CINVESTAV

Centro de Investigación y de Estudios Avanzados del I.P.N.
Unidad Guadalajara

Thermal-Aware Hard Real Time Task Scheduling in MPSoC's using Timed Continuous Petri Nets

Doctoral Thesis in Electrical Engineering

Presented by:

Gaddiel Desirena López

to obtain the degree of:

Doctor in Science

In the subject of:

Electrical Engineering

Thesis Advisors

Dr. Antonio Ramírez Treviño

Dr. José Luis Briz Velasco

Guadalajara, Jalisco. June, 2019

CENTRO DE INVESTIGACION Y DE ESTUDIOS AVANZADOS DEL INSTITUTO POLITECNICO NACIONAL

Copia del Acta, inscrita a fojas sesenta y uno del Libro Centésimo Cuadragésimo Quinto, del Examen Final presentado por el C. Gaddiel Desirena López para obtener el grado de Doctor en Ciencias en la especialidad de Ingeniería Eléctrica.

En la Ciudad de Guadalajara, Jalisco, a los veintiocho días del mes de junio del año dos mil diecinueve, se reunieron en la Unidad Guadalajara del Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional: la doctora Ofelia Begovich Mendoza, el doctor Antonio Ramírez Treviño y el doctor Arturo Díaz Pérez, todos ellos investigadores titulares de la mencionada Unidad; el doctor Pedro Mejía Álvarez, Investigador Titular del Departamento de Computación de la Unidad Zacatenco, Ciudad de México; y, como sinodales invitados por el Centro, el doctor José Luis Briz Velasco, Profesor Titular de Arquitectura y Tecnología de Computadores del Departamento de Informática e Ingeniería de Sistemas en la Escuela de Ingeniería y Arquitectura e Investigador del Instituto de Investigación en Ingeniería de Aragón de la Universidad de Zaragoza, en España; y el doctor David Gómez Gutiérrez, Investigador de Intel Labs de Intel Tecnologías de México, S.A. de C.V., a fin de efectuar el Examen Final que para obtener el grado de Doctor en Ciencias sustentó el C. Gaddiel Desirena López.

El C. Gaddiel Desirena López presentó la tesis titulada: "Planificación de Tareas en Tiempo Real Consientes de la Temperatura usando Redes de Petri Continuas Temporizadas", que fue desarrollada bajo la dirección del doctor Antonio Ramírez Treviño y del doctor José Luis Briz Velasco.

Con fundamento en los resultados de este examen, el Jurado dictaminó que el C. Gaddiel Desirena López aprobó el Examen Final para obtener el grado de Doctor en Ciencias en la especialidad de Ingeniería Eléctrica.

Ofelia Begovich Mendoza

Antonio Ramírez Treviño

Arturo Díaz Pérez

Pedro Mejía Álvarez

José Luis Briz Velasco

David Gómez Gutiérrez

El Director General del Centro certifica que las firmas que anteceden son auténticas.

José Mustre de León

Thermal-Aware Hard Real Time Task Scheduling in MPSoC's using Timed Continuous Petri Nets

Doctoral Thesis in Electrical Engineering

Presented by:
Gaddiel Desirena López

Scholarship granted by CONACYT, No.282123

Thesis Advisors
Dr. Antonio Ramírez Treviño
Dr. José Luis Briz Velasco

CINVESTAV del IPN Unidad Guadalajara. June, 2019

Resumen

Los planificadores en tiempo real (RTS, por sus siglas en inglés) son uno de los temas de investigación más importantes en sistemas operativos. Este tipo de planificadores se centran en asignar tareas a los CPUs para cumplir con restricciones temporales que las tareas o aplicaciones tienen. La omnipresencia de los RTS abarca desde sistemas críticos en tiempo real, como dispositivos médicos o controladores automáticos, hasta aplicaciones de entretenimiento, como videojuegos. Además, los dispositivos modernos, como teléfonos celulares, dispositivos médicos y otros dispositivos portátiles introducen nuevas restricciones, como cumplir en no superar los límites térmicos o reducir la energía consumida. Por ejemplo, la energía consumida por un audífono médico (aparato para el oído), debe ser lo más baja posible, y la temperatura en este dispositivo debe ser cómodo para los pacientes. En las industrias automotriz y aeroespacial, el control térmico también es primordial para evitar acortar la vida útil de los componentes, además de minimizar el consumo de energía cuando la energía es escasa, como ocurre en los sistemas satelitales alimentados por paneles solares.

El problema que se aborda en esta Tesis, se centra en el diseño de planificadores en tiempo real para sistemas multiprocesadores en chip (MPSoC), que también toman en cuenta las restricciones térmicas y de energía. Dado que este problema abarca problemas discretos y continuos, el enfoque adoptado se basa en redes de Petri continuas temporizadas (TCPN, por sus siglas en inglés), un formalismo capaz de capturar el comportamiento de señales continuas positivas, como la temperatura o la energía, y simular comportamientos discretos, como ejecuciones de tareas. Una conjetura que motivó esta investigación fue que combinando técnicas de algoritmos combinatoriales y de control automático es posible derivar mejores planificadores en tiempo real, donde utilizando un controlador continuo es posible solucionar el problema de la NP-Complejidad y el rechazo de perturbaciones, como detenciones de CPU, llegada de tareas aperiódicas o incertidumbres de modelado. Y mediante el análisis combinatorio es posible discretizar una planificación continua. Con esta investigación, se puede confirmar que la conjetura es cierta.

El Capítulo 1 proporciona algunos antecedentes, establece la justificación de la Tesis, los objetivos, y formaliza el problema aquí estudiado.

El Capítulo 2 propone una metodología de modelado novedosa que representa, en un

modelo monolítico: la llegada de tareas, la asignación de tareas, la generación de calor, el consumo de energía y la selección de frecuencia mediante una única herramienta formal. Esta metodología brinda grandes ventajas en comparación a las técnicas previamente reportadas en la literatura. Por ejemplo, el modelo es tan preciso como un modelo de elementos finitos. Se representa en una ecuación de variable de estado, susceptible para fines de control y propósitos de planificación en tiempo real. Más aún, la metodología evita las etapas de calibración que otros enfoques realizan. Un punto clave de esta metodología de modelado radica en el hecho de que la ecuación diferencial parcial que representa el comportamiento del sistema se aproxima a un conjunto de ecuaciones diferenciales ordinarias de primer orden, lo que reduce considerablemente la complejidad del análisis.

En este capítulo se presenta una herramienta de software para simular el modelo y los planificadores propuestos.

El Capítulo 3 presenta un primer planificador de tiempo real, que se enfoca en cumplir con las restricciones temporales para un conjunto de tareas periódicas e independientes. Con este propósito, el planificador calcula los tiempos de ejecución requeridos de cada tarea y los representa como una función fluida (continua) de ejecución. Un controlador por modos deslizantes garantiza que todas las tareas se ejecuten según lo indicado por la función previamente calculada, obteniendo una planificación continua. Finalmente, la planificación continua es discretizada mediante un algoritmo de discretización.

El Capítulo 4 extiende el controlador anteriormente propuesto para incluir variables térmicas. Dado que las variables térmicas y temporales deben controlarse y sabiendo que sólo existe una única variable de control en el modelo, se concluye que el sistema no es controlable. Sin embargo, para solucionar este problema, se propone un planificador fluido que cumple con las restricciones tanto temporales y térmicas. Utilizando un problema de programación lineal se obtiene una conjunto de funciones fluidas de ejecución que toma en cuenta el cumplimiento de las restricciones térmicas y temporales, luego mediante un controlador de modos deslizantes se forza al sistema a seguir estas nuevas funciones fluidas incluso en presencia de perturbaciones, obteniendo así una planificación continua. El planificador continuo se discretiza con un enfoque que equilibra la precisión en el control térmico y la sobrecarga de tareas.

El Capítulo 5 introduce un planificador que es capaz de minimizar la energía y cumplir con las restricciones temporales de las tareas y las térmicas del sistema. Más aún, el planificador es capaz de gestionar la llegada de tareas aperiódicas, gestionar los tiempos de asignación de tareas y controlar la frecuencia de las CPUs. El esquema de planificación final que se propone, se basa en un control en cascada que se beneficia de una ley de control PID para calcular la frecuencia y un controlador de modos deslizantes para lidiar con las perturbaciones y las variaciones paramétricas del sistema.

Abstract

Real-Time schedulers (RTS) is one of the paramount research topics in operating systems. This kind of schedulers are focused on allocating tasks to CPUs to honour task temporal constraints. The omnipresence of RTS ranges from critical hard-real time system, such as medical devices or automatic controllers to entertaining applications, such as video-games. Moreover, modern devices, such as cellphones, medical devices and other portable devices introduce new constraints such as thermal upper bounds or consumed energy. For instance, the energy consumed by a hearing aid must be as low as possible, and the temperature of the device must be comfortable for patients. In the automotive and aerospace industries, thermal control is also paramount to avoid shortening the lifespan of the components, and power draw must be minimized when energy is scarce as in satellite systems powered by solar panels.

The problem herein addressed focuses on designing Hard Real-Time schedulers for Multiprocessor Systems on Chip (MPSoCs), also considering thermal and energy constraints. Since this problem encompasses discrete and continuous issues, the approach taken is based on Timed Continuous Petri Nets (TCPN), a formalism capable of capturing the behavior of positive continuous signals, such as temperature or energy, and simulating discrete behaviours, such as task executions. A conjecture that motivated this research was that combining combinatorial and automatic control techniques it is possible to derive better schedulers, where a continuous controller is used to work around the NP-Completeness of the problem and to reject some disturbances, such as CPUs detentions, arrival of aperiodic tasks or modeling uncertainties, and the combinatorial analysis is used to discretize the continuous schedules. Now, we can confirm that the conjecture is true.

Ch. 1 provides some background, sets the Thesis rational and objectives, and formalizes the problem herein addressed.

Ch. 2 introduces a novel modeling methodology that represents in a monolithic model the task arrival, task allocation, heat generation, energy consumption and frequency selection using a single formal tool. This methodology offers great advantages over previously reported techniques. For example, the model is as precise as a finite element model. It is represented in a state variable equation, amenable for control and scheduling purposes, and the calibration

stages of other approaches are avoided. A key point of this modeling methodology lays in the fact that the partial differential equation representing the system's behavior is translated into a set of first order ordinary differential equations, reducing considerably the complexity of the analysis.

A software tool to simulate and validate the model and proposed schedulers is introduced in this chapter.

Ch. 3 presents a first RT scheduler, which focuses on honoring the constraints of a HRT task set. With this purpose, the scheduler computes the required task execution times and represents it as a fluid execution job function. A sliding mode controller ensures that every tasks is executed as indicated by the function. Finally, the continuous schedule is discretized.

Ch. 4 extends the previous controller to include thermal aspects. Since the temporal and thermal aspects must be controlled and there exists only one control variable, the system is not controllable. To overcome this problem, a fluid schedule honoring both temporal and thermal aspects is computed using a Linear Programming Problem, then a sliding mode controller is used to force the system to track this new fluid schedule even in the presence of disturbances. The continuous schedule is discretized taking an approach which balances accuracy in thermal control and scheduling overhead.

Ch. 5 introduces a scheduler that is capable of minimizing energy and honouring temporal and thermal constraints of the tasks. Even more, it is capable of manage arriving of aperiodic tasks, and task allocation times and CPU frequency can be managed by the controller. The final scheduling scheme relies on a cascade control that leverages a simple PID control law to compute the frequency and the sliding mode controller to deal with disturbances and parametric variations by bringing the fluid error to zero.

Contents

List of Tables

List of Figures

Acronyms and Symbols

ASIC	application-specific integrated circuits
ATM	Aperiodic Task Manager
BF	Best Fit
Bfair	Boundary Fairness
CL	load capacitance
CMOS	Complementary Metal-Oxide-Semiconductor
CPU	Central Processing Unit
DP	Dynamic priority
DP-Fair	Deadline Partitioning Fairness
DSPs	demand-side platform
DVFS	Dynamic Voltage Frequency Scaling
ECU	Electronic Control Unit
EDF	Earlier Deadline First
EDZL	Earliest Deadline first until Zero Laxity
EKG	Earliest deadline first with task splitting and K processors in a Group
FJP	Fixed Job Priority
FPGA	Field-Programmable Gate Array
FPZL	Fixed Priority until Zero Laxity
FTP	Fixed task Priority
GCD	Greatest Common Divisor
gEDF	Global Earliest Deadline First
GPGPUs	general-purpose computing on graphics processing units
GUI	Graphical User Interface
HRT	Hard Real-Time
ILP	Integer Linear programming problem
ISAs	Instruction Set Architectures
LL	Least Laxity
LLF	Least-Laxity-First
LPP	linear programming problem
METARTS	Minimum Energy Thermal Aware Real Time Scheduler
MPSoC	Multiprocessor on a Chip
NP	Non-deterministic polynomial-time hard

ODEs	Ordinary Discrete Equations
OLDTFS	On-line Discretization of the Thermal-Aware Fluid Schedule
PCB	Printed Circuit Board
Pfair	Proportionate Fairness
PI	Proportional Integral
PID	Proportional Integral Derivative
PN	Petri Net
QPS	Quasi-partitioned Scheduling
RM	Rate Monotonic
RT	Real-Time
RTOS	Real-Time Operating System
RUN	Reduction to Uniprocessor
SAS	Synchronous Arrival Sequence
SI	Infinite server semantics
SMPS	symmetric multiprocessors
SRAM	static RAM
SRT	soft real-time
SWaP	Small Weight and Power
TCPN	Timed Continuous Petri Net
TFS	Temporal Fluid Schedule
UMA	Uniform Memory Access
WCET	Worse case execution time
WF	Worst-Fit

Chapter 1

Introduction

This chapter motivates the work presented in this dissertation, summarizes the objectives and contributions of this Thesis, examines the previous related work, provides the necessary background, and formally states the central problem to address.

1.1 Motivation

Multicore system-on-chips (MPSoCs) are being increasingly incorporated into embedded systems, even to support applications with hard real-time (HRT) applications in critical, conservative environments such as the automotive or aerospace industry. Flight management or on-board maintenance systems tend to expand, and MPSoCs override the strong limits imposed by single-cores on CPU time when the applications grow. Additionally, MPSoCs can significantly contribute to savings in space, weight and power (known as the SWaP factor). For example, current cars can boast more than 70 electronic control units *ECUs*) weighing more than the entire engine block itself, wires included [1, 2].

However, leveraging the computing power of MPSoCs is quite a challenge. First, RT task scheduling is more complicated on multiprocessors than on a single-core processor. Second, power consumption can be an issue in battery-powered systems, or in spatial systems that obtain the energy from solar panels, which limits the instant power draw. Last, inefficient thermal management can lead to unexpected failures or to a short chip lifespan, two critical aspects in avionics and satellites. Besides the higher cost to power increasingly demanding

electronic systems, the higher energy dissipation represents a serious design issue as such energy is transformed in heat which. If not effectively dissipated, heating may increase the probability of faults and shorten the overall lifetime. Also, handhelds and other embedded devices impose thermal constraints because of ergonomic reasons (typically 45^oC for plastic and 41^oC for aluminum enclosures [3]).

Power consumption and heating can be lowered by decreasing V or the frequency, or by disconnecting voltage domains in a chip, as long as correct values are produced. Nevertheless, data correctness in RT systems do not only depend on the the output values, but also on when they are produced. Thus, RT schedulers, which are in charge of guaranteeing that jobs successfully meet their deadlines, must deal with an interesting trade-off between time requirements associated to the workload execution and energy dissipation.

This work addresses the energy and thermal management problem in a RT system. The first step to deal with this problem is the ability to model the temperature of a MPSoC accurately and efficiently. The following steps involve the design of a thermal-aware multiprocessor scheduler able to ensure the correct execution of a hard RT task set, keeping the system below a maximum temperature, minimizing the energy and managing disturbances and soft RT aperiodic tasks. This Thesis claims that TCPNs are a suitable a modeling methodology for this purposes. First, because of the continuous nature of the thermal problem and its solid mathematical background, which include a state space representation. Second, because they can be built in a modular way, with a graphical representation.

