under elementary row operations with no scaling, then each row in the stair structures from  $M'$  can be transformed to unit vectors. Thus it is sufficient to prove that  $A$  is TUM.

We claim that  $A$  is TUM because it satisfies Theorem 3 from [46], with a row partition ( $T_1$  and  $T_2$ ) such that rows associated with the *Maximum utilization constraints* (M.c) are elements of  $T_1$  and the restrictions corresponding to the *Execution constraints* (E.c) are in  $T_2$ . □

**B. ZERO LAXITY POLICY**

The clock frequencies  $F^*$ ,  $F^+$  and the workload  $X$  computed previously determine that task  $\tau_i$  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 the ZL policy as posed in Alg. 2, following the results from Prop. 1.

*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  $SD = \{0, 5, 10, 15, 20, 25, 30\}$ . The solution from LPP (22) (per task  $\tau_i$  and per  $I_{SD}^k$ ) is

		k=1	k=2	k=3	k=4	k=5	k=6
$x_i^k$	i=1	3	3	3	3	3	3
	i=2	3	3	5	1	5	1
	i=3	5	1	3	3	1	5
	i=4	1	5	1	5	3	3
	i=5	3	3	3	3	3	3

**Algorithm 2** 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**

Applying the zero-laxity policy (Alg.2) up to the hyperperiod, we find the target schedule in Fig. 5a.

**VIII. GET-REFERENCE**

**A. COMPUTING REFERENCES**

The following task *execution paths* are computed from the cyclic executive obtained off-line. They will serve as *references* for the controller and will add robustness to the scheduler. Fig. 5b shows the *execution paths* of  $\tau_1$  on each CPU for *Example 1*.

Given the nature of the scheduling problem, these *execution paths* are *piecewise smooth functions*, as shown in Fig. 5b. Let define the *execution path*  $\dot{R}_{i,j}$  per task  $\tau_i$  on  $CPU_j$  as:

$$\dot{R}_{i,j}(\zeta) = F^*[W_{i,j}(\zeta)] \tag{24}$$

where  $\zeta$  stands for time,  $F^*$  is the operating frequency, and

$$W_{i,j}(\zeta) = \begin{cases} 1 & \text{if } \tau_i \text{ is executed on } CPU_j \\ 0 & \text{otherwise} \end{cases} \tag{25}$$

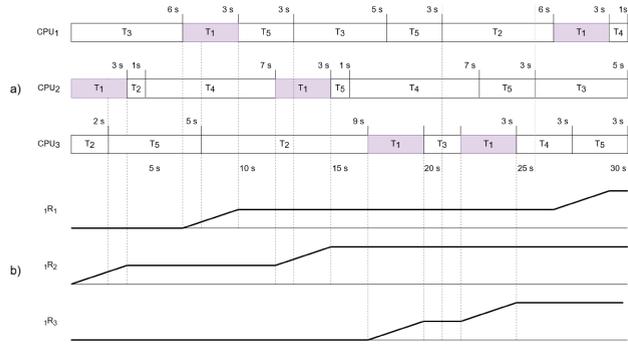
Furthermore, from the cyclic executive we can define a set  $D = \{d_0, d_1, \dots, d_k, \dots, d_h\}$  with all the time stamps  $d_k$  when a context switch occurred, from  $d_0 = 0$  up to the hyperperiod ( $d_h$ ), and

$$\delta_k = [d_{k-1}, d_k) \tag{26}$$

as the time interval between consecutive context switches  $d_k$  and  $d_{k+1}$ , where  $k = 1, \dots, h$ . These intervals are of special interest because they dictate when a task  $\tau_i$  must complete a certain *execution cycles*. We solve Eq. (24) at the end of interval  $\delta_k$  to compute such execution cycles:

$$R_{i,j}(d_k) = R_{i,j}(d_{k-1}) + F^* [W_{i,j}(d_{k-1})](d_k - d_{k-1}) \tag{27}$$

Consecutively,  $R_{i,k}(d_k)$  represents the number of CPU cycles that  $\tau_i$  must execute on  $CPU_j$  before being preempted.



**FIGURE 5.** a) Schedule of system from Example 1. b) Execution paths for  $\tau_1$  on each processor. The vertical lines link context switches from a) to b).

### IX. FEEDBACK CONTROLLER

The formal problem addressed solved by the feedback controller is stated as follows.

*Problem 2: Control RT Scheduler (CRTS).* Given the system defined in Def.1, the CRTS problem consists in designing a control law that tracks a feasible schedule computed beforehand. Additionally, the controller should execute or reject aperiodic tasks from Def. 4 upon arrival, subject to the temporal and thermal constraints of the HRT task set.

The allocation and execution of the HRT tasks at runtime is determined by the flow of transitions  $t_{i,j}^{alloc}$  and  $t_{i,j}^{exec}$  respectively. The markings  $m_{i,j}^{busy}$  and  $m_{i,j}^{exec}$  represent the tasks allocated and executed, and both constitute the variables under control. The allocation and execution control will work per  $\delta_k$  interval.

#### A. ALLOCATION CONTROL

Let us define the *vector allocation error*  $\mathcal{E}^{alloc}(\zeta)$  as,

$$\mathcal{E}^{alloc}(\zeta) := [\mathcal{E}_{1,1}^{alloc}, \dots, \mathcal{E}_{n,1}^{alloc}, \dots, \mathcal{E}_{n,m}^{alloc}]^T \quad (28)$$

where each  $\mathcal{E}_{i,j}^{alloc}(\zeta)$  is computed as,

$$\mathcal{E}_{i,j}^{alloc}(\zeta) = m_{i,j}^{exec}(\zeta) + m_{i,j}^{busy}(\zeta) - R_{i,j}(d_k) \quad (29)$$

where  $d_{k-1} \leq \zeta < d_k$ . Taking the time derivative of Eq. 29 and using Eqs. (5)- (6), the dynamics of each error is given by:

$$\dot{\mathcal{E}}_{i,j}^{alloc}(\zeta) = \dot{m}_{i,j}^{exec} + \dot{m}_{i,j}^{busy} = \frac{\lambda_{i,j}^{alloc}}{\eta} m_{i,j}^{idle} - u_{i,j}^{alloc}. \quad (30)$$

*Proposition 2:* Let  $u_{i,j}^{alloc}$  be an ON/OFF control for system (30), such that

$$u_{i,j}^{alloc} = \begin{cases} \frac{\lambda_{i,j}^{alloc}}{\eta} m_{i,j}^{idle} & \text{if } \mathcal{E}_{i,j}^{alloc}(\zeta) \geq 0 \\ 0 & \text{if } \mathcal{E}_{i,j}^{alloc}(\zeta) < 0 \end{cases} \quad (31)$$

Then system (30) is stable and each  $\mathcal{E}_{i,j}^{alloc}$  remains bounded for all  $d_{k-1} \leq \zeta < d_k$ .