1.2 Thesis objectives

1.2.1 General Objective

The main objective of this Thesis is to show that, by using an fluid scheduler combining with automatic control techniques, it is possible to deal with the NP-Completeness of the hard real time scheduling problem while temporal, thermal and energy consumption aspects are considered.

Specific Objectives

- To obtain a novel modeling methodology for an MPSoC, that represents the activity and allocation of a RT task set on a set of CPUs, while the thermal activity, energy consumption and disturbances are considered. The purpose is to eliminate the problems of calibration, generalization and accuracy presented by other methodologies and to summarize in a single monolithic model of state variables, the phenomena that influence the design of HRT schedulers.

- To design and test a thermal-aware RT scheduler using the aforementioned methodology. To endow the thermal-aware RT scheduler with the capability of managing aperiodic tasks minimizing power consumption, rejecting disturbances and eliminating the combinatorial and unidimensional of the existing schedulers.

1.3 Background on Petri Nets (PN)

This chapter presents the Petri net model, its main aspects and properties. A special focus on Timed Continuous Petri nets (TCPN) is paid since they are the formal tool that will be used to model temporal aspects of tasks and thermal aspects of CPU. An interested reader must refer to [4, 5] for a deeper insight into this field.

Definition 1.1 *A Petri net structure ($\mathcal{N}$) is a bipartite digraph formed by the four-tuple $\mathcal{N} = \langle P, T, Pre, Post \rangle$ where:*

- $P = \{p_1, p_2, \ldots, p_{|P|}\}$ *is a finite set of nodes named places,*
- $T = \{t_1, t_2, \ldots, t_{|T|}\}$ *is a finite set of nodes named transitions,*
- Pre *is a* $|P| \times |T|$*, where* $Pre[i, j]$ *represents the weighted (non-negative integer number) arc going from* p_i *to* t_j*,*
- $Post$ *is a* $|P| \times |T|$*, where* $Post[i, j]$ *represents the weighted (non-negative integer number) arc going from* t_j *to* p_i*,*

The Petri net structure fulfills that $P \cap T = \emptyset$ *and* $P \cup T \neq \emptyset$*.*

In a Petri net structure the incidence matrix is defined as $C = Post - Pre$.

Pictorially, places, transitions and arcs are represented by circles, rectangles, and arrows pointing from the source node to the ending one, respectively.

As a notation, if $x \in P \cup T$ is a node of the $\mathcal{N}$, then the input set of x, is ${}^\bullet x = \{x_i \in P \cup T |$ there exists an arc from x_i to $x\}$, and the output set of a node x, is $x^\bullet = \{x_i \in P \cup T |$ there exists an arc from x to $x_i\}$.

1.3.1 (Discrete) Petri nets

Definition 1.2 *A Petri net system (or Petri net) is the duple $PN = (\mathcal{N}, M_0)$ where $\mathcal{N}$ is a Petri net structure and $M : P \to \mathbb{N} \cup \{0\}$ is a marking function assigning to each place a non-negative integer number. M_0 is the initial marking.*

In a PN, a transition t_j is enabled at marking M_k iff $M_k(p_i) \geq Pre(p_i, t_j)$, $\forall p_i \in P$. In a PN, an enabled transition t_j, at M_k, can be fired. The firing of an enabled transition t_j produces a new marking that is computed with the fundamental PN equation:

$$M_{k+1} = M_k + Cv_k \tag{1.1}$$

where $v_k(j) = 1$ and $v_k(i) = 0$, $\forall i \neq j$

1.3.2 Continuous Petri nets

A continuous Petri net is a Petri net structure together with a marking, where transitions can be activated in any real amount between zero and its *enabling degree*. As a consequence, the number of tokens residing in a place is represented by a non-negative real number. This model is formally defined below.

Definition 1.3 *A* continuous *Petri net (CPN) is the tuple* $\langle \mathcal{N}, m_0 \rangle$ *where* $\mathcal{N}$ *is a Petri net structure and* m_0 *is the initial marking, where the marking* $m : P \to \mathbb{R}_+^{|P|}$ *is a vector representing the non-negative real number of tokens residing inside each place.*

In a CPN a transition $t_i \in T$ is *enabled iff* for every $p_j \in {}^\bullet t_i,\ m(p_j) > 0$. The *continuous enabling degree* of a transition t_i is given by

$$enab(t_i, m) = \min_{p_j \in {}^\bullet t_i} \left\{ \frac{m(p_j)}{Pre(p_j, t_i)} \right\} \tag{1.2}$$

The enabling degrees are well defined if every transition has at least one input place, which is assumed through this work. An enabled transition t_i at a marking m can be fired in any real amount $0 \leq \sigma_i \leq enab(t_i, m)$, leading to a new marking m' that is computed using the fundamental continuous Petri net equation:

$$m' = m + C\vec{\sigma} \tag{1.3}$$

where $\vec{\sigma} = \left[\sigma_1, \ldots, \sigma_{|T|}\right]^T$.

1.3.3 Timed Continuous Petri Nets (TCPNs)

Definition 1.4 *A* Timed Continuous Petri Net (TCPN) *(or* timed fluid net*) is a time-driven continuous-state system described by the tuple* $\langle \mathcal{N}, \lambda, m_0 \rangle$*, where* $\langle \mathcal{N}, m_0 \rangle$ *is a CPN and the vector* $\lambda \in \mathbb{R}_+^{|T|}$ *represents the transition firing rates.*

Under the *infinite server semantics*, an enabled transition t_i fires with a speed defined as the product of its associated rate, λ_i, and its enabling degree, $enab(t_i, m)$. The *flow vector* $\mathbf{f(m)}$ is defined such that $f_i(\mathbf{m})$ denotes the flow through t_i, thus,

$$f_i(m) = \lambda_i \cdot enab(t_i, m) = \lambda_i \cdot \min_{p_j \in {}^\bullet t_i} \left\{ \frac{m(p_j)}{Pre(p_j, t_i)} \right\}.$$

Given a TCPN $\langle \mathcal{N}, \lambda, m_0 \rangle$, the pair $\langle \mathcal{N}, \lambda \rangle$ is referred as the TCPN timed structure.

The enabling degree vector $\mathbf{enab}(m) \in \mathbb{R}_+^{|T|}$ is defined such that $enab_i(m) = enab(t_i, m)$.

A *configuration* is a set of arcs $\mathcal{C}_j = \{(p_i, t_k) | p_i \in {}^\bullet t_k\}$ where every $t_k \in T$ appears in only one pair. An upper bound for the number of configurations is $\prod_{t \in T} |{}^\bullet t| = |{}^\bullet t_1| \times \cdots \times |{}^\bullet t_{|T|}|$.

For each possible configuration $\mathcal{C}_j$, the configuration matrix Π_j of dimension $|T| \times |P|$ is defined as:

$$\begin{array}{l} \forall i \in \{1, .., |T|\}, \forall k \in \{1, .., |P|\} \\ \Pi_{j_{ik}} = \left\{ \begin{array}{ll} \frac{1}{Pre(p_k, t_i)}, & \text{if } (p_k, t_i) \in \mathcal{C}_j \\ 0, & \text{otherwise} \end{array} \right. \end{array}$$

Then, the flow through the transitions in a given configuration $\mathcal{C}_j$ can be written as the vector $\mathbf{f(\boldsymbol{m})} = \mathbf{\Lambda \Pi_j \boldsymbol{m}}$, where $\mathbf{\Lambda}$ is the diagonal matrix whose elements $\Lambda_{i,i}$ are the firing transition rates λ_i.

In the systems modeled by TCPNs, control actions are applied to transitions and they can be seen as decrements of activity speeds from the maximum allowed determined by transition flows $f(m)$. Thus, control actions in TCPNs are applied to transitions, decreasing their flow. Transitions where a control action can be applied are said to be *controllable*. The set of all controllable transitions is denoted by T_c, and the set of uncontrollable transitions is denoted by $T_{nc} = T \setminus T_c$.

The control vector $\mathbf{u} \in \mathbb{R}^{|T|}$ is defined such that u_i represents the control action over $t_i \in T$. The effective flow through a controlled transition is given by:

$$\begin{array}{l} w_i(m) = f_i(m) - u_i, \\ 0 \leq u_i \leq \lambda_i \cdot enab_i(m) \end{array} \tag{1.4}$$

where $w_i(m) \geq 0$.

Thus, the behaviour of a TCPN system, in any configuration $\mathcal{C}_i$ is described by the state equation:

$$\begin{array}{rcll} \dot{m} & = & C\Lambda\Pi_i m - C\mathbf{u}, & \\ & = & C\Lambda\Pi_i m - C_c \mathbf{u}_c & \text{with } \mathbf{0} \leq \mathbf{u} \leq \Lambda\Pi_i m \end{array} \tag{1.5}$$

where $C\Lambda\Pi_i$ is the *dynamic matrix* for $\mathcal{C}_i$; the *input matrix* $C_c = C(P, T_c)$ is the restriction of the incidence matrix C to the columns of controllable transitions; $\mathbf{u}_c = \mathbf{u}(T_c)$ is the restriction of $\mathbf{u}$ to the controllable transitions T_c.

1.3.4 TCPN model of a system: an example

Consider the manufacturing system in Fig. 1.1b. Raw materials arrive to input buffer I. A resource of type R moves the material to the machine M, after which the resource is released. M performs some operations over the material to obtain a final product. A resource of type R unloads the machine and moves the product to the output buffer O. Then, the resource is released.

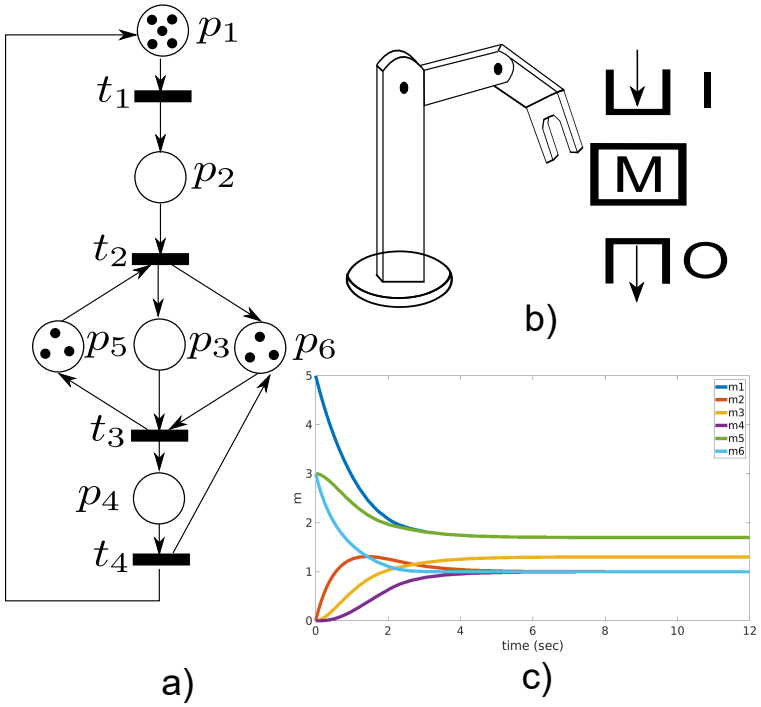


Figure 1.1: a) TCPN model. b) Manufacturing Process. c) Marking evolution.

Fig. 1.1a shows the TCPN model of the system. The quantity of raw material is modeled as the amount of tokens in place p_1, the resource is modeled by place p_6, and the capacity of the machine is the initial marking of the net. The resource allocation policy is modeled by the firing of transitions, and the number of products per unit time is the throughput of the net system. If the initial marking is set as $m_0 = [5\ 0\ 0\ 0\ 3\ 3]$ and the firing rate vector is $\lambda = [1\ 1\ 1\ 1]^T$, then $\Lambda = diag(\lambda)$. Therefore, $\boldsymbol{f(m) = \Lambda\Pi m}$ and from Eq. (1.5), we have:

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

Note that $enab(t_1, m_0) = m_0[p_1] = m_1$, $enab(t_2, m_0) = m_0[p_2] = m_2$, $enab(t_3, m_0) = m_0[p_3] = m_3$ and $enab(t_4, m_0) = m_0[p_4] = m_4$. In the initial state, the *configuration* in the intial state is (p_1, t_1), (p_2, t_2), (p_3, t_3) and (p_4, t_4). The marking evolution is depicted on Fig. 1.1c. As the system evolves, the marking m_3 increases and the marking m_6 decreases. At time $\tau = 2$, $m_2 = m_6$ and the *configuration* switches to (p_1, t_1), (p_2, t_2), (p_6, t_3) and (p_4, t_4). The marking evolution graph shows the two different dynamics provided by the two different system configurations.

1.4 Background on Real-time Systems

Typically, a real-time (RT) system consists of a controlling system and a controlled system (environment). The controlling system can be one or more Electronic Control Units (ECU), which consists on a number of application-specific integrated circuits (ASICs) and /or microprocessors, running tasks baremetal or on top of a RT operating system. The ECU interacts with the environment upon the information provided by sensors. Sensors will typically provide readings at periodic intervals, and the ECU must respond by sending signals to actuators. There may be unexpected or irregular events which must also receive a response. In all cases, there will be a time bound within which the response should be delivered. The ability of the computer to meet these demands depends on its capacity to perform the necessary computations in the given time.

Thus, the principal characteristic of a real-time (RT) system is to provide correct outputs in bounded time. The correctness of a RT system depends not only on the values but also on the time at which they are produced. Some examples of RT systems include process control systems, flight control systems, flexible manufacturing applications, robotics, intelligent highway systems, and high speed and multimedia communication systems. Belated outputs can be useless or even harmful to the system. [6, 7] or [8] provide broadly accepted, technically sound definitions related to the subject. This section summarizes the principal terms and concepts with the precise terminology and notation used in this dissertation.

1.4.1 RT Task Models, constraints and assumptions

A *task* is unit of work such as a program or code-block that when executed provides some service of an application. RT systems typically consist of sets of *RT tasks*. A *RT task* is a task that requires a specific amount of particular resource during a specific period of time. It can be either dependent or independent. For a dependent RT task, its execution may require an exclusive access to a shared resource (e.g., a file, a data structure in shared memory) or it has some precedence constraints. Tasks are said to be independent when they do not interact in any manner (accessing shared data, exchanging messages, etc.) with other tasks. If tasks have precedence constraints, then one task may need to wait until another task finishes its execution. In this work, each task is assumed to be independent where the only resource the tasks share is the processor platform. *RT tasks* are normally recurrent. Each instance of a task is called a *job*.

Each *RT task* occurring in a RT system has some timing properties. These timing properties should be considered when scheduling tasks on a RT system. The timing properties of a given task refer to the following:

- *Release time*: Time at which the task is ready for processing.
- *Worst case execution time (WCET)*: Maximum time taken to complete the task, after the task is released. The worst case execution time is also referred to as the worst case response.
- *Deadline*: Time by which execution of the task should be completed, after the task is released. time.

RT tasks can be classified as *hard*, *firm* or *soft* real-time according to its criticality level. Missing a deadline of a *hard* RT (HRT) task leads to unrecoverable or fatal results in the system. The results of a belated *Soft* RT (SRT) task can still be useful or recoverable to the system, and useless and harmless in the case of *firm* RT tasks.

A RT system can formally defined as follows:

Definition 1.5 *RT system is defined as a system $\mathcal{T}$, that consist of a set of RT tasks $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_n\}$, where the worst case execution time of each task $\tau_i \in \mathcal{T}$ is c_i. The system is said to be real-time if there exists at least one task $\tau_i \in \mathcal{T}$, which falls into one of the following categories:*

- *Task τ_i is a HRT task. That is, the execution of the task τ_i should be completed by a given deadline d_i, i.e., $c_i \leq d_i$.*
- *Task τ_i is a SRT task. That is, the later the task τ_i finishes its computation after a given deadline d_i, the more penalty it pays.*

- *Task τ_i is a* firm *RT task. That is, the earlier the task τ_i finishes its computation before a given deadline d_i, the more rewards it gains.*

This work mostly deals with HRT task sets. RT systems theory has lead to a number of theoretical task models. The most relevant and most related to the work in this dissertation are presented next.