*Proof:* To prove the stability, we assume that marking  $m_{i,j}^{idle}$ , from place  $p_j^{idle}$ , constrains transitions  $t_{i,j}^{alloc}$ . Hence,

a Lyapunov [47] candidate function  $V$  can be defined, satisfying  $V(\mathcal{E}^{alloc}) > 0, \forall \mathcal{E}^{alloc} \neq 0$  and  $V(\mathcal{E}^{alloc}) = 0$  for  $\mathcal{E}^{alloc} = 0$ , as

$$V = \frac{1}{2} \mathcal{E}^{alloc T} \mathcal{E}^{alloc}$$

Taking its time derivative yields:

$$\dot{V} = \mathcal{E}^{alloc T} \dot{\mathcal{E}}^{alloc} = \sum_{i=0}^{|\mathcal{T}|} \sum_{j=0}^{|\mathcal{P}|} \mathcal{E}_{i,j}^{alloc} \dot{\mathcal{E}}_{i,j}^{alloc}$$

For each  $\mathcal{E}_{i,j}^{alloc}$  there are two possible scenarios:

1.  $\mathcal{E}_{i,j}^{alloc} \geq 0$ . Then,  $u_{i,j}^{alloc} = \frac{\lambda_{i,j}^{alloc}}{\eta} m_{i,j}^{idle}$ , therefore

$$\mathcal{E}_{i,j}^{alloc} \dot{\mathcal{E}}_{i,j}^{alloc} = 0.$$

2.  $\mathcal{E}_{i,j}^{alloc} < 0$ . Since  $\mathcal{E}_{i,j}^{alloc} = -|\mathcal{E}_{i,j}^{alloc}|$  and  $u_{i,j}^{alloc} = 0$  we can state that

$$\mathcal{E}_{i,j}^{alloc} \dot{\mathcal{E}}_{i,j}^{alloc} = -\frac{\lambda_{i,j}^{alloc}}{\eta} |\mathcal{E}_{i,j}^{alloc}| m_{i,j}^{idle} \leq 0.$$

because  $m_j^{idle} \geq 0$  for all  $\zeta$ . Consequently we can conclude that  $\dot{V} \leq 0$  which implies that  $V(\zeta) \leq V(d_k), \forall d_{k-1} \leq \zeta < d_k$ . Therefore  $\mathcal{E}^{alloc}$  remains bounded for all  $d_{k-1} \leq \zeta < d_k$ .  $\square$

#### B. EXECUTION CONTROL

The allocation control assigns tasks to processors, whereas the execution control modifies the rate of execution. The allocation control is a scheduler, and the execution control modifies the frequency at which the processor should operate to comply with the task execution. The frequency is an input to the execution module of the TCPN model in Sec. III-B. In accordance with the off-line calculations, we assume that the CPUs are identical, they work at the same frequency and the operating frequency  $F$  is in the set  $\mathcal{F} = \{F^*, \dots, F^+\}$ .

In our model (Sec. III-B), each processor can only attend one task at a time and parallelism is not allowed. Consequently, the number of active tasks is  $|\mathcal{P}|$ , and therefore there must be  $|\mathcal{P}|$  transitions  $t_{i,j}^{exec}$  enabled. Accordingly, we define

$$\mathbf{m}_{active}^{exec} = A(\delta_k) \mathbf{m}^{exec} \quad (32)$$

where  $\mathbf{m}^{exec}$  stands for the vector that holds every element  $m_{i,j}^{exec}$ , and  $A(\delta_k)$  is a matrix with  $\{0, 1\}$  entries that depend on the tasks which are active during the interval  $\delta_k$ , defined in Eq. (26). Thus,  $\mathbf{m}_{active}^{exec}$  holds only  $m_{i,j}^{exec}$  values corresponding to the active tasks in the  $\delta_k$ . Therefore,  $A(\delta_k)$  has only one nonzero element per row, and no more than one nonzero element per column.

Then, the execution error of the active tasks  $\mathcal{E}_{exec}$  will be defined as:

$$\mathcal{E}_{exec}(\zeta) := A(\delta_k) \mathbf{m}^{exec} - \mathbf{R}(d_k) \quad (33)$$

for all  $\zeta \in \delta_k$ , and  $\mathbf{R}(d_k)$  is the vector of elements  $R_{i,j}(d_k)$  computed from Eq. (27) corresponding to every active tasks in  $\delta_k$ .

Using Eq. (33), the dynamics of the execution error is given by

$$\dot{\mathcal{E}}_{exec}(\zeta) = A(\delta_k) \mathbf{m}^{exec}(\zeta) \quad (34)$$

where  $A(\delta_k)$  remains constant during each interval  $\delta_k$ .

*Proposition 3:* Let  $F$  be the frequency at which all CPUs work during the interval  $\delta_k = (d_{k-1}, d_k]$ , such that

$$\frac{\|\mathcal{E}_{exec}(d_{k-1})\|}{\lambda|\delta_k|} \leq F \quad (35)$$

where  $|\delta_k|$  is the duration of interval  $\delta_k$ . Then  $\mathcal{E}_{exec}(\zeta)$  reaches zero before the end of the interval.

*Proof:* Let  $V$  be the candidate Lyapunov function:

$$V = \frac{1}{2} \mathcal{E}_{exec}^T \mathcal{E}_{exec}. \quad (36)$$

Deriving Eq. (36) and using Eq. (6),

$$\begin{aligned} \dot{V} &= \mathcal{E}_{exec}^T A(\delta_k) \mathbf{m}^{exec}(\zeta) \\ &= \mathcal{E}_{exec}^T A(\delta_k) \sigma F = \mathcal{E}_{exec}^T \Phi F \end{aligned} \quad (37)$$

where  $\sigma$  is a vector containing all  $\sigma_{i,j}$  and  $\Phi$  is a vector with all  $|\mathcal{P}|$  entries equal to 1. Then Eq. 37 can be rewritten as

$$\dot{V} = \sum_{j=1}^{|\mathcal{P}|} \mathcal{E}_{exec_j} F \quad (38)$$

From proposition 2,  $u_{alloc}$  restricts transitions  $t_{alloc}$  such that only  $R_{i,j}(d_{k+1})$  tokens are available, where  $\delta_k = [d_{k-1}, d_k)$ . Hence,  $\mathcal{E}_{exec_j} \leq 0$ , for  $j = 1, \dots, |\mathcal{P}|$ , and therefore:

$$\dot{V} = -\|\mathcal{E}_{exec}\|_1 F. \quad (39)$$

From Eq. (36),

$$V = \frac{1}{2} \mathcal{E}_{exec}^T \mathcal{E}_{exec} = \frac{1}{2} \|\mathcal{E}_{exec}\|_2^2$$

Solving for  $\|\mathcal{E}_{exec}\|$ ,

$$\|\mathcal{E}_{exec}\|_2 = (2V)^{1/2}, \quad (40)$$

Using Eq. (39),

$$\begin{aligned} \dot{V} &= -\|\mathcal{E}_{exec}\|_1 F \leq -\lambda \|\mathcal{E}_{exec}\|_2 F \\ \dot{V} &\leq -\lambda (2V)^{1/2} F \end{aligned} \quad (41)$$

By comparison with lemma [47], where  $\zeta \in \delta_k$ ,

$$\begin{aligned} \frac{dV}{d\zeta} &\leq -\lambda (2V)^{1/2} F \\ \int_{V(d_{k-1})}^{V(\zeta)} x^{-1/2} dx &\leq -\lambda \sqrt{2} F \int_{d_{k-1}}^{\zeta} d\tau \\ 2V(\zeta)^{1/2} - 2V(d_{k-1})^{1/2} &\leq -\lambda \sqrt{2} F (\zeta - d_{k-1}) \end{aligned} \quad (42)$$

Substituting  $V(\zeta)$  from Eq. (40) and solving for  $\|\mathcal{E}_{exec}(\zeta)\|$  we obtain:

$$\frac{2}{\sqrt{2}} \|\mathcal{E}_{exec}(\zeta)\| - \frac{2}{\sqrt{2}} \|\mathcal{E}_{exec}(d_{k-1})\| \leq -\lambda \sqrt{2} F (\zeta - d_{k-1})$$

$$\|\mathcal{E}_{exec}(\zeta)\| \leq \|\mathcal{E}_{exec}(d_{k-1})\| - \lambda F (\zeta - d_{k-1})$$

Solving for  $\zeta$  when  $\|\mathcal{E}_{exec}(\zeta)\| = 0$ , we get

$$\zeta \leq \frac{\|\mathcal{E}_{exec}(d_{k-1})\|}{\lambda F} + d_{k-1}$$

To ensure that task deadlines are met, the execution error should be zero before the interval  $\delta_k$  is over, thus

$$\zeta \leq \frac{\|\mathcal{E}_{exec}(d_{k-1})\|}{\lambda F} + d_{k-1} \leq d_k$$

Hence,

$$\begin{aligned} \frac{\|\mathcal{E}_{exec}(d_{k-1})\|}{\lambda F} &\leq d_k - d_{k-1} \\ \frac{\|\mathcal{E}_{exec}(d_{k-1})\|}{\lambda|\delta_k|} &\leq F \end{aligned}$$

Any frequency that satisfies Eq.(35) is suitable to take  $\mathcal{E}_{exec}$  to zero before the interval  $\delta_k$  is over. Therefore,  $F$  is chosen as the smallest  $F \in \mathcal{F}^s$  (from Ec. (17)) such that Eq.(35) is satisfied.