The Liu and Layland (LL) Sporadic Task Model

The activations (jobs) of the *sporadic* tasks are separated by a *minimum*. Arrival time and deadline are relative to the task activation, without a global clock. This task model was formally defined in [9] by C.L. Liu and J. Layland; hence the name. Sporadic tasks are so common that they are customarily named *periodic* tasks, with the *period* considered as the minimum time between consecutive jobs [8]

The Three-Parameter Sporadic Tasks Model

This model is a generalization of the Liu and Layland's task model [9]. A sporadic task is defined as follows:

Definition 1.6 *A sporadic task τ_i is defined by the 3-tupla $\tau_i = (c_i, d_i, \omega_i)$, where c_i is the WCET, d_i the relative deadline and ω_i is the period or the minimum inter arrival separation time.*

A 3-parameter sporadic task system consists of a finite number of such 3-parameter sporadic tasks executing on a shared platform. The relationship between the values of task relative deadlines and periods determines a further classification of 3-parameter sporadic task systems:

- **Implicit-deadline** task system: the relative deadline of each task is equal to the task's period ($d_i = \omega_i$).
- **Constrained-deadline** task system: the relative deadline of each task is no larger than the task period ($d_i \leq \omega_i$).
- **Arbitrary-deadline** task systems: their relative deadlines are not subject to any constraint with regard to their periods.

Fig. (1.2) shows a *constrained-deadline* task system and Fig. 1.3 depicts a *implicit-deadline* task system.

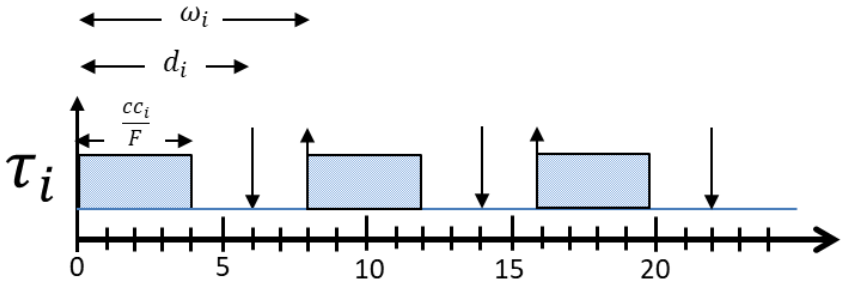


Figure 1.2: Sporadic task with constrained-deadline.

A sporadic task is said to generate a synchronous arrival sequence (SAS) of jobs (Fig. 1.3), if it generates jobs as rapidly as permitted to do so, i.e., a job arrives immediately the minimum inter arrival separation has elapsed since the arrival of the previous job. This notion is extended to task systems:

Definition 1.7 *A task system is said to generate a SAS of jobs if all tasks in the task system generate a job at the same instant in time, and each task generates subsequent jobs as rapidly as permitted.*

Periodic Tasks

A periodic task (Fig. 1.3), generates an unlimited sequence of jobs. Each job of a task arrives with a continuous and deterministic pattern of time interval named period. That is, it continuously requests resources at time values. In addition to this requirement, a RT periodic task must complete processing by a specified deadline relative to the time that it acquires the processor.

Definition 1.8 *Let τ_i be a periodic task defined by the 4-tupla $\tau_i = (a_i, c_i, d_i, \omega_i)$, where a_i represents the release time. Every job is generated at time $(a_i + k\omega_i)\ \forall\ k \in \mathbb{N}$. The computation time is denoted as c_i and constitutes the WCET of the task. The* deadline *d_i is the time by which the task should finish its execution. Finally, ω_i is the* period *of the task. The parameters (c_i, d_i, ω_i) are considered constant for every job of the task.*

This task model presents analytical difficulties in RT applications, and it has been shown that scheduling problems for periodic task models are highly intractable, NP-complete, even on uniprocessors [8].

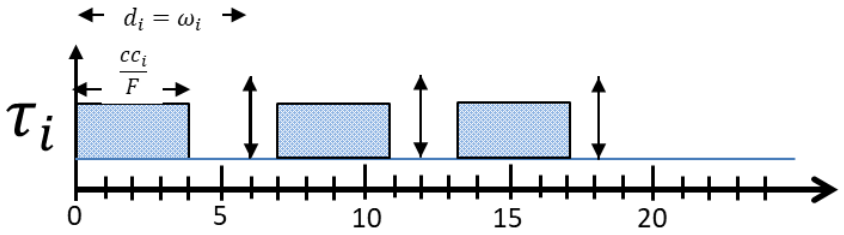


Figure 1.3: Sporadic SAS task with implicit deadline.

Aperiodic tasks

Aperiodic tasks also consists on a infinitive sequence of instances (Fig. 1.4). They differ with periodic tasks their irregular arrival time.

Definition 1.9 *Let τ_i^{ap} an aperiodic task defined by the tupla $\tau_i^{ap} = (c_i^a, d_i^a)$, where c_i^a is the WCET and d_i^a is the deadline. The arrival time of τ_i^{ap} is denoted as r_i^a and is an unknown parameter.*

System Utilization

The *utilization of a task* τ_i is defined as:

$$u_i = \frac{c_i}{\omega_i}. \tag{1.6}$$

The *system utilization* is the fraction of time the processor is busy running the task system, i.e., the sum of the utilization of all the tasks in the task set,

$$U = \sum_{i=1}^{n} \frac{c_i}{\omega_i}. \tag{1.7}$$

The utilization of a task system must be less or equal than the number of processors $U \leq m$. If this restriction is not met, then there does not exist an feasible schedule.

The *density* of a task τ_i is the ratio of the WCET to the smallest of its relative deadline or period,

$$D = \sum_{i=1}^{n} \frac{c_i}{\min(\omega_i, d_i)}. \tag{1.8}$$

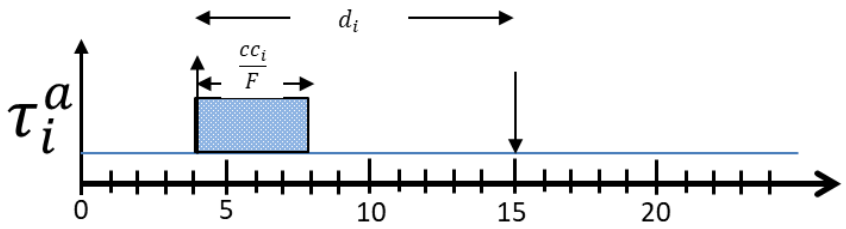


Figure 1.4: Aperiodic task

Without loss of generality, in this work the WCET c_i of task τ_i is characterized as a function of the processor frequency. The WCET is considered to be fully scalable with the processor frequency F:

$$c_i = \frac{cc_i}{F}$$

where cc_i is the WCET in cycles of the processor.

The hyper-period is defined as the period equal to the least common multiple of the n periodic tasks $H = lcm(\omega_1, \omega_2, \ldots, \omega_n)$.

Common assumptions

The following list summarizes common assumptions taken in RT multiprocessor scheduling, which this dissertation embraces [10, 8]:

- This work focuses on *implicit-deadline tasks*.
- Tasks are independent. The only resource shared by tasks are the CPUs. The contributions in this work mostly assume the three-parameters model (extended with aperiodic tasks when required) as presented in Sec. 1.4.1.
- Jobs do never relinquish the CPU voluntarily. They always remains in the ready state when forced to leave the CPU by preemption.
- Jobs can be preempted any time, without further restrictions than those imposed by the scheduling algorithm.
- Jobs must execute sequentially on at most one processor at any time
- WCET is never exceeded, and job activation do always meet the adopted task model.
- Context-switching, migration and scheduling overheads are either negligible or prorated in the WCETs.

- Speculative mechanisms or cache memories are absent or deactivated in the CPUs

1.4.2 Multiprocessor Systems

A multiprocessor system is composed of a number of processors. When the multiprocessor is integrated in a single chip, the CPUs are often called *cores*, giving rise to the terms *multicore*, *on-chip multicore* [11] and even *single-core* or *unicore* processors to denote a processor with a single CPU. In today's embedded systems, a *multiprocessor system on chip* (MPSoC) is a single chip that packages a number of general-purpose cores, specific-purpose functional units like DSPs, GPGPUs, cryptographic processors or FPGAs, and interconnections and I/O elements. This dissertation assumes symmetric multiprocessors (SMPs) with Uniform Memory Access (UMA): all CPUs share a common principal memory through a shared interconnection, and the memory access time does not depend on the CPU.

As far as RT scheduling is concerned, multiprocessor systems are described as follows [12]:

- Heterogeneous: The task execution rate depends on the task and the processor frequency and architecture. CPUs have different Instruction Set Architectures (ISAs). A specific task can only run on a specific CPU or set of CPUs.
- Homogeneous: The task execution rate is independent of the CPU. This implicitly assumes that all CPUs are identical, have the same ISA and run at the same frequency. Any task can run on any available CPU.
- Uniform: The task execution rate relies only on the CPU frequency. CPUs are identical but can run at different clock frequencies. Hence a processor of speed 2 will double the execution rate of all tasks with speed of 1.

An example of state-of-the-art multicore processor used in the industry include the MPPA-256 [13], which does not support context switching and multitasking, forcing the use of partitioned scheduling [14]. Another example is the LEON4 SPARC V8 Processor, manufactured by Cobham Gaisler and selected by the European Space Agency [15].

1.4.3 RT scheduling

RT scheduling manages the allocation of task sets to a processor or set of processors. Most basic concepts in RT scheduling belong to the broader subject of scheduling as known in computing and operations research, and appear in computer operating systems, manufacturing and service industries. However, the specificity of each field often implies distinct approaches and nuances. The following concepts are at the basis of this dissertation in the context of RT systems.

A *Scheduling algorithm* is the set of rules to solve the problem or allocating n tasks to m CPUs to satisfy (*scheduling problem*). With $n > m$, this implies sharing one or more processors among different tasks, which implicitly involves switching tasks on a processor at specific *scheduling events*. Common scheduling events are task activation, task termination, the expiration of a fixed or variable time interval (known as *quantum* or *time slice*, or the occurrence of more complex conditions such as time or thermal constraint violations. A scheduling algorithm is *work conserving* if it does never leave an idle processor as long as an unscheduled ready task exist [8].

The result of solving a scheduling problem by means of a scheduling algorithm is a *schedule*. Formally a *schedule* is defined as follows:

Definition 1.10 *A schedule is a set of 3-tuples* $(\tau_i^q, CPU_k, [\zeta_r, \zeta_s])$*, where* τ_i^q *is the* $q-th$ *job of task* τ_i *that is allocated to* CPU_k*, starting its execution at time* ζ_r *and completing or being preempted at* ζ_s*.*

A task set is said to be *feasible* if there exists some scheduling algorithm under which it is schedulable (i.e. tasks meet the imposed constraints). A scheduling algorithm is said to be *optimal* if it is able to schedule any task set that is feasible [16]. A schedule is said to be *feasible* if it can be repeated every hyper-period while meeting the required deadlines [17]. Feasible schedules can be computed off-line, before system operation, as a set of 3-tuples, or can be computed on-line, determining which jobs must be allocated to which processors.

The time interval $[\zeta_r, \zeta_s]$ is usually given as an integer multiple of a time quantity named *quantum*.

A *scheduler* is a functional entity of an operating system or baremetal routine that upon the occurrence of an scheduling event determines which jobs must run on which CPUs according to the rules of a scheduling algorithm. The *dispatcher* gives control of a CPU to the job allocated by the scheduler [18], causing the *preemption* of the job that was running in that CPU. In the context of RT systems context, *preemption* is the operation of suspending the execution of a job and inserting it into a ready-queue [19].

Complexity of the scheduling problem

The scheduling problem belongs to the broader class of combinatorial search problems. Combinatorial search is among the hardest common computational problems. These problems form the well-studied class of NP-hard problems, i.e. the problem is as difficult to solve as other problems tat have been widely studied and reported by experts [20]. It is believed that it is very unlikely that a NP-hard problem can be solved in polynomial time.

Hence, it is natural to think of an approximation solution method for the scheduling problem. Also, the problem formulation has been often simplified in the treatment of the

certain properties of the system, or by making a few assumptions as in the task models presented in Sec. 1.4.1.

1.4.4 RT Scheduling Algorithms

The aforementioned assumptions and the design space of RT scheduling introduce different classifications. The following summary presents the two most relevant ones in the context of RT scheduling for the purposes of this dissertation.

Priority management

According to how task and job priorities are managed, RT schedulers are classified as follows [21]:

- *Fixed task priority (FTP) scheduling.*- Tasks priorities are static. A well-known example is Rate Monotonic, where priorities are allocated according to task period (the shorter the period, the higher the priority) [17].
- *Fixed job priority (FJP) scheduling.*- Jobs of the same task may be assigned different static priorities: once assigned, the priority of that job never changes. An outstanding example is the Earlier Deadline First (EDF) policy [22].
- *Dynamic priority (DP) scheduling.*- Job priorities can dynamically change during the execution. Examples of RT schedulers leveraging this policy are *Pfair* [23], *DPfair* [24], *Least Laxity (LL)* [25] or *EDZL* [26].

Task allocation

The way RT tasks are distributed among the available CPUs in multiprocessors has been traditionally tackled through two different approaches: *partitioned* and *global* scheduling [27, 28] and mixed solutions [29].

- *Partitioned schedulers* allocate tasks statically to CPUs, and tasks are not allowed to migrate. Under this scheme, HRT schedulability analysis can be derived from uniprocessor scheduling, ensuring a maximum utilization bound of 50%, which severely hampers the SWaP compromises [30].
- *Global schedulers* can allocate tasks to any CPU and allow task migration. Global Earliest Deadline First (*gEDF*) [31] is a global preemptive scheduling algorithm where all ready tasks are enqueued in a single ready queue, and the m highest-priority tasks are executed on the m processors. Job priorities are inversely proportional to their

associated deadlines, with a smaller absolute deadline corresponding to a higher priority. This algorithm guarantees soft real-time (SRT) schedulability for implicit-deadline task sets, managing dynamic priorities at task level while fixing priorities at job level, but is not optimal under a HRT scheme [32, 33].

- A third approach mixes static and dynamic allocation. Thus, *clustered scheduling* statically allocate tasks to clusters of CPUs, but jobs can migrate within their cluster [29]. Alternatively, *semipartitioned scheduling* preforms a preliminary static allocation of some tasks over the whole set of available CPUs, allowing the rest of them to migrate [34].

Global scheduling can benefit from the concept of *fluid scheduling*, which consists in instantaneously sharing CPUs among all active jobs [35, 36, 37, 38]. Practical implementations approach this theoretical behavior by interleaving jobs, keeping a *fair* CPU share within time periods, and honoring time constraints in the case of RT tasks sets.

Upon this principles, global schedulers such as *Pfair* [39], PD [40] and PD^2 [35] leverage the idea of *proportionate fairness*. The time is discretized and the tasks can only be executed during an integer number of quanta Q. The time quanta are then fairly distributed between the tasks so that the difference (the *fluid error*) between the execution time of every task and the fluid (continuous) schedule is smaller than 1 quantum at any time.

Only this kind of schedulers has been proved HRT and SRT optimal for implicit deadline task sets, whereas partitioned scheduling is limited to a 50% utilization bound [30], and no optimal scheduler exists for constrained or arbitrary deadlines [41].

The downside of *Pfair* algorithms is that they incur in an unfeasible number of context switches and migrations, since scheduling actions are taken on a quantum basis. *Deadline partitioning* schedulers such as *BFair* (Boundary Fairness [38]) and *DPFair* [37, 24]) alleviate this overhead by limiting the scheduling points to the set of all task deadlines, i.e. scheduling actions are now at variable time intervals instead of at a fixed quantum. This overhead can still be reduced by relaxing the fairness requirement [42], but this means the loss of optimality in terms of CPU usage.

Laxity rules

The laxity of a job is the difference between its time and work remaining until its deadline i.e., the laxity of an instance of the task τ_i is defined as the deadline of the task d_i minus the sum of its remaining execution time and the current time,

$$l_i = d_i - (rem_i - \zeta) \tag{1.9}$$

where rem_i is the remaining execution time. The Laxity changes dynamically, and when it reaches zero it means that if the task does not begin to run at that moment it will lose its deadline.

Two common greedy algorithms for uniprocessor scheduling are Earliest Deadline Until Zero Laxity (EDZL) [43] and Least Laxity First (LLF) [44]. They are implicitly deadline partitioning algorithms that assign the highest priority to the job either with the earliest deadline or with least laxity, respectively.