It is straightforward to prove by induction the stability of  $\mathcal{E}_{exec}$  up to the hyperperiod, as this Proposition holds for the interval  $\delta_1$  and therefore no error is carried over into the next execution interval. Therefore,

$$\|\mathcal{E}_{exec}(\zeta)\|_\infty \leq \max\{|\delta_1|, \dots, |\delta_h|\} F^*.$$

□

Proposition 3 shows that when selecting the operating frequency as in Eq.(35) the HRT tasks will not miss their deadlines even when a bounded overload is present.

### C. APERIODIC MANAGER

As stated before, the on-line control is capable to reject disturbances, due to this feature, and aperiodic tasks will be treated as such by the execution controller. Namely, when an aperiodic task  $\tau_{ap}$  arrives to the system and if it is accepted, we will simulate a disturbance on the processor  $CPU_j$  assigned to the  $\tau_{ap}$ , this way the controller will automatically update the CPU frequency. When the HRT task that was originally assigned to  $CPU_j$  reaches its zero laxity,  $\tau_{ap}$  will be assigned to the next available  $CPU$ . For this purpose, we first calculate the frequency  $F_n$  required to serve both the HRT tasks and the aperiodic task  $\tau_{ap}$ ,

$$F_n = F + \frac{cc_a}{d_a U} \quad (43)$$

where  $F$  is obtained from Eq. (35),  $U$  is the system utilization and  $(cc_a, d_a)$  are the parameters of the aperiodic task  $\tau_a$ . If  $F_n \leq F^+$ , the aperiodic task is accepted, otherwise it is rejected.  $F_n$  is now the minimum frequency acceptable in order to reach every task deadline. Therefore the set  $\mathcal{F}$  is restricted to those  $F \geq F_n$ . The aperiodic task manager is described by Alg.3.

**Algorithm 3** Aperiodic Manager

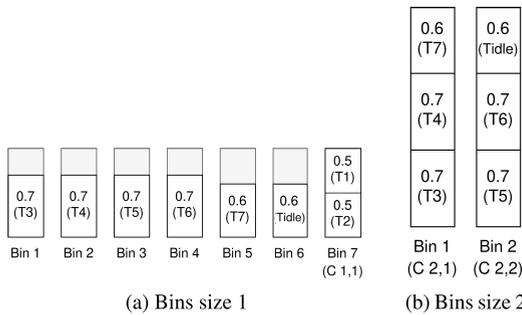
```

1: Input  $U$ : System utilization;  $\mathcal{P}$ : CPUs
2: Output  $F_n$ : frequency for aperiodic tasks;  $D$ : disturbance

3: Aux. functions
  · LaxyInCPU( $CPU_j, t$ ) – Returns laxity for  $\tau_{ap}$  at time  $t$ ;
4:  $F_n = F + \frac{cc_a}{d_i U}$ ,
5: if  $F_n \leq F^+$  then
6:   accept=1; // Accept  $\tau_{ap}$ 
7: end if
8: while accept == 1 do
9:   if LaxyInCPU( $CPU_j, t$ ) > 0 then
10:    Assign  $\tau_{ap}$  to  $CPU_j$ 
11:    Simulate Perturbation to  $CPU_j$ 
12:   else
13:     $CPU_j =$  next available CPU;
14:   end if
15:   if  $\tau_{ap}$  finishes execution then
16:    accept=0;  $F_n = F^*$ ;
17:    Eliminate disturbance to  $CPU_j$ 
18:   end if
19: end while
    
```

**TABLE 3.** Task set in the example. WCET ( $cc_i$ ) and relative deadlines ( $d_i$ ) given in cycles.

Tasks	$\tau_1$	$\tau_2$	$\tau_3$	$\tau_4$	$\tau_5$	$\tau_6$	$\tau_7$
$cc_i$	10	5	7	7	7	14	3
$d_i$	20	10	10	10	10	20	5
$u_i$	0.5	0.5	0.7	0.7	0.7	0.7	0.6
	$U_{tot}$ 4.4						



**FIGURE 6.** BBP in Alg. 1: a) Iteration 1; b) Iteration 2.

**X. PUTTING IT ALL TOGETHER: AN EXAMPLE**

**A. PROCESSORS AND HRT TASK SET**

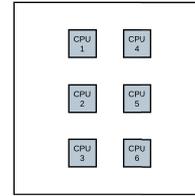
Consider the HRT task set in Tab. 3 with implicit deadlines, to be scheduled on an MPSoC with six available cores. Assume the cores can run at 1 Hz, 1.5 Hz, 2 Hz, 2.5 Hz and 3 Hz, just to be consistent with simple manageable WCETs and periods (relative deadlines) of the task set, as a way of example.

**B. TASK SET CONDITIONER**

The task set conditioner stage requires knowledge of the processors layout, materials, thermal properties and the CPU

**TABLE 4.** Material properties.

	Silicon	Copper
$cp$	712 J/Kg K	385 J/Kg K
$\rho$	2330 Kg/m <sup>3</sup>	8933 Kg/m <sup>3</sup>
$k$	148 W/m °C	400 W/m °C



**FIGURE 7.** Layout of the MPSoC board.

**TABLE 5.** Power frequency pairs.

Frequency	Power consumption
1.0 Hz	10 Watts
1.5 Hz	12 Watts
2.0 Hz	15.895 Watts
2.5 Hz	22.316 Watts
3.0 Hz	31.895 Watts

operating frequencies. In this example, the MPSoC is composed of six 1cm × 1cm silicon microprocessor mounted over a 7cm × 7cm copper board, as shown in Fig. 7. The thermal properties of the materials appear on Table 4 [37], where  $cp$ ,  $\rho$  and  $k$  stands for the specific heat capacity, density and thermal conductivity coefficient, respectively.

The task utilization in Table 3 ( $u_i$ ) assumes that the operating frequency is set to  $F^* = 1$  Hz, as calculated by the *Task set conditioner* following Eq. (14). Since the utilization of every task is less than 1 and the total utilization ( $U_{tot} = 4.4$ ) is less than the number of processors ( $m = 6$ ), we only need five out of the six processors to correctly run the HRT task set. Therefore,  $CPU_6$  will be off for the remaining of this example. Then, the optimization problem from Eq.(15) yields  $F^+ = 2.5$  Hz as the maximum frequency that the system stands with five cores running at maximum load and the sixth core off, while keeping the MPSoC temperature under  $T_{max} = 110^\circ C$ . The equation that describes the power dissipation for each core is assumed to be:  $P_{dyn} = C_1 * F^3 + C_2$ , where  $C_1 = 0.8421$  and  $C_2 = 9.1579$ , Table 5 shows the corresponding power frequency pairs.

**C. TASK CLUSTERING**

We now apply Alg. 1 to allocate the tasks to the five processors maximizing utilization. Line 8 adds the idle task  $\tau_{idle}$  such that  $d_{idle} = 20$  (Sec. VI). Since  $U - tot = 4.4$  (Table 3) and  $m = 5$ , the utilization of  $\tau_{idle} = 0.6$  and  $cc_{idle} = 12$ .

The first iteration of Alg. 1 solves the BBP with bins (clusters) of size (volume or capacity) binVolume = 1 using a BFD heuristic. The *size* of each task constitutes its utilization. This yields seven bins (Fig. 6 a), only one of which (Bin 7, (C 1,1)) is fully utilized. The heuristics ensures that no bin will be filled above its capacity. (C 1,1) can be allocated to a single CPU ( $CPU_1$ ), and accordingly removed from the pool of available cores, conforming a *minor cluster*.