1.5 Background on heat transfer

Temperature is a constraint of paramount importance in modern computing systems. In order to study how the temperature varies in a computer, it is needed to understand some basic notions of thermodynamics. A *thermodynamic analysis* focuses on the amount of *heat transfer* that exists in a system when it evolves from one steady state to another. Heat transfer is a spontaneous thermodynamic phenomenon caused by the flow of energy that appears among regions in thermal contact at different temperatures (i.e. with a *temperature gradient*). There is a thermal equilibrium (no thermal flow) when all the regions reach the same temperature. It is important to note that *heat* and *temperature* are different concepts. *Heat* entails an energy transfer produced by differences in temperature, while *temperature* is a measure of the heat. More precisely:

Definition 1.11 ***Heat*** *is defined as the total kinetic energy of all atoms or molecules in a substance.*

Definition 1.12 ***Temperature*** *is a measure of the average kinetic energy of individual atoms and molecules of a substance.*

Adding heat to a substance makes its atoms or molecules move faster, which increase its temperature. Removing heat produces the opposite effect.

Definition 1.13 *The* ***specific heat*** *(often denoted c) is the amount of energy required to raise by one degree Celsius the temperature per unit mass of a substance.*

There are two types of specific heat: the *specific heat at constant volume* (c_v) and the *specific heat at constant pressure* (c_p). A substance whose specific volume does not change with temperature or pressure is known as *incomprehensible substance*. In this substances $c_p \cong c_v \cong c$. Solids and liquids remain constant volumes that are regarded as incomprehensible substances.

Definition 1.14 *The* ***amount of heat transferred in a system*** *is denoted as Q. The* ***heat transfer rate*** *is the amount of heat transferred per unit time, measured in Jouls per second (J/s, i.e.* watts*, and is denoted as $\dot{Q}$.*

Definition 1.15 *The ratio of heat per unit area perpendicular to the direction of $\dot{Q}$ is called the* ***heat flow****, and the average heat flow is expressed as:*

$$\dot{q} = \frac{\dot{Q}}{A} \tag{1.10}$$

where A is the heat transfer area.

Heat transfer takes place according to the following different mechanisms governed by their corresponding physical laws.

1.5.1 Conduction

In *conduction*, heat transfer takes place due to temperature difference in a body or between bodies in thermal contact, without mixing of mass. The rate of heat transfer through conduction is governed by the *Fourier's law of heat conduction.*

Definition 1.16 ***Fourier's law of heat conduction****.*

Consider an element that has volume and a thickness Δx, with a cross-sectional area A, where the two opposite areas have temperatures T_1 and T_2. The rate of heat transfer is proportional to the difference of temperature $\Delta T = T_2 - T_1$, and inversely proportional to the thickness of that areas:

$$\dot{Q}_{cond} = kA\frac{T_1 - T_2}{\Delta x} = -kA\frac{\Delta T}{\Delta x} \tag{1.11}$$

where the proportionality constant k is a transport property known as the thermal conductivity coefficient *of the material.*

When $\Delta x \to 0$, the equation 1.16 is reduced to its differential form:

$$\dot{Q}_{cond} = -kA\frac{dT}{dx} \tag{1.12}$$

This equation is known as Fourier's law of heat conduction*, in which $\frac{dT}{dx}$ is the temperature gradient.*

The minus sign appears because the temperature decreases in the direction of heat transfer.

The *coefficient of thermal conductivity* $\mathbf{k}$ (W/mK) represents the capability of the material to transport heat from one part to another because of the temperature gradient. It is a material-specific property used to characterize the steady heat transport.

1.5.2 Convection

In *convection*, heat is transferred to a moving fluid at the surface over it flows by a combination of molecular diffusion and bulk flow. Convection involves conduction and fluid flow. The rate of convective heat transfer is governed by the *Newton's law of cooling*.

Definition 1.17 ***Newton's cooling law***. *The change rate of the temperature of an object is proportional to the difference between its own temperature and the ambient temperature (i.e. its surrounding temperature). Its defined as:*

$$\dot{Q}_{conv} = hA_s(T_s - T_\infty) \tag{1.13}$$

where h is the coefficient of convection in (W/m^2K); A_s is the heat transfer surface area; T_s is the surface temperature; T_∞ is the ambient temperature.

In thermodynamics and mechanics, the *heat transfer coefficient* is the proportionality constant between the heat flux and the thermodynamic driving force for the flow of heat. It is a parameter calculated experimentally and its value depends on factors that influence on the convection, such as the fluid properties and the nature of movement.

1.5.3 Radiation

Radiation heat transfer is the transport of energy due to the emission of electromagnetic waves or photons from a surface or volume. Radiation does not require a heat transfer medium, and can occur in a vacuum. Heat transfer by radiation is proportional to the fourth power of the absolute material temperature. Also, it depends on the properties of the material, represented by ϵ, and on its temperature T_s. Radiation is given by the *Stefan-Boltzman law*:

$$\dot{Q}_{emit} = \epsilon\sigma A_s T_s^4 \tag{1.14}$$

where $\sigma = 5.67 \times 10^{-8} W/m^2 \cdot K^4$ is the *Stefan-Boltzman constant* and $0 \leq \epsilon \leq 1$ is the *emissivity* of the material. An object with $\epsilon = 1$ is a *black object*.

1.5.4 Heat Generation

Heat generation is a volumetric phenomenon, since it occurs throughout the medium. The heat generation rate in a medium is usually specified per unit volume (W/m^3) and denoted as $\dot{e}_g$. The heat generation rate in a medium can vary with time and position.

Definition 1.18 *The total heat generation rate in a medium with volume V can be determined as:*

$$\dot{Q}_{gen} = \int_V \dot{e}_g dV \tag{1.15}$$

When the heat generation rate is uniform in a medium, the last equation is rewritten as:

$$\dot{Q}_{gen} = \dot{e}_g V \tag{1.16}$$

where $\dot{e}_g$ is the rate constant of heat generation per unit volume.

1.5.5 Power dissipation in CMOS

Power dissipation in CMOS circuits comes from two components. The *dynamic dissipation* is caused by charging and discharging load capacitance (CL) and by short-circuit currents. The *static dissipation* is due to leakage currents, and other currenta continuously drawn from the power supply.

The dominant component in dynamic power dissipation is the charging of CL. The average dynamic power dissipation P_{dyn} can be expressed as the average instantaneous energy dissipation:

$$P_{dyn} = \frac{V_{DD}}{T} \int_0^T i_{DD}(t) dt \tag{1.17}$$

where $i_{DD}(t)$ is the transient current drawn from the power supply and V_{DD} is the voltage supply. The integral is the total amount of charges delivered during the time interval T. During each transition, the load capacitance is charged to V_{DD} or discharged to ground, and the amount of charges delivered (Q) is therefore equal to $CL\,V_{DD}$. Assuming a nodal activity of α on the load and a clock frequency of F, the total number of transitions amounts to αTF [45]. Therefore, Eq. (1.17) simplifies to:

$$P_{dyn} = \frac{V_{DD}}{T} \alpha CLTFV_{DD} = C_{eff} V_{DD}^2 F \tag{1.18}$$

where $C_{eff} = \alpha CL$ is called *effective switching capacitance* as it is the part of total capacitance that contributes to power consumption.

The main component in the static power are leakage currents. Static power dissipation is always present when the device is energized. Up to the 65 nm process, the dynamic power was the dominant factor in the total power ($\sim$ 90%), because techniques and materials

allowed the leakage currents *sub-threshold*. However, especially from the $45nm$ process, the contribution of static power has been growing to become the most relevant factor [46].

Leakage comes from a number of currents, among which sub-threshold and gate leakage currents are the most important. Gate leakage is insensitive to temperature and can be removed by using high-k devices. Dual gate devices drastically reduce the leakage for the cost of higher dynamic power [47]. Leakage varies non-linearly with temperature, but knowledge of the average temperature Within regions of uniform design style is sufficient to accurately determine leakage power consumption using a simple model resulting in less than 1% error in leakage estimation [48].

The dissipated power in a CMOS circuit appears in the form of heat, which can damage the circuit if not properly removed. In summary, heat is generated from the silicon active surface due to active switching and leakage. All the energy consumed by the integrated circuit is first dissipated in the form of heat in the transistors and interconnects, and is eventually removed to the environment by heat transfer. Elevated temperatures can shorten interconnect and device lifetimes, and package reliability can be severely affected by local hot spots and higher temperature gradients. For all these reasons, in order to fully account for the thermal effects, it is important to model temperature and to leverage accurate and efficient thermal techniques to control temperature in the system [49].

1.5.6 Thermal Models and Temperature Control

Thermal models are usually build using numerical methods as finite differences, finite elements, finite volumes [50, 51] or by analogy using RC models [52, 53, 54, 55, 56]. All of these techniques lead to fine thermal models. Nevertheless, numerical methods do not provide a state space model, and the derived results cannot be used in further optimization or control stages. On the other hand, RC models require parameter calibration, which much relies on engineering expertise and empirical data.

1.6 Prior Work

This section highlights the novelty of the proposals and the relationship among the papers published as a consequence of the research in this dissertation, with respect to previous work in thermal and energy aware RT scheduling. It also explains the rationale behind the thesis, and states the problem to solve.

Thermal-aware scheduling has been largely studied in single core systems, scaling the processor frequency to reduce its power consumption and temperature. Dynamic Voltage and Frequency Scaling (DVFS) is a well-know technique used with this purpose [57] [58].

Chen et al. [59] study the temperature problem in both uniprocessors and homogeneous multiprocessor systems. They leverage an EDF-based algorithm that minimizes maximum temperature and energy. They also derive an approximation bound for the maximum temperature in the EDF algorithm. They stick to a partitioning (no migration) scheme. Schor et al. [60] perform a worst-case temperature analysis for RT tasks with non-deterministic workloads running on multiprocessors systems.

Feedback methods from control theory have been often used to cope with a dynamic environment for RT scheduling. The feedback control algorithm in [61] enforces both thermal and RT constraints but is restricted to single-core processors, as they do not consider inter-core thermal coupling in multicore processors. A general framework of dynamic thermal management for multicore processors is proposed in [62]. It basically consists in a hierarchical feedback control loop with PI controllers, but does not guarantee RT performance. The thermal problem is defined in [63] as a control theory problem with a state space representation, and proposes an optimum solution to the frequency assignment problem for thermal balancing in MPSoCs, but it does not consider RT constraints nor the scheduling problem. Other contributions based on control theory are limited to SRT systems, allowing for a certain percentage of missed deadlines [64, 65, 66].

Ahmed et al. [67] tackle the problem of thermal constrained scheduling of periodic tasks, but they assume partitioned scheduling instead of global (migration) scheduling. Chantem et al. [54] use an equivalent circuit model to estimate the temperature for a given set of a hard RT tasks on a multicore system, also referred to a partitioned scheme, but they lack of a true integrated state model.

We introduced in Sec. 1.4.4 as a suitable *deadline partitioning* algorithm for maximizing CPU utilization while reducing the high number of context switches and migrations typical of quantum-based *Pfair* algorithms. However, the deadline intervals defined in *DPfair* can be too long to cope with temperature variations. On the contrary, making scheduling decisions every fixed quantum as in *Pfair* can provide a very precise temperature control, at the cost of a higher overhead nonetheless.

RUN (*Reduction to Uniprocessor*) [68, 69] addresses the problem of reducing the context switching and migration that appears in global scheduling. RUN considers feasible systems composed of independent implicit-deadline periodic (not sporadic) tasks on identical processors, and transforms the multiprocessor RT scheduling problem into an equivalent set of uniprocessor problem. The key underlying tool is the utilization of a task (Sec. 1.4.1) and its *dual*, e.g. if 0.6 is the utilization of a task, its dual is 0.4. During an off-line stage, RUN tries to find *proper utilization subset*, which are task groups (named *servers*) with an aggregated utilization equal to 1 (i.e. with maximum CPU utilization). This is called a *pack* operation. A *proper utilization subset* can be allocated to a virtual processor, configuring a *proper subsystem*. After a succesful *pack*, a *dual* operation follows. The *dual* operation

mas groups with an aggregated dual utilization of 1groups considering the *dual* utilization. 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 (*unpacking*) the *pack-dual* operations performed off-line . Under the DUAL and PACK operations, this algorithm yields a small number of preemptions and migrations.

QPS (*Quasi-Partitioned Scheduling*) [70] does also partition the system tasks into subsets. There are two types of subsets, the *minor* and *major* execution sets, depending on whether they require one or multiple processors. If all subsets are minor, QPS boils down to a partitioned EDF. Major execution sets are scheduled either by a set of QPS servers on multiple processors, or by local EDF on a single processor depending on their execution requirements.

QPS is capable of adapting its scheduling strategy as a function of system load by monitoring major execution sets at 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 provide online adaptation.

Despite the optimality of P*fair* and DP*fair*, most of the aforementioned references leverage partitioned approaches. The overhead due to context-switch and migration has tipped the scale in favor of semipartitioned and empirical designs, and few contributions still leverage global scheduling techniques [71]. In semipartitioned schedulers, some tasks are statically allocated to the processors whereas others are split across multiple processors as in global scheduling [72]. On the other hand, empirical designs resort to a combination of techniques, but can only solve specific problems [73].

Far from all the research discussed to this point, the automotive, aeronautics and space industry remains understandably conservative. Although they are already embracing multicores, they follow a traditional cyclic executive upon static partitioning [74, 75, 76]. This is a safe approach, but it leads to unbalanced SWaP, and faces tough problems such as the fair Worst Case Delay calculation, which calls for complex ILP solutions [77, 78]. Difficulties exacerbate when including additional constraints such as a maximum temperature or resource sharing, requiring new models and tools [73].

1.7 Rationale and contributions of this Thesis

The analysis of the prior work shows that there are still many open avenues in the HRT multiprocessor scheduling arena. There does not exist a single or simple approach, particularly when additional constraints other than time enter the scenario.

This dissertation constitutes the first contribution exploring TCPNs as a possible tool

for the design, analysis and simulation of thermal-aware HRT multiprocessor schedulers, and extend on the faced problems and their solutions.

1.7.1 TCPNs as a modeling tool

The first problem is to show that TCPNs are a suitable formalism to describe a multiprocessor executing a RT task sets, including task allocation and thermal activity. Ch. 2 presents a novel methodology that, unlike the traditional methods of thermal analysis cited in 1.5.6, leads to a single global TCPN model that captures the state of the RT task set and the CPUs along with the thermal behavior of the system. The TCPN global model encompasses a task module, a CPU module and a thermal module. The thermal module leverages the results published in [79].

A simulation framework provides the necessary tools to automatically build the model and perform simulations comparing different RT schedulers and providing temperature variations. This framework is described in 2.7, and constitutes the basis of a more ambitious simulation project, publicly available, that evolves as a joint effort [80].

1.7.2 TCPNs as a tool for HRT multiprocessor scheduling

The second problem is to explore and select a HRT scheduling algorithm for thermal control, also suitable for leveraging control techniques. Among the different choices previously referenced, global scheduling is a suitable starting point. The continuous nature of TCPNs matches well the continuous nature of global fluid scheduling and the thermal dynamics of the underlying system. Besides, only global fluid schedulers have been proved HRT optimal for implicit-deadline HRT task sets (Sec. 1.4.4).

We discussed in the previous section (Sec. 1.6 that the many scheduling points in quantum-based schedulers such as *Pfair* are good candidates to control temperature, at the cost of a high overhead. The fewer scheduling points in interval-based schedulers such as *DPfair* can lead to a lower overhead, but the variable intervals between deadlines can be too long to cope with temperature variations. This suggested the idea of exploring a compromise between quantum-based and deadline interval-based global scheduling to achieve an optimal thermal control in a MPSoC, resorting to feedback control techniques.

The result of this exploration is presented in Ch. 3, and was published in in [81]. The compromise in this algorithm consists in calculating the fluid function (the fluid execution rate) of each task for every deadline interval up to the hyperperiod, according to a deadline partition policy, but making the scheduling decisions on a quantum basis. The quantum is calculated as the great common divisor (GCD) of all deadlines. Every quantum, the simulation every quantum of a TCPN model of the system provides the actual execution

time of the jobs, which is compared with the expected (fluid) execution time to obtain the fluid error. The error determines the priority of each job according to the time it must run until the next quantum. Next, jobs are allocated to the CPUs by priority order. If the actual execution is subject to disturbances, a sliding mode feedback controller minimizes the fluid error. The evolution of the TCPN barely takes a simple linear computation.

1.7.3 TCPNs and control for thermal-aware HRT multiprocessor scheduling

The third problem, the inclusion of a thermal constraint in the research, led to the scheme presented in Ch. 4. This approach constitutes a novelty with respect previous work on global and thermal HRT scheduling, and has been accepted for publication in [82].

Now, the solution of a linear programming problem (LPP), which is computed off-line in polynomial time, provides the fluid time share that each task must be granted at each processor during the hyperperiod to meet HRT and thermal constraints. If the LPP has a feasible solution for a given HRT task set, then there exists a thermal-aware HRT schedule under which the task set is schedulable, and the schedule is optimal in terms of thermal and temporal constraints and CPU utilization.

The LPP does not provide CPU allocation nor execution times, since this is a global scheduling approach, which is inherently dynamic. Thus, we look for (and the solution of the LPP is) just a set of coefficients ${}_j\beta_i$, which represent the optimal amount of task execution time that must be allocated to each processor. Using these coefficients ${}_j\beta_i$, the functions ${}_jFSC_{\tau_i}(\zeta) = \frac{{}_j\beta_i cc_i}{H}\zeta$ are computed, each one representing the amount of τ_i that must be executed in CPU_j by time ζ so that temporal and thermal constraints are fulfilled.

Next, an on-line thermal-aware RT fluid scheduler allocates tasks to CPUs ensuring that task τ_i is executed in CPU_j exactly ${}_jFSC_{\tau_i}(\zeta)$ at time ζ. The functions are calculated at each deadline interval over the ordered set of all task deadlines (i.e. at each $\zeta = sd_i$) following a deadline partitioning scheme. However, the fluid scheduler is discretized on-line, on a quantum basis, with $Q = GCD(sd_i)$, as in *pfair* global algorithms.

At each quantum, the sliding mode feedback controller leveraged in Ch. 3 makes the TCPN model of the system to evolve so that the error between the ${}_jFSC_{\tau_i}(\zeta)$ and the actual fluid execution time of τ_i on each CPU_j becomes minimum, accounting for disturbances. Then, a per-CPU task priority queue is build upon the difference between the fluid and the actual discrete execution time, and jobs are accordingly dispatched. This is different and more convenient than the global job queue in former implementations of *Pfair* algorithms, including [81]. The feedback controller allows the system to recover from disturbances such as CPU detentions due to environmental hazards causing energy interruptions or thermal peaks.

The advantage and principal novelty of this approach is the implementation of an on-off control law, which leads to a low-overhead scheduler, capable of handling perturbations in underloaded systems without rescheduling a job. The adopted scheme can be used to simulate a system or to generate a scheduler for a real physical system (Sec. 4.3.4).

1.7.4 TCPNs and control for thermal-aware HRT scheduling with aperiodic tasks and disturbances

The last problem is the management of SRT aperiodic tasks while preserving the thermal and time constraints of the HRT task set. With this purpose, the proposal in Ch. 5 integrates a PID controller into the scheduling scheme presented in Ch. 4, configuring a cascade control.

The underlying idea is inspired on [83], with substantial differences nonetheless. As in [83], the starting point is the calculation of a minimum system frequency (F^*) at which the system can properly run a HRT task set, achieving an utilization of 100 %, and the maximum frequency (F^+) at which the system can run without surpassing a temperature threshold. Any frequency increase over F^* up to F^+ (i.e. any $F|F^* \leq F \leq F^+$) will lower the CPU utilization, *freeing* a utilization slack in which an aperiodic task could be run unless it does not fit into the maximum achievable slack, which is limited by F^+.

In [83], the LPP solved during the off-line stage yields discretized fluid functions, in CPU cycles per task and deadline interval (over the set of all task deadlines, according to a deadline partitioning approach), instead of providing a fluid time share per task and CPU that must be discretized later on-line, as in Ch. 4. The on-line stage is event-based (upon a zero-laxity, task completion or aperiodic task arrival event) instead of quantum-based, making the calculations lighter than in Ch. 4 during normal operation. However, upon arrival of an aperiodic task, an adaptive scheduler performs the on-line re-computation of the CPU cycles that each task has to run during each deadline interval, also accounting for the cycles of the incoming aperiodic task (when accepted). Also, task allocation is performed in [83] by applying a Fixed-Priority Zero-Laxity algorithm (FPZL, [84]) over a single global job queue, whereas the scheduler in Ch. 4 allocates jobs leveraging per-CPU queues, because the off-line stage already provides per-task, per-CPU fluid functions.

The contribution of Ch.5 is a lighter management of aperiodic tasks that can be incorporated to different scheduling schemes, while retaining the ability of the scheduler presented in Ch. 4 to deal with disturbances, thanks to a cascade control. Upon the idea of freeing utilization by increasing the frequency over the minimum F^* that warrants the correct execution of the HRT task set, a PID controller avoids the heavy computation required to find the upper next safe frequency in the CPUs to run a SRT aperiodic task, accepted or not by an Aperiodic Task Manager, depending on the remaining capacity of the system.

1.8 Problem definition

Formally, the problem that constitutes the starting point of the Thesis can be formally stated as follows.

Definition 1.19 *The system consists of a set of n periodic tasks* $\mathcal{T} = \{\tau_1, \ldots, \tau_n\}$ *that must be allocated and executed on a set of m homogeneous processors* $\mathcal{P} = \{CPU_1, \ldots CPU_m\}$*. A task* τ_i *is characterized by a 4-tuple* $\tau_i = (cc_i, d_i, \omega_i, e_i)$*, where* cc_i *are the CPU cycles required to complete an instance of the task or* job*, calculated as the* $WCET$*;* d_i *is the relative implicit deadline,* ω_i *is the task period and* e_i *is the energy demanded by the task during execution. All tasks hold the same priority and are independent (with no resource sharing). A job* τ_i^k *of task* $\tau_i \in \mathcal{T}$ *must complete its execution within the time period* $[(k-1)\omega_i, (k-1)\omega_i + d_i]$*, i.e. it is executed for a fixed number of* cc_i *CPU cycles. Once a job* τ_i^k *is over, a new one* (τ_i^{k+1}) *becomes immediately ready for execution at time* $k\omega_i$*.*

The hyper-period is defined as the period equal to the least common multiple $H = lcm(\omega_1, \omega_2, \ldots, \omega_n)$ of all task periods. The system utilization is defined as the fraction of time during which the processor is busy running the task set i.e., $U = \sum_{i=1}^{n} \frac{cc_i}{\omega_i}$. In this dissertation we only consider cases where $U \leq m$.

Problem 1.8.1 *Thermal-aware fluid scheduler. Given the system defined in Definition 1.19, design an algorithm to allocate within the hyperperiod the tasks in* $\mathcal{T}$ *to the m identical CPUs in* $\mathcal{P}$ *in such a way that the deadlines for* $\mathcal{T}$ *are always satisfied and the CPU temperatures are kept always below a given temperature threshold* $\boldsymbol{T_{max}}$*.*

Chapter 2

System Modeling Methodology

This Chapter presents a modular methodology that models RT tasks, CPUs and their thermal behavior in an MPSoC. The methodology captures task arrival time, the allocation of tasks to CPUs, CPU execution cycles, energy consumed and temperature variations. The Chapter introduces a simulation tool, publicly available, developed to support the experiments in this Thesis.

2.1 Introduction

The first problem to face in order to explore the capabilities of TCPNs for thermal-aware multiprocessor RT scheduling is to build a model encompassing a RT task set, the CPUs and the thermodynamics of the system. Tasks and CPUs can be separately modeled using two distinct types of TCPN modules. A third different module divides the MPSoC into prisms, and models heat generation, thermal conduction and convection. Heat generation considers both the dynamic and the static power. All modules are assembled into a global TCPN model that represents the dynamical behavior of the system: task arrivals, CPU throughput and temperature variation.

This model has a number of benefits. First, it inherits the advantages of Petri nets, such as the graphical representation and a sound mathematical background. Second, the underlaying Ordinary Discrete Equations (ODEs) correspond to a central discretization in space while the time is kept continuous. Therefore, the model shares advantages of the

Table 2.1: System notation

Symbol	Description
n	Number of tasks
m	Number of processors
$\mathcal{T}$	A task set
$\mathcal{P}$	a Processor set
τ_i	The i^{th} task
cc_i	The worst-case execution in CPU cycles of τ_i
c_i	The worst-case execution time of τ_i
d_i	The relative deadline of τ_i
ω_i	The period of τ_i
u_i	The utilization of task τ_i
U	System utilization
H	Hyperperiod
$\mathcal{F}$	Set of discrete frequencies
$\mathcal{F}^s$	Set of discrete operating frequencies $\mathcal{F}^s \subseteq \mathcal{F}$
$\Phi*$	The normalized minimum frequency
$F*$	The minimum frequency
F_c	The solution of Eq.(5.4)
F^+	The maximum thermal frequency
F_n	The operating frequency
sd_i^k	The $k-th$ deadline of task τ_i
SD	The set of ordered deadlines sd_i^q
I_{SD}^k	The $k-th$ scheduling interval
x_i^k	The cycles of task τ_i to be executed during I_{SD}^k
$x_{\tau_i^a}^k$	The cycles of task τ_i^a to be executed during I_{SD}^k
X^k	Set of all active tasks in I_{SD}^k
k_q	Thermal conductivity of component q
V_q	Volume of component q
V_{CPU_j}	Volume of CPU j
ρ_q	Density of component q
cp_q	Specific heat capacity of component q
h	convection coefficient
T_q	Temperature of component q

Table 2.2: Notation for the TCPN model

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

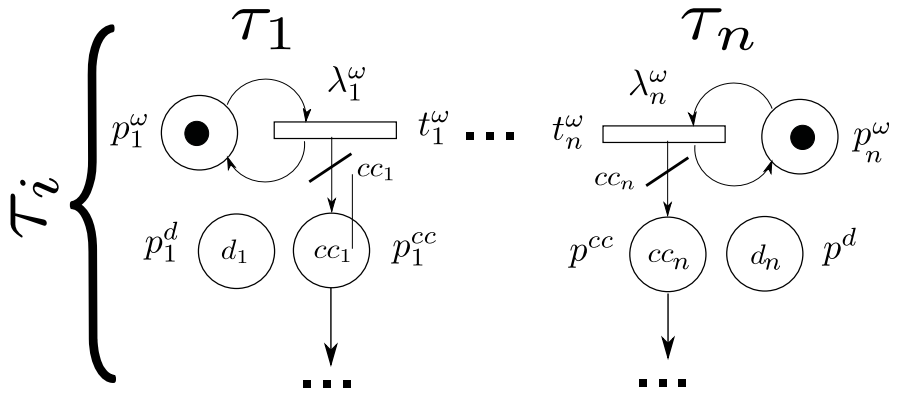


Figure 2.1: TCPN module for task τ_i. Firing of transition t_i^ω represents job generations. A job generated means cc_i cycles in place p_i^{cc}. The relative deadline is captured in p_i^d for model completeness.

finite difference methods like consistence, order and stability while avoiding the calibration stage. Last, this approach leads to a state space equation that encompasses task arrival, task execution and thermal evolution. This makes easier to perform the thermal and temporal analysis of CPUs, and it allows controlling the temperature of the system by selecting an appropriate task execution sequence. This constitutes a new approach that leverages the power of TCPNs to integrate a modular description of a RT physical system along with its thermal and energy characteristics.

The methodology can be applied to a wide range of tasks and scheduling models. However, this work abides by the assumptions described in Sec. 1.4.1, implicit in the formal definition of the general problem tackled in this Thesis (Sec. 1.19).

Secs. 2.2, 2.3 and 2.4 describe the task, CPU and thermal modules respectively. Sec. 4.3.4. Table 2.1 summarizes system parameters and symbols used throughout this dissertation, and Table 2.2 gathers the notation related to the global TCPN model.

2.2 Task module

Modeling tasks arrival

The period ω_i of task τ_i implies that $\frac{1}{\omega_i}$ jobs arrive per second in average (*arriving frequency*). This is captured in the TCPN module as the firing rate $\lambda_i^\omega = \frac{1}{\omega_i}$ of transition t_i^ω (Fig. 2.1).

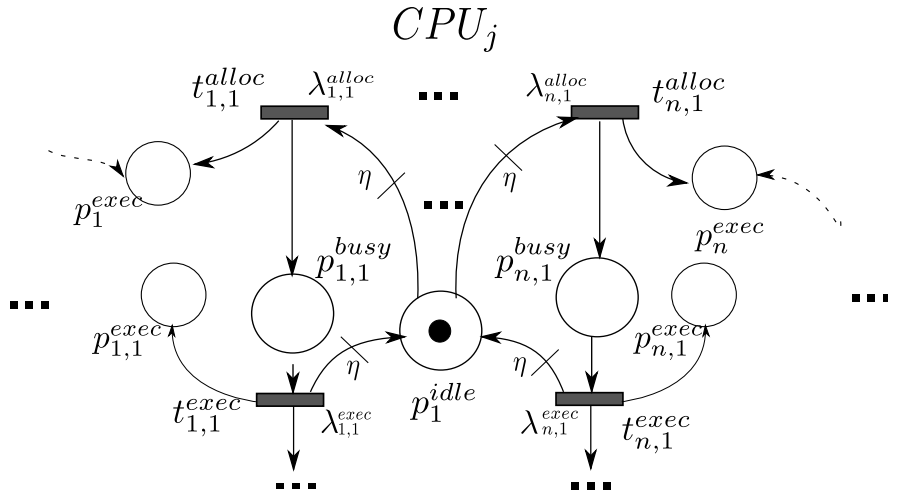


Figure 2.2: TCPN module for a single CPU_j. Places p_j^{busy} represents the busy state of processor. The firing transition t_j^{exec} represents the execution of a job. Place p_j^{idle} represents the idle state.

Modeling task deadlines

The relative deadline d_i of task τ_i is represented in the model of by the marking d_i at place p_i^d (Fig. 2.1). Since this work only considers implicit deadlines (Secs. 1.4.1 and 1.19), $d_i = \omega_i$. Gathering all the system information into the TCPN requires the addition of the marking of an isolated TCPN place, although is never used.

Modeling task execution time

The duration of a task is represented in the TCPN model by the arc going from transition t_i^{ω} (Fig. 2.1). This arc models job arrival. Its weight (cc_i) is the WCET of the corresponding task in CPU cycles. Accordingly, the marking of place p_i^{cc} stands for the CPU cycles that remain to be executed.

2.3 CPU module

This modeling methodology can be applied to homogeneous and heterogeneous cores, and for global, partitioned and semi partitioned scheduling. This work considers homogeneous cores and global scheduling. Therefore, any CPU in the set $\mathcal{P} = \{CPU_1, \ldots, CPU_m\}$ can accommodate any task in the set $\mathcal{T}$.

Modeling CPU throughput

The TCPN module of a CPU CPU_j consists of transitions $t_{1,j}^{alloc}$, ... , $t_{n,j}^{alloc}$, transitions $t_{1,j}^{exec}$, ... , $t_{n,j}^{exec}$, places $p_{1,j}^{busy}$, ... , $p_{n,j}^{busy}$ and place p_j^{idle} (Fig. 2.2). Transitions $t_{1,j}^{alloc}$, ... ,$t_{n,j}^{alloc}$ model the allocation of the jobs of the tasks τ_1, ... ,τ_n to processor CPU_j. They are considered *immediate* transitions (transitions that fires in zero time once they are enabled), and named *allocation transitions* throughout this dissertation. Transition rates $\lambda_{1,j}^{alloc}$, ... , $\lambda_{n,j}^{alloc}$ match the allocation rate of jobs to CPU_j. Places $p_{1,j}^{busy}$, ... , $p_{n,j}^{busy}$ represent the busy state of the processor, when a job of task τ_i is allocated to CPU_j. Transitions $t_{1,j}^{exec}$, ... , $t_{n,j}^{exec}$ represent the execution of the corresponding jobs on CPU_j. Transition rates $\lambda_{1,j}^{exec}$, ... , $\lambda_{n,j}^{exec}$ model the execution rate. The model considers that jobs start to be executed instantaneously after the allocation [1] . The marking of place p_j^{idle} models the available cycles of CPU_j (*throughput capacity*). The initial marking at p_j^{idle} is set to 1, to express that the CPU_j is idle. The arcs going from transitions $t_{1,j}^{exec}$, ... , $t_{n,j}^{exec}$ to place p_j^{idle} and from place p_j^{idle} to transitions $t_{1,j}^{alloc}$, ... , $t_{n,j}^{alloc}$ are weighted by a constant value η, to ensure that the flow in transitions $t_{i,j}^{alloc}$ is limited by the throughput capacity of the CPU modeled by place p_j^{idle}.