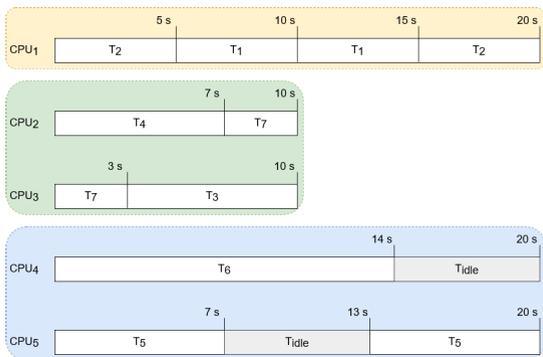


FIGURE 8. Scheduling the resulting clusters over their respective hyperperiod.

Since the iteration number (1) is less than the number of cores still available (4), the algorithm performs a second iteration using bins of volume  $\text{binVolume} = 2$ . This yields two major clusters, (Bin 1, (C 2,1)) and (Bin 2, (C 2,2)) (Fig. 6 b) each of which requires two processors. Consequently, we can allocate CPU<sub>2</sub> and CPU<sub>3</sub> to (C 2,1) (tasks  $\tau_3, \tau_4, \tau_7$ ), and CPU<sub>4</sub> and CPU<sub>5</sub> to (C 2,2) (tasks  $\tau_5, \tau_6, \tau_{idle}$ ). Each cluster can be scheduled using any optimal global scheduler. The current number of available cores (0) is less than the iteration number (3), so the algorithm ends.

D. PER-CLUSTER SCHEDULING

At the next step we apply the proper scheduler to each cluster. Fig. 8 (CPU<sub>1</sub>) shows the result of applying EDF to (C 1,1) (tasks  $\tau_1, \tau_2$ , hyperperiod equal to 20s), resulting in zero migrations, since it is a single-core cluster. For the dual-core clusters (C 2,1) and (C 2,2) we apply the AIECS global scheduler (hyperperiod equal to 10s in both cases). Upon the resulting per-cluster scheduling, we can now generate a cyclic executive for the HRT task set  $\mathcal{T}$  over its hyperperiod (20s), replicating the scheduling in the case of clusters whose hyperperiod is less than the hyperperiod of  $\mathcal{T}$  (Fig. 9). Each cluster hyperperiod is always a divisor of the hyperperiod of the initial HRT task set. In a system with no on-line aperiodic task or disturbance management, this is the resulting thermal-safe HRT schedule, at minimum frequency, with low context switches and migrations, which can be implemented as a cyclic executive in a straightforward manner.

E. GET-REFERENCE

In a system where we have to cope with SRT aperiodic tasks we can apply the on-line modules *Get-reference* (Sec. VIII) and the feedback controller (Sec. IX). The input to *Get-reference* is the output of the precedent module *Pre-scheduler*, i.e. a cyclic executive (Fig. 9). The first step consists in determining the set  $D$  up to the hyperperiod. Considering these intervals and the cyclic executive, we easily obtain the cycles each task must run on each CPU at each interval  $\delta_k$  (Table 6). The references in this table are the input for the on-line feedback controller at the next stage.

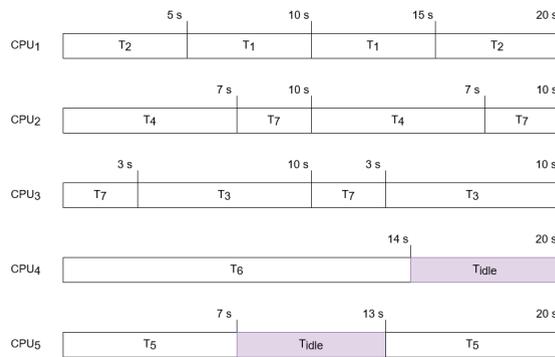


FIGURE 9. Cyclic executive for the HRT task set in Table 3.

TABLE 6. Intervals and task cycles per-interval  $\delta_k$  and task obtained from the cyclic executive in Fig. 9 and the intervals determined by the context switches.

	$\delta_1$ [0, 3)	$\delta_2$ [3, 5)	$\delta_3$ [5, 7)	$\delta_4$ [7, 10)	$\delta_5$ [10, 13)	$\delta_6$ [13, 14)	$\delta_7$ [14, 15)	$\delta_8$ [15, 17)	$\delta_9$ [17, 20)
T1	0	0	2	3	3	1	1	0	0
T2	3	2	0	0	0	0	0	2	3
T3	0	2	2	3	0	1	1	2	3
T4	3	2	2	0	3	1	1	2	0
T5	3	2	2	0	0	1	1	2	3
T6	3	2	2	3	3	1	0	0	0
T7	3	0	0	3	3	0	0	0	3
Tidle	0	0	0	3	3	0	1	2	3

F. FEEDBACK CONTROLLER

In a real system running this example, the feedback controller would behave just as described in Sec. IX, subject to the arrival of SRT aperiodic tasks or the occurrence of small run-time disturbances caused by parameter variations. This might cause additional context switches.

To demonstrate the behavior of the feedback controller, assume that an aperiodic task with a relative deadline of 9 s and WCET of 7 s, when running at  $F = 1$  Hz, arrives at  $t = 1$  s. In Fig.10 we see that the feedback controller increases the frequency of the processors at  $t = 1$  s to 1.5 Hz, holding it up to  $t = 7$  s, in order to allow the execution of the aperiodic task. The feedback controller will be able to execute the aperiodic task during this interval meeting the timing constraints, then restoring the previous frequency and the execution references at  $t=7s$ .

G. THERMAL MANAGEMENT

To demonstrate the proper behavior of our thermal control, we have performed a simulation with the processor distribution shown in Fig. 7. The MPSoC is cooled down by forced air, with a heat transfer coefficient of  $500 W/m^2 * K$  [48].

Fig. 11 displays the temperature evolution at the center of the cores when the system is running the cyclic executive. Aligned side-by-side, Fig. 12 shows the temperature evolution at the same places when the system manages an aperiodic task throttling the frequency during that interval. The core temperature increases along the interval [1, 7], then decreases when the frequency is set back to  $F^*$ , once normal execution resumes.

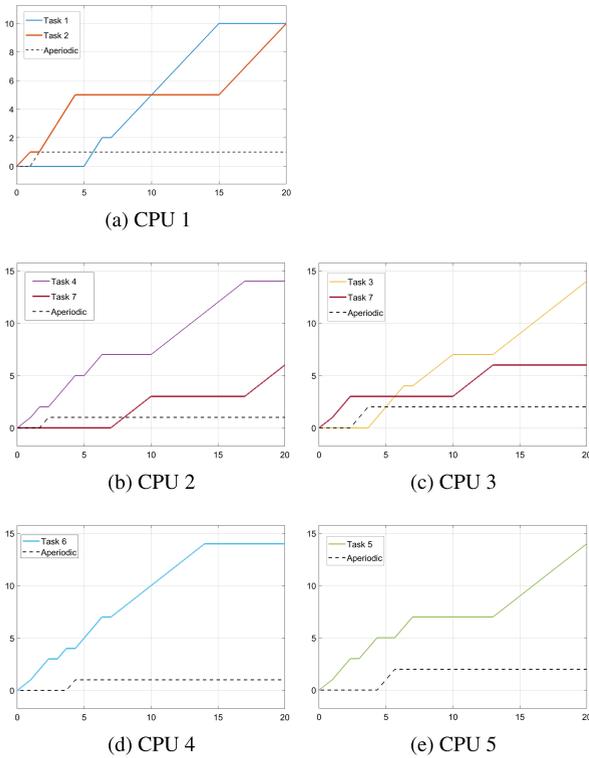


FIGURE 10. Execution of tasks with the admission of an aperiodic task.

XI. COMPARISON WITH RUN

CAIECS aims to maximize processor utilization and to minimize job migration and context switching. In this section we compare the number of context switches and migrations entailed by CAIECS, AIECS and RUN, the latter of which is considered as an optimal reference (Sec. II). We include AIECS because it is just the worst-case clustering solution of CAIECS —the situation in which there is a single (major) cluster encompassing the whole HRT task set and all the CPUs.