Modeling CPU Power and Energy

The power consumed by a CPU_j has two components: the dynamic power due to computational activities of tasks P_{dyn}, and the static power. The latter is mostly due to leakage currents. Although it depends non-linearly of temperature, it can be modeled as a linear function (Sec. 1.5.5). Thus,

$$P_{CPU_j} = P_{dyn_j} + P_{leak_j} \tag{2.1}$$

where P_{dyn_j} is the dynamic power due the activity in CPU_j. The summation, $P_{leak_j} = \delta T_j + \rho$, is an approximation of leakage power consumption, where T_j is the $j-th$ processor's temperature and δ, ρ are modeling constants [85].

The average energy consumed in one cycle by task τ_i running on CPU_j at a frequency F_j can be defined as $e_i = P_{CPU_j}/F_j$. In Fig. 2.6 (module CPU_j) the execution flow $f_{i,j}^{exec}$ of transitions $t_{i,j}^{exec}$ stands for the number of CPU cycles of task τ_i executed on CPU_j, then the dynamic power consumed by CPU_j when task τ_i is executed during its cc_i cycles can be expressed as:

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

$$P_{dyn_j} = \sum_{\tau_i \text{executed in } CPU_j} f_{i,j}^{exec} \frac{e_i}{cc_i} \tag{2.2}$$

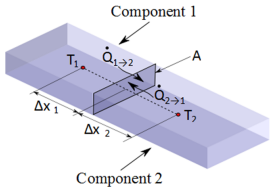
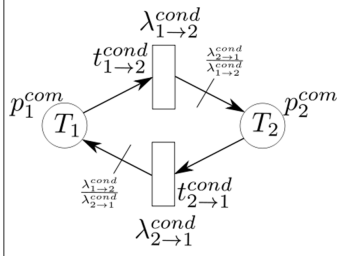
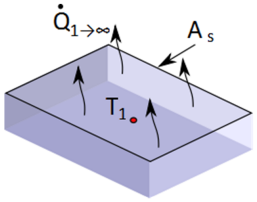
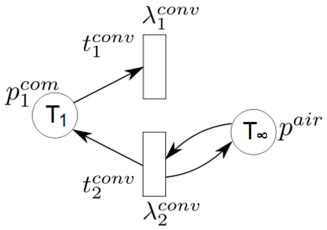
	Components Subject to Heat Transfer Mechanisms	TCPN module of each Heat Transfer Mechanisms	TCPN module parameters
Thermal Conduction			$\boldsymbol{\lambda^{cond}} = \begin{bmatrix} \boldsymbol{\lambda^{cond}_{1\to2}} \\ \boldsymbol{\lambda^{cond}_{2\to1}} \end{bmatrix} = \begin{bmatrix} \frac{\mathbf{1}}{\boldsymbol{V_1\rho_1cp_1}} \cdot \frac{\boldsymbol{k_1k_2A}}{\boldsymbol{k_2\Delta x_1 + k_1\Delta x_2}} \\ \frac{\mathbf{1}}{\boldsymbol{V_2\rho_2cp_2}} \cdot \frac{\boldsymbol{k_1k_2A}}{\boldsymbol{k_2\Delta x_1 + k_1\Delta x_2}} \end{bmatrix}$ $Pre = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, $Post = \begin{bmatrix} 0 & \frac{\lambda^{cond}_{1\to2}}{\lambda^{cond}_{2\to1}} \\ \frac{\lambda^{cond}_{2\to1}}{\lambda^{cond}_{1\to2}} & 0 \end{bmatrix}$
Thermal Convection			$\boldsymbol{\lambda^{conv}} = \begin{bmatrix} \boldsymbol{\lambda^{conv}_1} \\ \boldsymbol{\lambda^{conv}_2} \end{bmatrix} = \begin{bmatrix} \frac{\boldsymbol{hA_s}}{\boldsymbol{V_1\rho_1cp_1}} \\ \frac{\boldsymbol{hA_s}}{\boldsymbol{V_1\rho_1cp_1}} \end{bmatrix}$ $Pre = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, $Post = \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix}$

Figure 2.3: Thermal conduction and convection mechanisms and their TCPN models.

2.4 Thermal module

This module embodies a thermal model that rewrites the thermal partial differential equation by a set of ordinary thermal differential equations. It is as precise as a Finite Element approach, with the advantage of yielding a state model. It also avoids the calibration stages of RC thermal approaches. This thermal model was accepted for publication in [86].

The module is composed of several thermal submodels, representing thermal conduction, convection and heat generation.

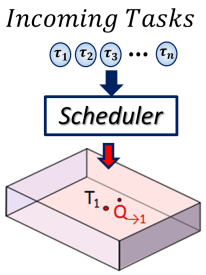
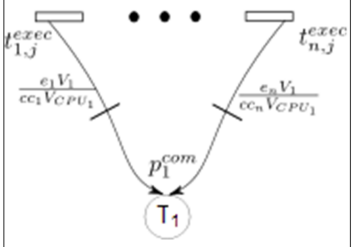
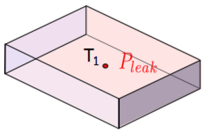
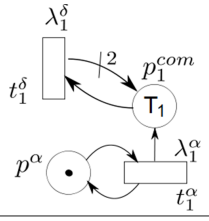
	Components Subject to Heat Transfer Mechanisms	TCPN module of each Heat Transfer Mechanisms	TCPN module parameters
Heat Generation due to dynamic Power			$\boldsymbol{\lambda}_{\mathcal{P}}^{\mathcal{T}} = \begin{bmatrix} \boldsymbol{\lambda}_{1,1}^{exec} \\ \vdots \\ \boldsymbol{\lambda}_{n,m}^{exec} \end{bmatrix}$ $Pre = [0 \quad \dots \quad 0]$, $Post = \begin{bmatrix} \frac{e_1 V_1}{cc_1 V_{CPU_1}} & \dots & \frac{e_n V_k}{cc_n V_{CPU_m}} \end{bmatrix}$
Leakage Power			$\boldsymbol{\lambda}^{leak} = \begin{bmatrix} \boldsymbol{\lambda}_1^{\delta} \\ \boldsymbol{\lambda}_1^{\alpha} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\delta} \\ \boldsymbol{\alpha} \end{bmatrix}$ $Pre = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, $Post = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}$

Figure 2.4: Heat generation mechanisms and their TCPN models.

Thermal Conduction and convection

The thermal conduction of two adjacent prismatic components is modeled according to the equation of the TCPN module together with the parameters shown in the row named *Thermal Conduction* of Fig. 2.3. These components are assumed to hold isotropic properties (thermal conductivity coefficients k_1, k_2; volumes V_1, V_2; densities ρ_1, ρ_2 and specific heat capacities cp_1, cp_2). The marking in the Petri net places (p_1^{com} and p_2^{com}) represents the average temperature of component 1 and component 2 respectively.

The thermal convection in a prismatic component (with a convection coefficient h) having a temperature T_1 is modeled by the TCPN module shown at the row *Thermal Convection* of Fig. 2.3. The marking in places p_1^{com} and p^{air} represents the average temperatures of the component and the ambient temperature, respectively.

Heat generation

A prismatic component increments its thermal energy due to tasks' jobs execution, this phenomenon is modeled by Eq.(2.2), and the equivalent TCPN module shown at the first

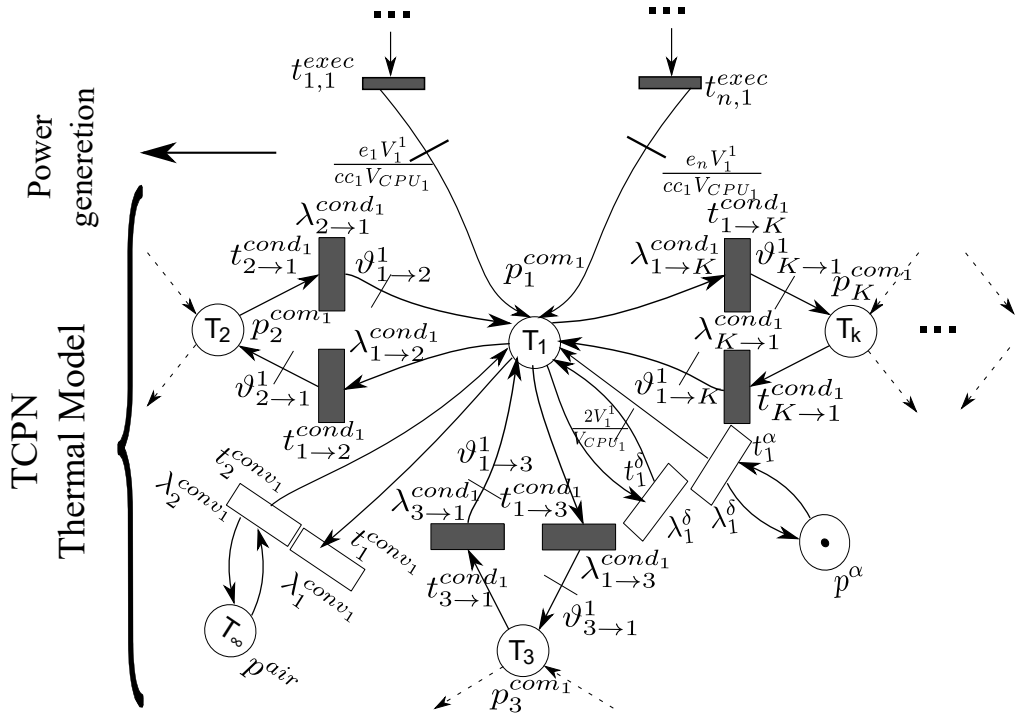


Figure 2.5: TCPN of the thermal model module. Weights $\vartheta_{q\to q+1}^j = \frac{\lambda_{q\to q+1}^{cond_j}}{\lambda_{q+1\to q}^{cond_j}}$, where $q = 1, \ldots, K-1$

row of Fig. 2.4.

The thermal energy due to leakage power of a prismatic component with temperature T_1 is represented by the TCPN module at the second row of Fig. 2.4. The leakage coefficients δ and α depend on the technology node and on the type of subcircuit (SRAM, logic) modeled by each element. Our model allows setting both coefficients for each element separately, honoring a precise floor-plan if required. However, the schedulers presented in this dissertation are unaffected by the magnitude of the coefficients. Therefore, we assume throughout this Thesis an average leakage behavior for every element, setting $\delta = 0.1$ and $\alpha = 0.001$ [67]. The marking at place p_1^{com} represents the average temperature of the solid component.

Integrating conduction, convection and heat generation

The TCPN of the thermal module of Fig. 2.5 depicts the thermal model of a component which generates heat due to the dynamic power of jobs executions and transfers heat by conduction and convection to its surrounding components. The model is derived from the basic models of Figs. 2.3 and 2.4, by merging the corresponding places of the basic TCPNs

that represent the temperature of the component. By the principle of superposition of the thermal system, the merged model is dynamically equivalent to merging the Petri net places in the TCPN [86].

The number of prismatic thermal components necessary to permit an accurate thermal analysis depends on the desired accuracy. A single thermal CPU_j is modeled by the aggregation of k prismatic components into a larger component with a volume V_{CPU_j}. Note that there exist prismatic components without heat generation.

2.5 Putting the modules all together: the global TCPN model

Fig. 2.6 shows a detailed part of the TCPN global model, joining tasks, CPU and thermal modules framed by dashed boxes. At the frontier of these modules there are *boundary places and transitions*. The modules are linked by connecting the boundary nodes using arcs. The arcs from places p_i^{cc} to transitions $t_{i,j}^{alloc}$ represent the jobs of τ_i allocated to processor CPU_j. Places $p_{i,j}^{exec}$ and arcs going from $t_{i,j}^{exec}$ to $p_{i,j}^{exec}$ are added to merge the models. The marking of place $p_{i,j}^{exec}$ stands for the total amount of jobs of τ_i that have been executed in CPU_j from the initial time.

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

Fig. 2.7 shows how all the modules are connected to integrate a monolithic global model. As of today, we have developed a tool to automate the generation of the matrices and vectors representing the thermal model, and we are currently extending the tool to automatically include CPUs and tasks.

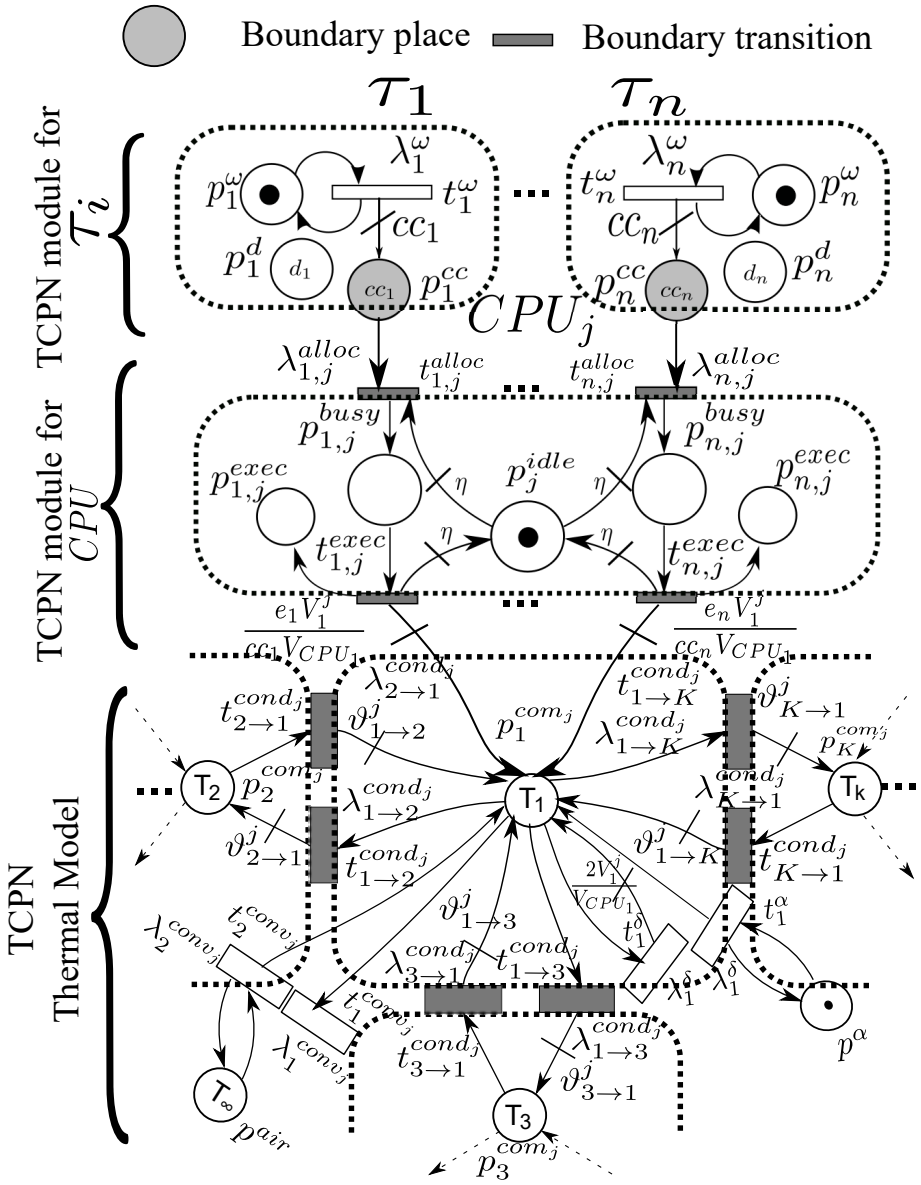


Figure 2.6: Detailed TCPN global model of a uniprocessor system. It includes the task, CPUs and thermal models.