A. SIMULATION ENVIRONMENT AND SETUP

We carry out the comparison using Tertimuss [11], an open-source framework to model a RT multiprocessor system, simulate different RT schedulers, and process the results. We only consider job preemptions as context switches. We rule out job start and job termination events, since they are independent of the scheduling algorithm. We compute a migration when a job resumes execution on a CPU different from the one on which it was previously running.

The task sets for the comparison are generated using the UUniFast-discard algorithm [49]. The total utilization of each task set is equal to the number of processors in the experiment. Task periods are randomly selected between the divisors of 60, to obtain a major cycle of at most 60 s. The task sets are executed on systems with 2 and 4 cores, with task-to-core ratios of 4, 8, 12, 16, and 20. This amounts to 200 experiments per combination, totaling 2000 experiments in all.

Tertimuss takes the WCET in cycles, and performs the simulation on a cycle-by-cycle basis. Since RUN can yield

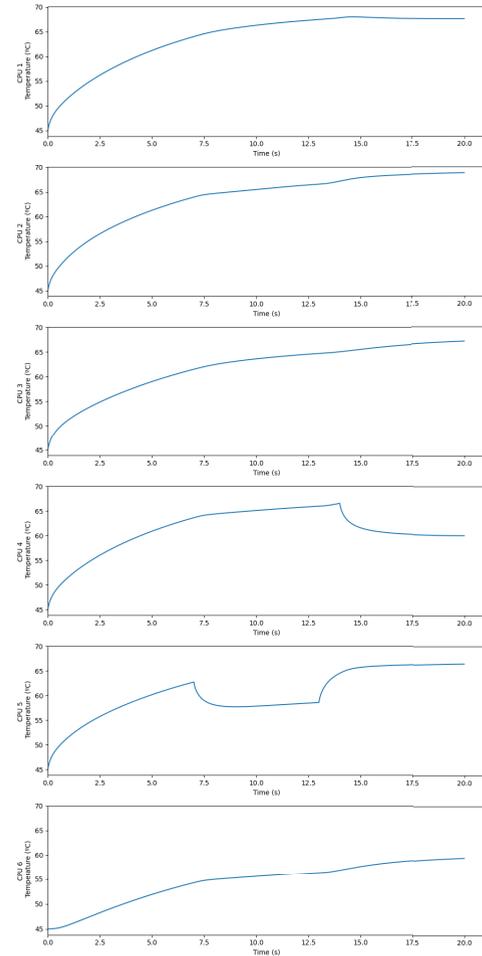


FIGURE 11. Processors temperature evolution of the cyclic executive.

fractional execution times, we apply some adjustments to make the comparison feasible. Following the original description of the algorithm in [9], we implement RUN using a Worst-Fit policy for the packing operation, the EDF policy for the scheduling of the servers, and the original task-to-processor allocation policy. When applying EDF, deadline ties in servers are either solved at random if no server has run yet, or by choosing the server that was last executed, otherwise. Since the three schedulers in the comparison are optimal and the selected task sets are feasible, the three schedulers obtain a feasible schedule in all experiments.

B. MIGRATIONS PER JOB

The boxplots in Fig. 13 summarizes the number of migrations per job (Y-axis) as the number of tasks per processor (TPP) varies (X-axis). Tab. 7 details the mean and the standard deviation. A common trend to all the three schedulers under comparison is that both the mean and the standard deviation of migrations per job decrease as the number of processors decreases and the TPP increases. These results support the intuition discussed in Sec. VI, upon the rationale that a minor cluster yields zero migrations.

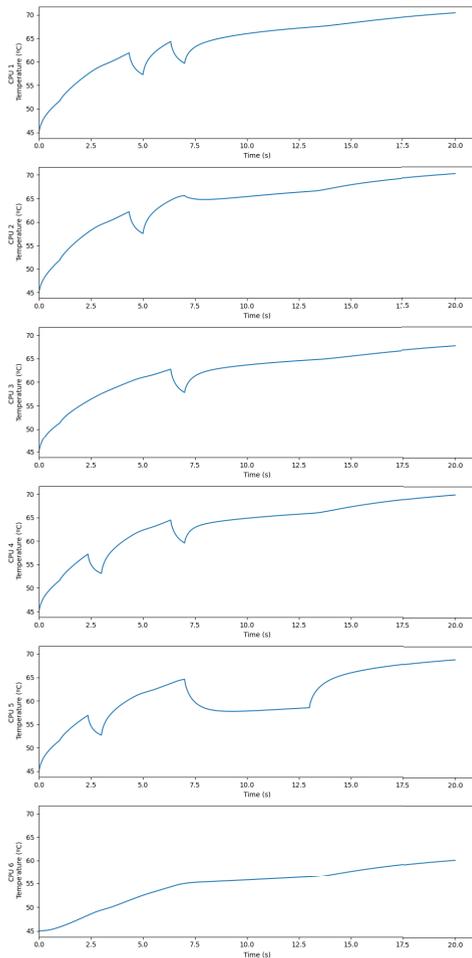


FIGURE 12. Processors temperature evolution of the cyclic executive with the admission of an aperiodic tasks.

TABLE 7. Number of migrations per job depending on the tasks-per-processor ratio (TPP).

m	n	RUN		AIECS		CAIECS	
		Mean	SD	Mean	SD	Mean	SD
2	8	0,676	0,375	0,320	0,096	0,298	0,122
	16	0,407	0,254	0,054	0,054	0,193	0,119
	24	0,224	0,192	0,196	0,039	0,113	0,101
	32	0,145	0,151	0,162	0,033	0,059	0,083
	40	0,087	0,117	0,130	0,024	0,032	0,058
4	16	0,957	0,283	0,516	0,096	0,431	0,140
	32	0,526	0,165	0,359	0,061	0,192	0,099
	48	0,316	0,153	0,270	0,041	0,090	0,070
	64	0,182	0,135	0,203	0,031	0,041	0,048
	80	0,108	0,112	0,159	0,024	0,014	0,028

AIECS achieves fewer migrations than RUN with low TPP ratios, but RUN outperforms AIECS with high TPP ratios (16, 20). Also, the mean of migrations decreases faster in RUN than in AIECS as the TPP increases. Last, AIECS fails to reach zero migrations ever, while RUN yields a Q1 of zero with high TPP ratios, also in the case of the 24/2 TPP, and reaches a median of zero in the case 40 tasks executed in two CPUs. However, the standard deviation is outstandingly lower in AIECS than in RUN in all cases. The analysis of the behavior of RUN with high TPP ratios shows that RUN

TABLE 8. Clustering performance.

m	n	Distribution of CPUs in clusters	CAIECS	RUN
2	8	1,1 2	13 187	6 194
	16	1,1 2	49 151	33 167
	24	1,1 2	85 115	72 128
	32	1,1 2	132 68	92 108
	40	1,1 2	152 48	120 80
4	16	1, 1, 1, 1	0	0
		1, 1, 2	7	0
		2, 2	15	0
		1, 3	62	17
		4	116	183
	32	1, 1, 1, 1	7	4
		1, 1, 2	97	9
		2, 2	11	0
		1, 3	60	40
		4	25	147
	48	1, 1, 1, 1	53	20
		1, 1, 2	113	21
2, 2		9	0	
1, 3		22	59	
4		3	100	
64	1, 1, 1, 1	108	56	
	1, 1, 2	85	20	
	2, 2	2	0	
	1, 3	4	56	
	4	1	68	
80	1, 1, 1, 1	160	89	
	1, 1, 2	38	26	
	2, 2	1	0	
	1, 3	1	32	
	4	0	53	

manages to find a full partition of the task-set (minor cluster) during its packing operation in a considerable amount of experiments, leading to a zero migrations figure in those cases.

The results prove that CAIECS outperforms AIECS and RUN in all configurations because it is able to find as many partitionable task-sets as RUN, while retaining the properties of AIECS when scheduling major clusters. Also, it can reach a median of zero migrations when the TPP ratio is high, even yielding a Q3 of zero in the case of 80 tasks executed in four CPUs. The standard deviation is lower in CAIECS than in RUN but higher than in AIECS.

RUN is not specifically designed to find clusters, particularly minor clusters (achieving zero migrations), but its ability to find them often, when possible, allows RUN to reach the zero migrations notch in cases where AIECS fails to do this. The *task clustering* stage in CAIECS aims to find low-size clusters, reaching zero migrations in more experiments than RUN (Fig. 13). We can examine Table 8 to better understand the clustering behavior of RUN and CAIECS (AIECS does not apply clustering) and to assess the consistency of our experimental results. The third column in this table shows the number of clusters of each size (from 1 to 4 CPUs) obtained for each TPP. For example, with 64 tasks on four

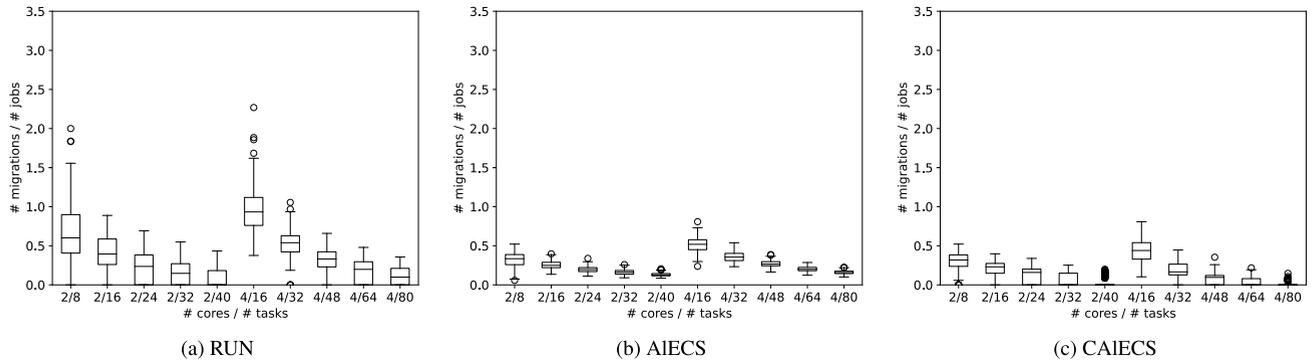


FIGURE 13. Simulation results for the number of migrations/number of jobs.

TABLE 9. Number of preemptions per job.

m	n	RUN		AIECS		CAIECS	
		Mean	SD	Mean	SD	Mean	SD
2	8	1,449	0,608	0,575	0,117	0,561	0,132
	16	0,832	0,291	0,431	0,082	0,410	0,090
	24	0,495	0,199	0,313	0,058	0,288	0,063
	32	0,367	0,157	0,249	0,047	0,228	0,049
	40	0,267	0,114	0,199	0,034	0,183	0,035
4	16	1,362	0,307	0,653	0,116	0,614	0,120
	32	0,816	0,172	0,433	0,074	0,371	0,068
	48	0,534	0,137	0,317	0,049	0,273	0,047
	64	0,370	0,116	0,236	0,035	0,214	0,034
	80	0,266	0,090	0,184	0,026	0,174	0,025

processors, CAIECS finds a full partition (four minor clusters) in 108 experiments, whereas RUN does it only 56 times. In 85 experiments, CAIECS fits the HRT task set into two minor clusters and a major cluster with two CPUs, whereas RUN only finds that configuration in 20 experiments. This trend holds for all design points, with CAIECS always finding substantially more minor clusters (entailing zero migrations) than RUN.

C. PREEMPTIONS PER JOB

The boxplots in Fig. 14 and Tab. 9 help to analyze the mean of preemptions per job by experiment for each TPP configuration. The results indicates that AIECS always perform better than RUN on the average, yielding a lower standard deviation and fewer outliers. However, there are some specific cases where RUN outperform AIECS, such as when scheduling 8 tasks over two CPUs, with RUN providing a minimum value of 0.09091 while AIECS reaches 0.2593. As for CAIECS, it outperforms RUN and AIECS on the average, with a slightly greater standard deviation than AIECS nonetheless.

As we observed with migrations, the mean of preemptions and its standard deviation decreases in all the three schedulers as the TPP ratio increases. Also, for each TPP ratio, the mean of preemptions decreases with the number of CPUs.

XII. COMPUTATIONAL COMPLEXITY

This section gathers the computational complexity of the various parts of the proposed system as described in Fig. 2. The

off-line stages calculate the minimum and maximum frequencies, resolve the task clustering, and apply a pre-scheduling which, depending on the container size of each cluster, follows the EDF or ZL dispatching rules. The on-line stage consists on a continuous controller and the aperiodic manager.

A. OFF-LINE CALCULATIONS

The Task Set Conditioner (Sec. V) calculates the minimum and maximum frequencies ( $F^*$  and  $F^+$ ). Computing  $F^*$  is linear in the number of tasks  $\mathcal{O}(n)$ . The calculation of  $F^+$  requires solving a non-linear optimization problem, whose number of iterations to find a solution depends on some parameters such as the required convergence error and the gradient weighting. To this end, we employ an interior point algorithm. Tests show that it provides a good performance and converges to the optimum in a very short time.

The computational complexity of our task clustering (Sec. VI) depends on Alg. 1. The outer `while` loop (line 9) iterates  $m$  times. At each iteration it first executes `SolveBPP` (line 10), whose complexity is  $n \times \log(n)$ . Then, it runs the inner `for` loop. All instructions inside the inner `for` loop are executed  $\alpha \times m$  times. Thus, the computational complexity of Alg. 1 is of order  $\mathcal{O}(m^2 \times n \times \log(n))$ .

Next, the complexity of solving Eq. (22) to compute the workload per scheduling point is linear in  $x_i^k$ , i.e. in  $n^2 \times \beta$ , where  $\beta = \max_{\tau_i} \frac{H}{w_i}$ . Thus, it is in the order of  $\mathcal{O}(n^2)$ . Finally, EDF or ZL (Alg.2) policies run in polynomial time.

Hence, all the algorithms in the off-line stage run in polynomial time except, but the non-linear optimization. As mentioned before, in our tests, the non-linear optimization algorithm runs in a very short time.

B. ON-LINE COMPUTATIONS

References are only computed when a context switch occurs, which results on gathering the appropriate execution requirement  $R_{i,j}(d_k)$  (Ec. 27).

The feedback controller (Sec. IX) runs at every sampling period, which depends on the minimum difference between two consecutive  $d_i - d_{i-1}$  from set  $D$ . Since the RT clock routine is executed with a fixed period and more frequently

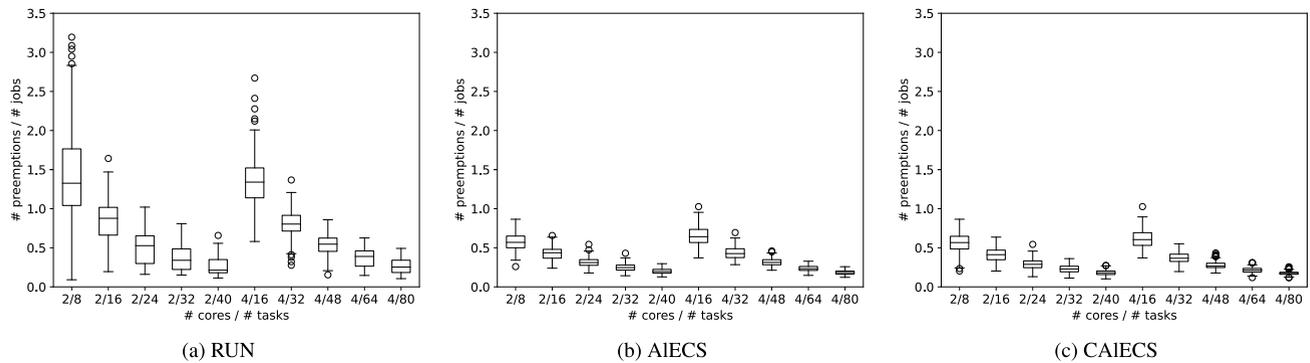


FIGURE 14. Simulation results for the number of preemption/number of jobs.

than any other RT task, we propose to implement the feedback controller routine as a call back (deferred function, softirq or tasklet depending on the RT operating system), activated at each RT clock routine execution. In this approach we include a feedback controller routine per CPU. Thus, their complexity is  $\mathcal{O}(1)$ , because it requires to solve Eqs. (31) and (35), that have a fixed number of operations. A different implementation approach may use a devoted CPU for the feedback controller routine.