2.5.1 Fundamental equation

Based on the dynamical behavior of a TCPN system of Fig. 2.6 the fundamental equation of the global model can be derived using Eq. (1.5) and formulated by the following equations:

$$\begin{aligned}
\dot{\boldsymbol{m}}_{\boldsymbol{T}} =& \boldsymbol{C}_{\boldsymbol{T}}\boldsymbol{\Lambda}_{\boldsymbol{T}}\boldsymbol{\Pi}_{\boldsymbol{T}}(\boldsymbol{m})\boldsymbol{m}_{\boldsymbol{T}} + \boldsymbol{C}_{\boldsymbol{a}}\boldsymbol{\Lambda}_{\boldsymbol{a}}\boldsymbol{\Pi}_{\boldsymbol{a}}(\boldsymbol{m})\boldsymbol{m}_{\boldsymbol{a}} \\
& + \boldsymbol{C}_{\boldsymbol{\mathcal{P}}}^{\boldsymbol{exec}}\boldsymbol{f}^{\boldsymbol{exec}} && \text{(2.3a)} \\
\dot{\boldsymbol{m}}_{\boldsymbol{a}} =& 0 && \text{(2.3b)} \\
\dot{\boldsymbol{m}}_{\boldsymbol{\mathcal{T}}} =& \boldsymbol{C}_{\boldsymbol{\mathcal{T}}}\boldsymbol{\Lambda}_{\boldsymbol{\mathcal{T}}}\boldsymbol{\Pi}_{\boldsymbol{\mathcal{T}}}(\boldsymbol{m})\boldsymbol{m}_{\boldsymbol{\mathcal{T}}} - \boldsymbol{C}_{\boldsymbol{\mathcal{T}}}^{\boldsymbol{alloc}}\boldsymbol{w}^{\boldsymbol{alloc}} && \text{(2.3c)} \\
\dot{\boldsymbol{m}}_{\boldsymbol{\mathcal{P}}} =& \boldsymbol{C}_{\boldsymbol{\mathcal{P}}}\boldsymbol{\Lambda}_{\boldsymbol{\mathcal{P}}}\boldsymbol{\Pi}_{\boldsymbol{\mathcal{P}}}(\boldsymbol{m})\boldsymbol{m}_{\boldsymbol{\mathcal{P}}} + \boldsymbol{C}_{\boldsymbol{\mathcal{P}}}^{\boldsymbol{alloc}}\boldsymbol{w}^{\boldsymbol{alloc}} && \text{(2.3d)} \\
\dot{\boldsymbol{m}}_{\boldsymbol{exec}} =& \boldsymbol{C}_{\boldsymbol{\mathcal{P}}}^{\boldsymbol{exec}}\boldsymbol{f}^{\boldsymbol{exec}} && \text{(2.3e)}
\end{aligned}$$

The fundamental equation of the global model represents the dynamic behavior of the system: task arrivals, CPU throughput and temperature variation. Actually, this equation expands to a set of equations, because the complete system is modeled by the interconnecting TCPN basic models (Figs. 2.3, 2.4), which requires the addition of a few extra linking places or transitions. Note that Eqs. (2.3) is in a state space form, thus being able to design controllers. The actual input control is the vector $\boldsymbol{w}^{\boldsymbol{alloc}}$, it represents the necessary flow going through transitions $\boldsymbol{t}^{\boldsymbol{alloc}}$; i.e., the allocation of tasks to CPUs to meet temporal constraints. The controlled variables are $\boldsymbol{m}_{\boldsymbol{exec}}$ and $\boldsymbol{m}_{\boldsymbol{T}}$; the former represents the accumulated task execution and must be equal to the required task execution over time; the later represents the temperature of the CPUs.

The differential equations undelying the thermal model in the TCPN are represented in Eq. (2.3a) and (2.3b). The marking $\boldsymbol{m}_{\boldsymbol{T}}$ and $\boldsymbol{m}_{\boldsymbol{a}}$ represent the temperature vector of the system and the fixed ambient temperature respectively. Matrices $\boldsymbol{C}_{\boldsymbol{T}}$, $\boldsymbol{\Lambda}_{\boldsymbol{T}}$, and $\boldsymbol{\Pi}_{\boldsymbol{T}}(\boldsymbol{m})$ are the incidence matrix, the firing rate transitions and the configuration matrix respectively of the subnet in Fig. 2.6 thermal module, corresponding to the prismatic components of the floor-plan. Matrices $\boldsymbol{C}_{\boldsymbol{a}}$, $\boldsymbol{\Lambda}_{\boldsymbol{a}}$, and $\boldsymbol{\Pi}_{\boldsymbol{a}}(\boldsymbol{m})$ represent the incidence matrix, the firing rate transitions and the configuration matrix respectively, for the ambient temperature of the thermal sub net.

The TCPN model for task arrival is described by Eq. (2.3c). The marking $\boldsymbol{m}_{\boldsymbol{\mathcal{T}}}$ stands for the dynamic task allocation in the system. Matrices $\boldsymbol{C}_{\boldsymbol{\mathcal{T}}}$, $\boldsymbol{\Lambda}_{\boldsymbol{\mathcal{T}}}$, and $\boldsymbol{\Pi}_{\boldsymbol{\mathcal{T}}}(\boldsymbol{m})$ are the incidence matrix, the firing rate transitions and the configuration matrix corresponding to the sub net of the task model in Fig. 2.6 modules task model.

The CPU behavior is modeled by Eq. (2.3d). The marking $\boldsymbol{m}_{\boldsymbol{\mathcal{P}}}$ corresponds to the places of the CPU modules of Fig. 2.6. Matrices $\boldsymbol{C}_{\boldsymbol{\mathcal{P}}}$, $\boldsymbol{\Lambda}_{\boldsymbol{\mathcal{P}}}$, and $\boldsymbol{\Pi}_{\boldsymbol{\mathcal{P}}}(\boldsymbol{m})$ represent the incidence

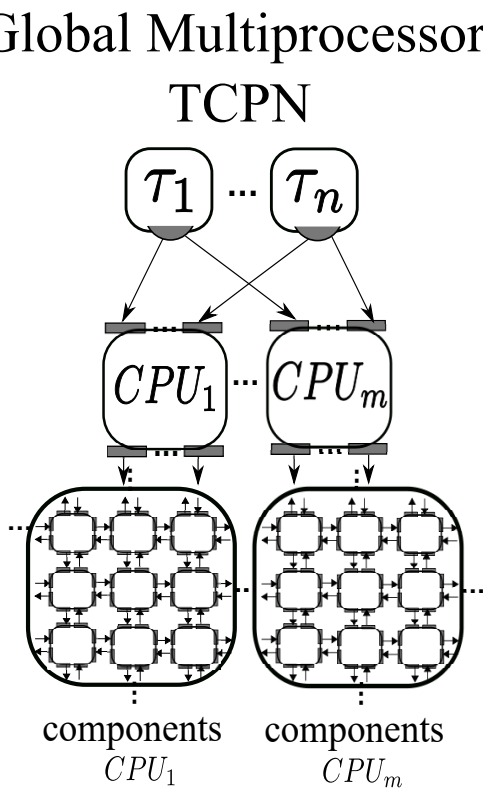


Figure 2.7: Global multiprocessor TCPN schematic representation.

matrix, the firing rate transitions and the configuration matrix corresponding to the CPU models.

Matrices $\boldsymbol{C}_{\boldsymbol{\mathcal{P}}}^{\boldsymbol{exec}}$, $\boldsymbol{C}_{\boldsymbol{\mathcal{T}}}^{\boldsymbol{alloc}}$ and $\boldsymbol{C}_{\boldsymbol{\mathcal{P}}}^{\boldsymbol{alloc}}$ stand for the connections of transitions $t_{i,j}^{exec}$ and $t_{i,j}^{alloc}$ from (to) places in the thermal model, task arrival model and CPU model, respectively. Fig. 2.7 shows the connection of the aforementioned models, by means of the allocation transitions $t_{i,j}^{alloc}$ and the execution transitions $t_{i,j}^{exec}$. Notice that the task model does not depend on the temperature model or the CPU model, however it depends on $\boldsymbol{w}^{\boldsymbol{alloc}}$ (see Eq. 2.3c). The vector $\boldsymbol{w}^{\boldsymbol{alloc}}$ represent the jobs assigned per unit time. These jobs are removed from the task model and allocated into the processors for execution. In addition, the CPU model evolves independently from the thermal model (Eq. 2.3a). The marking $\boldsymbol{m}_{\boldsymbol{exec}}$ (the integral of $\boldsymbol{f}^{\boldsymbol{exec}}$ Eq. 2.3e) stands for the accumulated executed jobs.

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

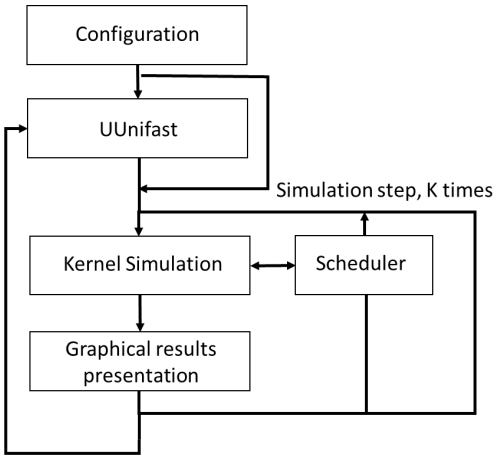


Figure 2.8: Simulator Architecture

2.6 Methology and Simulation Environment

This Section describes a framework tool for testing HRT Schedulers. This tool consists of three modules. The first one helps the user to define the problem: task set with periods, deadlines and WCET in CPU cycles, along with the task allocation models, CPU usage, temperature and energy consumption. In the second module, the user selects the scheduler model. Finally the last module allows to execute the simulation either in manual or automatic mode. In manual mode the simulator uses the task set data provided in the first section. In automatic mode, the task set is generated by parameterizing the integrated UUniFast algorithm [87].

2.7 Framework architecture

The simulation framework has been programmed in MatLab [88]. It is distributed as open source software, publicly available, and can be used out-of-the-box [80]. Its modular design provides flexibility to test a wide variety of schedulers and platforms. It includes a signal routing interface allowing switching among different user-defined scheduling algorithms.

This framework makes easier to evaluate a large number of different scenarios, where the platform (hardware), the set of tasks, and schedulers can be defined by the user through a Graphical User Interface (GUI).

Fig. 2.8 shows the main modules of the framework. The user introduces the set of tasks, platform, and the scheduler in the *configuration* module. After the completion of the configuration stage, the simulation is executed. Later the results are presented to the user. The

following subsections describe these modules.

2.7.1 Configuration module

This module allows the introduction of all the information required by the framework. It is organized in four sections: a) Task definition, b) CPU definition, c) Thermal definition, and d) Scheduler definition. The order in which the information is introduced is irrelevant. The user can resort to default values or turn-off some sections, like the thermal definition.

a) The Task definition section allows two different ways to introduce the information. One way is to manually enter the number of tasks along with their parameters. Another way is to let the algorithm UUniFast ([87]) automatically generate a task set with the desired characteristics.

b) The CPU definition section requires two parameters: the number of CPUs and their frequency scale. The frequency scale is a set of normalized frequencies at which the platform could operate, where 1 indicates the highest frequency. The framework assumes homogeneous CPUs, a feature that will be relaxed in future releases.

c) The thermal definition section requires the Printed Circuit Board (PCB) and CPU dimensions and thermal parameters, ambient temperature and the maximum operating temperature.

d) The Scheduler definition section is generic. The user either, can select one from a set of pre-programmed schedulers, or can define his/her own scheduler.

The user should consider signals like CPU temperature, system utilization, energy consumption, (or a subset of them) to design her/his own scheduler. At every time step, the section generates a matrix of size *Number of tasks* $\times$ *Number of CPUs*, where the $ij-th$ entry represents the allocation of the $i-th$ task to the $j-th$ CPU.

2.7.2 UUniFast module

The user can opt for the UUniFast algorithm ([87]) to generate the task set in the configuration stage, indicating the number of tasks to be generated, the system utilization U and the hyper-period H. The output is a feasible real-time task set with random task periods, WCETs, deadlines and consumed energy. UUniFast generates one set of tasks at a time. It allows to stress the scheduler under analysis with different set of tasks.

2.7.3 Kernel simulation module

The *Kernel* module builds up a global simulation model according to the task set, CPUs, thermal and energy parameters and the selected scheduler, and runs the simulation.

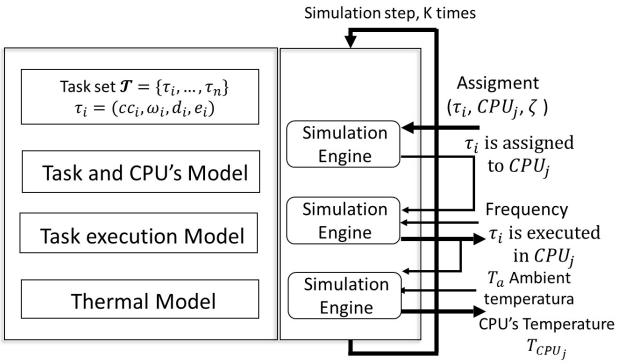


Figure 2.9: Kernel of the simulator.

The model represents task, CPU and thermal modules by a set of ordinary differential equations, and generates the signals to/from the scheduler. The scheduler can be represented either as a continuous or a discrete system. Accordingly, the scheduler can be modeled by the paradigm of differential equations or finite automata.

Next subsections describe how to build module's models. Later, we explain how the modules are merged into a global model.

2.7.4 Scheduler module

The scheduler module allows to select at configuration time, one of the scheduling policies available in the framework, or any scheduler defined by the user. The signals available to the scheduler from other modules are $m_{i,j}^{exec}$ representing the amount of τ_i executed by CPU_j, and m_{T_j} the CPU_j temperature. The signals that other modules require from the scheduler are the task allocations signals ${}_j w_i^{alloc}$.

2.7.5 Building the global model

The global model is simulated by solving the system:

$$\begin{aligned} \dot{\boldsymbol{M}} &= \boldsymbol{AM} + \boldsymbol{Bw}^{\boldsymbol{alloc}} + \boldsymbol{B'm_a} \\ \boldsymbol{Y} &= \boldsymbol{SM} \end{aligned} \tag{2.4}$$

where $\boldsymbol{M} = [\boldsymbol{m_{\mathcal{T}}}, \boldsymbol{m_{\mathcal{P}}}, \boldsymbol{m_T}]^{\boldsymbol{T}}$, and the matrices are:

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{C_{\mathcal{T}}\Lambda_{\mathcal{T}}\Pi_{\mathcal{T}}} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{C_{\mathcal{P}}\Lambda_{\mathcal{P}}\Pi_{\mathcal{P}}} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{C_{\mathcal{P}}^{exec}\Lambda^{exec}\Pi^{exec}} & \boldsymbol{C_T\Lambda_T\Pi_T} \end{bmatrix} \tag{2.5}$$

$$\boldsymbol{B} = \begin{bmatrix} \boldsymbol{C_{\mathcal{T}}^{alloc}} \\ \boldsymbol{C_{\mathcal{P}}^{alloc}} \\ \mathbf{0} \end{bmatrix} \quad \boldsymbol{B'} = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \boldsymbol{C_a\Lambda_a\Pi_a} \end{bmatrix} \tag{2.6}$$

$$\boldsymbol{S} = \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{A_{\mathcal{P}}} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \boldsymbol{S_T} \end{bmatrix} \tag{2.7}$$

$\boldsymbol{A_{\mathcal{P}}}$ corresponds to the output matrix for task execution and $\boldsymbol{S_T}$ represents the temperature output matrix. Thus the output vector $\boldsymbol{Y} = [\boldsymbol{m_{exec}}, \boldsymbol{m_T}]$ contains the task execution and the temperature of each processor. If the thermal module is not selected for simulation the global model is slightly different: vector $\boldsymbol{M}$ will only contain $\boldsymbol{M} = [\boldsymbol{m_{\mathcal{T}}}, \boldsymbol{m_{\mathcal{P}}}]$, every matrix (Eqs. 2.5 - 2.7) will lose its last row, and Eq. (2.5) will also lose its last column, and the output vector $\boldsymbol{Y} = [\boldsymbol{m_{exec}}]$ will only contain the task execution.