### XIII. CONCLUSION

RT scheduling on multicore processors remains a challenge in many ways, all the more when temperature and energy counts. One of the points is maximizing processor utilization to avoid overprovisioning. Partitioned scheduling approaches fall short in this aspect, whereas optimal global schedulers are too complex to be effective. Near-optimal solutions are possible by leveraging a mix of ad-hoc techniques, but there is still a shortage of practical methods and tools interesting enough for the industry to adopt them.

CAIECS leverages the ability of known scheduling algorithms to simplify the global scheduling problem, and the power of a fluid model and feasible mathematical optimization to account for thermal constraints and to maximize processor utilization. It provides an off-line cyclic executive for a HRT task set which is thermal-safe, energy efficient, easy to implement and more effective than RUN. Our comparison with RUN reveals that CAIECS is able to find more minor clusters and more major clusters of smaller size, along with the fact that the global scheduler AIECS performs particularly well in lowering migrations in major clusters.

As a cyclic executive, the number of context switches and migrations is known for the hyperperiod, and therefore the WCET, which usually includes scheduling costs, can be fine-tuned in future work to obtain a schedule with more realistic bounds. This cyclic executive can feed an on-line controller which manages the SRT aperiodic tasks, deals with small disturbances, due to parameter variations for example. In the future, it can easily fit a slack reclamation scheme. The modular design of CAIECS permits designers to employ other thermal models or schedulers.

### REFERENCES

- [1] K. Vipin, "CANNoC: An open-source NoC architecture for ECU consolidation," in *Proc. IEEE 61st Int. Midwest Symp. Circuits Syst. (MWSCAS)*, Aug. 2018, pp. 940–943.
- [2] Intel. *ECU Consolidation Reduces Vehicle Cost, Weight and Testing*. Accessed: May 1, 2021. [Online]. Available: <https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/ecu-consolidation-white-paper.pdf>
- [3] D.-I. Oh and T. P. Bakker, "Utilization bounds for n-processor rate monotone scheduling with static processor assignment," *Real-Time Syst.*, vol. 15, no. 2, pp. 183–192, 1998.
- [4] A. Mascitti, T. Cucinotta, and L. Abeni, "Heuristic partitioning of real-time tasks on multi-processors," in *Proc. IEEE 23rd Int. Symp. Real-Time Distrib. Comput. (ISORC)*, May 2020, pp. 36–42.
- [5] *Specification 651: Design Guide for Integrated Modular Avionics*, ARINC, Annapolis, MD, USA, 1997.
- [6] N. Diniz and J. Rufino, "ARINC 653 in space," in *Proc. DASIA-Data Syst. Aerosp.*, vol. 602. Edinburgh, Scotland: ESA Publications Division, May 2005.
- [7] AUTOSAR. (2017). *Specification of RTE Software*. [Online]. Available: <http://www.autosar.org>.
- [8] B. B. Brandenburg and M. Gül, "Global scheduling not required: Simple, near-optimal multiprocessor real-time scheduling with semi-partitioned reservations," in *Proc. IEEE Real-Time Syst. Symp. (RTSS)*, Nov. 2016, pp. 99–110.
- [9] P. Regnier, G. Lima, E. Massa, G. Levin, and S. Brandt, "RUN: Optimal multiprocessor real-time scheduling via reduction to uniprocessor," in *Proc. IEEE 32nd Real-Time Syst. Symp. (RTSS)*, Washington, DC, USA, Nov. 2011, pp. 104–115.
- [10] M. L. Dertouzos, "Control robotics: The procedural control of physical processes," in *Proc. IFIP Congr. (IFIP)*, 1974, pp. 807–813.
- [11] G. Desirena, L. Rubio, A. Ramirez, and J. Briz. (2019). *Thermal-Aware HRT Scheduling Simulation Framework*. [Online]. Available: <https://webdiis.unizar.es/gaz/repositories/termimuss>.
- [12] S. Sha, W. Wen, G. A. Chaparro-Baquero, and G. Quan, "Thermal-constrained energy efficient real-time scheduling on multi-core platforms," *Parallel Comput.*, vol. 85, pp. 231–242, Jul. 2019. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0167819118300280>
- [13] G. Desirena-Lopez, A. Ramirez-Treviño, J. Briz, C. Vázquez, and D. Gómez-Gutiérrez, "Thermal-aware real-time scheduling using timed Continuous Petri nets," *ACM Trans. Embedded Comput. Syst.*, vol. 18, no. 4, p. 36, 2019.
- [14] L. Rubio-Anguiano, G. Desirena-López, A. Ramirez-Treviño, and J. Briz, "Energy-efficient thermal-aware multiprocessor scheduling for real-time tasks using TCPN," *Discrete Event Dyn. Syst.*, vol. 29, pp. 1–28, Jul. 2019.
- [15] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel, "Proportionate progress: A notion of fairness in resource allocation," *Algorithmica*, vol. 15, no. 6, pp. 600–625, Jun. 1996.
- [16] S. K. Baruah, J. E. Gehrke, and C. G. Plaxton, "Fast scheduling of periodic tasks on multiple resources," in *Proc. IPPS*, 1995, p. 280.
- [17] J. H. Anderson and A. Srinivasan, "Mixed Pfair/ERfair scheduling of asynchronous periodic tasks," in *Proc. 13th Euromicro Conf. Real-Time Syst.*, 2001, pp. 76–85.
- [18] N. Fisher, J. Goossens, and S. Baruah, "Optimal online multiprocessor scheduling of sporadic real-time tasks is impossible," *Real-Time Syst.*, vol. 45, nos. 1–2, pp. 26–71, Jun. 2010.

- [19] M. Leoncini, M. Montangero, and P. Valente, "A parallel branch-and-bound algorithm to compute a tighter tardiness bound for preemptive global EDF," *Real-Time Syst.*, vol. 55, no. 2, pp. 349–386, Apr. 2019.
- [20] S. Funk, G. Levin, C. Sadowski, I. Pye, and S. Brandt, "DP-fair: A unifying theory for optimal hard real-time multiprocessor scheduling," *Real-Time Syst.*, vol. 47, no. 5, pp. 389–429, Sep. 2011.
- [21] G. Nelissen, V. Berten, J. Goossens, and D. Milojevic, "Reducing preemptions and migrations in real-time multiprocessor scheduling algorithms by releasing the fairness," in *Proc. IEEE 17th Int. Conf. Embedded Real-Time Comput. Syst. Appl.*, vol. 1, Aug. 2011, pp. 15–24.
- [22] M. Thammawichai and E. C. Kerrigan, "Energy-efficient real-time scheduling for two-type heterogeneous multiprocessors," *Real-Time Syst.*, vol. 54, no. 1, pp. 132–165, Jan. 2018.
- [23] J. Zhou, J. Sun, P. Cong, Z. Liu, X. Zhou, T. Wei, and S. Hu, "Security-critical energy-aware task scheduling for heterogeneous real-time MPSoCs in IoT," *IEEE Trans. Services Comput.*, vol. 13, no. 4, pp. 745–758, Jul. 2020.
- [24] A. Bertout, J. Goossens, E. Grolleau, and X. Poczekajko, "Template schedule construction for global real-time scheduling on unrelated multiprocessor platforms," in *Proc. Design, Automat. Test Eur. Conf. Exhib. (DATE)*, Mar. 2020, pp. 216–221.
- [25] S. Baruah, "Feasibility analysis of preemptive real-time systems upon heterogeneous multiprocessor platforms," in *Proc. 25th IEEE Int. Real-Time Syst. Symp.*, Dec. 2004, pp. 37–46.
- [26] S. Moulik, R. Devaraj, and A. Sarkar, "HEART: A heterogeneous energy-aware real-time scheduler," in *Proc. 32nd Int. Conf. VLSI Design, 18th Int. Conf. Embedded Syst. (VLSI/IC)*, Jan. 2019, pp. 476–481.
- [27] D. Doan and K. Tanaka, "A novel task-to-processor assignment approach for optimal multiprocessor real-time scheduling," in *Proc. IEEE 12th Int. Symp. Embedded Multicore/Many-Core Syst.-Chip (MCSoC)*, Hanoi, Vietnam, Sep. 2018, pp. 101–108.
- [28] D. Doan and K. Tanaka, "Adaptive local assignment algorithm for scheduling soft-aperiodic tasks on multiprocessors," in *Proc. IEEE 25th Int. Conf. Embedded Real-Time Comput. Syst. Appl. (RTCSA)*, Aug. 2019, pp. 1–6.
- [29] P. Regnier, G. Lima, E. Massa, G. Levin, and S. Brandt, "Multiprocessor scheduling by reduction to uniprocessor: An original optimal approach," *Real-Time Syst.*, vol. 49, no. 4, pp. 436–474, Jul. 2013.
- [30] D. Compagnin, E. Mezzetti, and T. Vardanega, "Putting RUN into practice: Implementation and evaluation," in *Proc. 26th Euromicro Conf. Real-Time Syst.*, Jul. 2014, pp. 75–84.
- [31] E. Massa, G. Lima, P. Regnier, G. Levin, and S. Brandt, "Quasi-partitioned scheduling: Optimality and adaptation in multiprocessor real-time systems," *Real-Time Syst.*, vol. 52, no. 5, pp. 566–597, Sep. 2016.
- [32] E. Massa, G. Lima, and P. Regnier, "Revealing the secrets of RUN and QPS: New trends for optimal real-time multiprocessor scheduling," in *Proc. Brazilian Symp. Comput. Syst. Eng.*, Nov. 2014, pp. 150–155.
- [33] J. Mayank and A. Mondal, "Non-preemptive multiprocessor scheduling for periodic real-time tasks," in *Proc. 7th Int. Symp. Embedded Comput. Syst. Design (ISED)*, Dec. 2017, pp. 1–6.
- [34] B. Kim and H. Yang, "Reliability optimization of real-time satellite embedded system under temperature variations," *IEEE Access*, vol. 8, pp. 224549–224564, 2020.
- [35] P. Dziurzanski and A. Singh, "Feedback-based admission control for firm real-time task allocation with dynamic voltage and frequency scaling," *Computers*, vol. 7, no. 2, p. 26, Apr. 2018. [Online]. Available: <https://www.mdpi.com/2073-431X/7/2/26>
- [36] L. Rubio-Anguiano, G. Ramírez-Treviño, J. Briz, and A. Chils, "Real time scheduler for multiprocessor systems based on continuous control using timed continuous Petri nets," *IFAC-PapersOnLine*, vol. 53, no. 4, pp. 371–377, 2020.
- [37] G. Desirena-Lopez, C. R. Vázquez, A. Ramírez-Ireviño, and D. Gómez-Gutiérrez, "Thermal modelling for temperature control in MPSOC's using timed continuous Petri nets," in *Proc. IEEE Conf. Control Appl. Part Multi-Conf. Syst. Control*, Oct. 2014, pp. 2135–2140.
- [38] S. Baruah, M. Bertogna, and G. Butazzo, *Multiprocessor Scheduling for Real-Time Systems*. Secaucus, NJ, USA: Springer-Verlag, 2015.
- [39] B. B. Brandenburg, "Scheduling and locking in multiprocessor real-time operating systems," Ph.D. dissertation, Univ. North Carolina Chapel Hill, Chapel Hill, NC, USA, 2011.
- [40] M. Silva and L. Recalde, "Redes de Petri continuas: Expresividad, análisis y control de una clase de sistemas lineales conmutados," *Revista Iberoamericana Automática Informática Ind.*, vol. 4, no. 3, pp. 5–33, Jul. 2007.
- [41] R. David and H. Alla, "Discrete, continuous and hybrid Petri nets," *IEEE Control Syst. Mag.*, vol. 28, no. 3, pp. 81–84, Jun. 2008.
- [42] M. Silva, J. Júlvez, C. Mahulea, and C. R. Vázquez, "On fluidization of discrete event models: Observation and control of continuous Petri nets," *Discrete Event Dyn. Syst.*, vol. 21, no. 4, pp. 427–497, Dec. 2011.
- [43] R. Ahmed, P. Ramanathan, and K. K. Saluja, "Necessary and sufficient conditions for thermal schedulability of periodic real-time tasks under fluid scheduling model," *ACM Trans. Embedded Comput. Syst.*, vol. 15, no. 3, p. 49, 2016.
- [44] D. S. Johnson, "Fast algorithms for bin packing," *J. Comput. Syst. Sci.*, vol. 8, no. 3, pp. 272–314, Jun. 1974.
- [45] K. Truemper, *Matroid Decomposition*, vol. 6. Boston, MA, USA: Academic, 1992.
- [46] A. J. Hoffman and J. B. Kruskal, "Integral boundary points of convex polyhedra," in *50 Years of Integer Programming 1958–2008*. Berlin, Germany: Springer, 2010, pp. 49–76.
- [47] H. K. Khalil, *Nonlinear Systems*. Upper Saddle River, NJ, USA: Prentice-Hall, 2002.
- [48] P. Kosky, R. Balmer, W. Keat, and G. Wise, "Mechanical engineering," in *Exploring Engineering*, P. Kosky, R. Balmer, W. Keat, and G. Wise, Eds., 5th ed. New York, NY, USA: Academic, 2021, ch. 14, pp. 317–340. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/B9780128150733000144>
- [49] R. I. Davis and A. Burns, "Priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems," in *Proc. 30th IEEE Real-Time Syst. Symp.*, Dec. 2009, pp. 398–409.



**LAURA ELENA RUBIO-ANGUIANO** received the B.Sc. degree in mechatronic engineering from the Instituto Tecnológico de Estudios Superiores de Monterrey, Mexico, in 2015, and the M.Sc. degree in electrical engineering from the Cinvestav Unidad Guadalajara, Mexico, in 2018. She is currently pursuing the Ph.D. degree with Cinvestav and Universidad de Zaragoza, in co-tutelage.



**ABEL CHILS TRABANCO** received the B.Sc. degree in computer engineering from the University of Zaragoza, Spain, in 2019, where he is currently pursuing the M.Sc. degree.



**JOSÉ LUIS BRIZ VELASCO** received the B.Sc. and M.Sc. degrees in geology, the M.Sc. degree in computer science, and the Ph.D. degree in computer engineering from the University of Zaragoza (UZ), Spain, in 1996. He is currently an Associate Professor with the Department of Computer and Systems Engineering and a Researcher at the I3A Research Institute, UZ. His research interests include memory hierarchy, processor microarchitecture, and real-time systems. He is a member of the GAZ Group and the Spanish Society of Computer Architecture (SARTECO). He is also an Affiliate Member of the HiPEAC European Network of Excellence.



**ANTONIO RAMÍREZ-TREVIÑO** received the B.Sc. degree in electrical engineering from Universidad Autónoma Metropolitana, Mexico City, Mexico, in 1986, the M.Sc. degree from Cinvestav, Mexico, in 1990, and the Ph.D. degree from the University of Zaragoza, Spain, in 1993. He is currently a Professor with Cinvestav Unidad Guadalajara, Mexico. His research interests include scheduling, analysis, and control of discrete event systems, including controllability, observability, and stability.

...