Chapter 3

A HRT Fluid Scheduler

This chapter presents a fluid-time scheduler based on a sliding mode controller where the sliding surface is related to fluid task executions. The scheduler leverages the TCPN model introduced in Ch. 2. Also, this chapter proposes an implementation of this fluid scheduler as a feasible discrete scheduler where the number of task migrations and preemptions is bounded.

3.1 Introduction

This chapter introduces a *fluid* scheduler based on a sliding mode control technique. The scheduler is derived from the TCPN global model obtained in Ch. 2, which constitutes itself a modeling methodology for fluid schedulers in RT systems. The on-line discretization of this fluid scheduler allows reducing task switching and migration. The scheduler here presented shares with *Pfair* algorithms their HRT optimality in terms of CPU utilization. Furthermore, the scheduler keeps the number of context switches and migrations reasonably low, by solving the control equation only upon the job deadlines, profiting from the qualities of deadline partitioning algorithms.

3.2 Temporal Fluid schedule (TFS)

This chapter proposes a *fluid* schedule based on a sliding mode control technique [89]. The choice of this type of control obeys to the fact that it is an on-off control type, which requires

negligible calculations, a key factor in an on-line scheduler. From now on, ζ represents the current time. The control approach herein reported starts by computing the task *fluid*-schedule function:

$$FSC_{\tau_i}(\zeta) = \frac{c_i}{\omega_i}\zeta \tag{3.1}$$

This function represents the optimal *fluid execution* of task τ_i [39, 38]. It is the time that task τ_i should have run until time ζ to meet the RT constraints, also known as the *fluid share*.

3.3 RT Sliding surface

The purpose of the sliding mode controller in this approach is to compute the *error fluid execution*. The *execution error* of task τ_i (denoted $E_{\tau_i}(\zeta)$) is the difference between the marking $m_i^{exec} = \sum_{j=1}^{m} m_{i,j}^{exec}$ (total amount of jobs of τ_i allocated) in the global TCPN model (Fig. 2.6) and the optimal *fluid* execution.

$$E_{\tau_i}(\zeta) = FSC_{\tau_i}(\zeta) - m_i^{exec}(\zeta) \tag{3.2}$$

To build the sliding surface, let $x_i^1 = E_{\tau_i}(\zeta)$ and $x_i^2 = \sum_{j=1}^{m} m_{i,j}^{busy}$. Then, the following system holds:

$$\begin{array}{rcl} \dot{x^1}_i & = & u_i - \lambda_i^{exec} x_i^2 \\ \dot{x^2}_i & = & w_i^{alloc} - \lambda_i^{exec} x_i^2 \end{array} \tag{3.3}$$

where $\lambda_i^{exec} = \lambda_{i,1}^{exec} = \ldots = \lambda_{i,m}^{exec}$, $w_i^{alloc} = \sum_{j=1}^{m} {}_j w_i^{alloc}$ and $u_i = \frac{cc_i}{\omega_i}$. Thus, for this system, the sliding surface may then be set to be of the form:

$$\mathcal{S}_i(\zeta) = \frac{K_1}{\lambda_i^{exec}} x_i^1 + \frac{u_i}{\lambda_i^{exec}} - x_i^2 \tag{3.4}$$

where K_1 is a real positive number.

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

$$x_i^2 = -\frac{K_1}{\lambda_i^{exec}} x_i^1 - \frac{u_i}{\lambda_i^{exec}} \tag{3.5}$$

Therefore

$$\dot{x^1}_i = -K_1 x_i^1 \tag{3.6}$$

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

3.4 Control law computation

The following continuous sliding-mode control law is used [89] to reach the task optimal fluid execution of each task τ_i and meet the temporal requirements.

$$w_i^{alloc}(\zeta) = K_2 sign(\mathcal{S}_i(\zeta)) + \frac{K_1}{\lambda_i^{exec}} u_i \tag{3.7}$$

where $sign(x) = 1$ if $x \geq 0$; 0 otherwise.

Proposition 3.1 *Let $\mathcal{T}$ and $\mathcal{P}$ be the sets of n tasks and m processors, respectively. Let FSC_{τ_i} be the optimal fluid execution function of task τ_i. If the control law given by Eq. (3.7) is applied to the global system with $K_1 = \lambda_i^{exec}$ and guaranteed that $0 < K_2 < u_i$ then each RT fluid execution error $E_i(\zeta)$ converges to zero.*

Proof **3.1** *The total controlled flow of a transitions $t_{i,j}^{alloc}$ for each task τ_i is given by $w_i^{alloc} = \sum_{j=1}^{m} {}_j w_i^{alloc}(\zeta)$ in Eq. (3.7), where ${}_j\hat{w}_i^{alloc}(\zeta)$ is the control action. Note that when ${}_j w_i^{alloc} > 0$, transition $t_{i,j}^{alloc}$ is fired, i.e., jobs of τ_i are being allocated to CPU_j.*

To prove the asymptotic stability of Eq. (3.2), a Lyapunov function can be defined, satisfying $V(0) = 0$, $V(x) > 0$ and $\dot{V}(x) < 0$ $\forall x \neq 0$ ([90]). Let us consider the following quadratic candidate Lyapunov function V:

$$V(\mathcal{S}_1, \ldots, \mathcal{S}_n) \;=\; \tfrac{1}{2} \sum_{i=1}^{n} \mathcal{S}_i^2 \tag{3.8}$$

$V = 0$ iff each $\mathcal{S}_i = 0$, and $V > 0$ if any $\mathcal{S}_i \neq 0$. Therefore, V can be considered a Lyapunov function (and Eq. (3.2) is asymptotic stable) iff $\dot{V} < 0$ *for any $\mathcal{S}_i \neq 0$.*

To prove this, we first compute the derivative of V:

$$\begin{aligned}\dot{V} &= \sum_{i=1}^{n} \mathcal{S}_i \dot{\mathcal{S}}_i = \sum_{i=1}^{n} \mathcal{S}_i \left(-K_2 sign(\mathcal{S}_i) + x_i^2(\lambda_i^{exec} - K_1)\right) \\ &\leq \sum_{i=1}^{n} -K_2 \mathcal{S}_i sign(\mathcal{S}_i) + |\mathcal{S}_i||x_i^2||\lambda_i^{exec} - K_1| \\ &\leq \sum_{i=1}^{n} -|\mathcal{S}_i| \left(K_2 - |x_i^2||\lambda_i^{exec} - K_1|\right)\end{aligned} \tag{3.9}$$

Now, let us prove that following holds for each term in the sum (i.e. for each task) $-|\mathcal{S}_i| \left(K_2 - |x_i^2||\lambda_i^{exec} - K_1|\right) < 0$. To do that, the term is negative if K_2 satisfies $K_2 > |x_i^2||\lambda_i^{exec} - K_1|$. Moreover, since the controlled flow is always positive (i.e., $w_i^{alloc} = K_2 sign(\mathcal{S}_i) + \frac{K_1}{\lambda_i^{exec}} u_i > 0$), then if $\mathcal{S}_i$ is positive, K_2 must satisfy $K_2 > -\frac{K_1}{\lambda_i^{exec}} u_i$, otherwise (when $\mathcal{S}_i$ is negative) K_2 must satisfy $K_2 < \frac{K_1}{\lambda_i^{exec}} u_i$

Therefore, for each task τ_i and the RT fluid execution error $E_i(\zeta)$ converges to zero asymptotically.

The correct selection of the gains K_1 and K_2 for the derived control law provided by Eq. (3.7) allows tracking the optimal fluid schedule for each task τ_i. Therefore, the fluid execution $\boldsymbol{m_{exec}}$ follows a fluid schedule. However, any practical implementation of this scheduling requires the discretization of the fluid instant share of the CPU.

3.5 On-line discretization of a Fluid schedule

The previous section has proved that the proposed *fluid* scheduler is theoretically feasible (i.e, the fluid execution error converges to zero and tasks meet the time constraints). As all *fluid* schedulers, it triggers an unfeasible number of task preemptions and migrations. To deal with this issue we provide a discrete implementation described by Alg. 1 based on a deadline partitioning approach.

3.5.1 Deadline partitioning approach

Due to the periodicity of the schedule, we can limit the schedule of the tasks up to the hyperperiod (from time 0 to time H) [39]. We consider the ordered set of all job deadlines to define scheduling intervals, as in deadline partitioning ([24]). Each task τ_i must be executed $n_i = \frac{H}{\omega_i}$ times within the hyperperiod H. Thus every $q * \omega_i$, where $q = 1, ..., n_i$ is a deadline that must be considered in the analysis. These deadlines can be ordered and joined in the set $SD_i = \{sd_i^1, ..., sd_i^{n_i}\}$. A general set of deadlines is defined as $SD = SD_0 \cup ... \cup SD_{|\mathcal{T}|}$

ALGORITHM 1: On-line discretization of the *fluid* schedule

Input: The TCPN of the task set $\mathcal{T}$, the ordered set SD where any $sd_k \in SD$ is lower or equal than H. The quantum Q. The task *fluid*-schedule functions FSC_{τ_i}.

Output: A feasible discrete schedule

1 **Initialize** $i = 1$, $sd = sd_i$, $\zeta = 0$, $M_i^{exec}(\zeta) = 0\ \forall \tau_i \in \mathcal{T}$;

2 **for** $\zeta \leq H$ **do**

3 All tasks are preempted from the processors;

4 $RE_i(\zeta) = FSC_{\tau_i}(sd) - M_i^{exec}(\zeta)$; /* Compute remaining jobs */

5 $ET(\zeta) = \{\tau_i | RE_i(\zeta) > 0\}$; /* Compute the set of tasks to be executed */

6 $PR_i(\zeta) = m_i^{exec}(\zeta) - M_i^{exec}(\zeta)$; /* Compute the priority for every task τ_i in $ET(\zeta)$ */

7 **for** $j = 1$ *to* m **do**

8 Select the task τ_a for CPU_j with the highest priority value in $ET(\zeta)$;

9 Remove the task τ_a from $ET(\zeta)$;

10 $M_a^{exec}(\zeta + Q) = M_a^{exec}(\zeta) + Q$; /* Compute the discrete execution of task τ_a */

11 Remove τ_a from ET_j ;

12 **end**

13 *Simulate the global TCPN model from* ζ *to* $\zeta + Q$; /* Solve Eqs. (2.3) to compute m^{exec} */

14 $\zeta = \zeta + Q$; /* Update time */

15 **if** $\zeta == sd$ **then**

16 $i = i + 1, sd = sd_i$

17 **end**

18 **end**

where $SD_0 = \{0\}$. The elements of SD can be arranged in ascendant order and renamed as $SD = \{sd_0, ..., sd_\alpha\}$, where α is the last deadline. The scheduling interval $I_{SD}^k = [sd_{k-1}, sd_k]$ is defined and $|I_{SD}^k| = sd_k - sd_{k-1}$ represents the scheduling interval duration.

Assuming a quantum-based time and without loss of generality let Q denote the quantum length in time units. Q is defined as the GCD of the elements $sd_i \in SD$ (where $i \leq \alpha$) and the values of the function FSC_{τ_i} evaluated at sd_i.

Alg. 1 computes a discrete schedule from the fluid schedule introduced in Sec. 3.2. The control is applying to the system when the algorithm simulates the global TCPN model in step 13. It solves Eqs. (2.3) to compute m^{exec}. The algorithm requires a new discrete variable $M_i^{exec}(\zeta)$ which represent the discrete execution of task τ_i. The evolution of $M_i^{exec}(\zeta + Q) = M_i^{exec}(\zeta) + Q$ is determined by the algorithm, and means that Q CPU cycles of task τ_i are allocated. Thus, this equation represents the total amount of CPU cycles of task τ_i that have been allocated from the initial time. The discrete schedule resulting from Algorithm 1 equals the fluid schedule at every deadline time $sd_k \in SD$, i.e. it ensures that the discrete schedule meets all deadlines of all tasks.

If at any time ζ, $sd_i < \zeta < sd_{i+1}$, it holds that $FSC_{\tau_i}(sd_k) > M_i^{exec}(\zeta)$ (the required fluid schedule at the end of the interval is bigger than the current discrete schedule), then τ_i must be allocated in a CPU. Thus the m tasks with the current positive greatest RE

(*remaining job execution*, $RE_{\tau_i}(\zeta) = FSC_{\tau_i}(sd_k) - M_i^{exec}(\zeta)$) and PR (*task priority function*, $PR_{\tau_i}(\zeta) = m_i^{exec}(\zeta) - M_i^{exec}(\zeta)$ must be allocated to a CPU.

The value of $m_i^{exec}(\zeta)$ can be replaced by $FSC_{\tau_i}(\zeta)$ since they have the same value. However, we use it here because we will later include thermal characteristics, making these values no longer be equal because we will have to balance thermal and temporal trade-offs.

Proposition 3.2 *If a feasible fluid schedule is given as input to Algorithm 1, then the resulting discrete schedule has the following properties.*

1. *It meets task time constraints at every scheduling point $sd_k \in SD$.*
2. *The number of task migrations and preemptions is bounded.*

Proof **3.2** *Part 1) Sentence 1.*

From the definition of quantum, we know that the time interval $[sd_k, sd_{k+1}]$ is divided by the quantum into $D_k^{k+1} = (sd_{k+1} - sd_k)/Q$ time sub-intervals. From [39] we know that the fluid schedule meets the task time constraints at every time, which is specially true at the sd_k points. Moreover, since the fluid schedule is feasible then, at time sd_k, the $CPU's$ are capable to execute the required percentage $FSC_{\tau_a}(sd_k)$ of any task τ_a. Assuming that there exist m processors and n tasks, and since the m processors are capable to execute the fluid schedule, then:

$$m \cdot D_k^{k+1} \geq \sum_{i=1}^{n} \left(FSC_{\tau_i}(sd_{k+1}) - FSC_{\tau_i}(sd_k)\right) \tag{3.10}$$

At time zero, both the fluid schedule and discrete schedule have executed zero percentage of each task, thus the discrete schedule meets task time constraints at sd_0.

Now, we will show that if the discrete schedule meets the fluid schedule at any sd_k then it meets the fluid schedule at any sd_{k+1} as well.

Proceeding by contradiction, assume that the discrete schedule does not meet the fluid schedule at sd_{k+1}. Then the remaining jobs functions are positive for some tasks (i.e. the discrete PN has executed these tasks in a lower percentage than the fluid one). For the sake of explanation, suppose that there exists only one task τ_a such that $RE_{\tau_a}(sd_{k+1}) = N_a > 0$. Since Eq. (3.10) holds, then the processors have the capability to execute the required percentage of tasks at time sd_{k+1}. However, since τ_a was not allocated in the required percentage ($FSC_{\tau_a}(sd_{k+1})$), then τ_a had some remaining jobs at time $\zeta = sd_{k+1} - Q$.

Therefore, two cases are possible:

Case 1: If $N_a = \alpha Q$, where $\alpha = 1$ (i.e., $PR_{\tau_a} = Q$)

In this case yet another two possibilities arise:

a) τ_a was fired at time ζ, then τ_a finishes the required percentage of execution, i.e., $RE_{\tau_a}(sd_{k+1}) = 0$, which is a contradiction.

b) τ_a was not allocated at time ζ. Thus, according to step 8 of the algorithm, m tasks, different from τ_a, were found having priority larger or equal than that of τ_a, thus they were allocated.

If the behavior of the algorithm is analyzed at time $\zeta - Q$ (a previous time step), it will result that m tasks were found, different than τ_a, having larger or equal priorities than that of τ_a (otherwise, task τ_a would be allocated and thus finished its execution). By repeating this analysis, going back in time until sd_k, it will be obtained that $RE_{\tau_a}(sd_k) = N_a > 0$, i.e., the discrete schedule does not meet the fluid schedule at sd_k, reaching a contradiction.

Case 2: If $N_a = \alpha Q, \alpha > 1$, then at time ζ, $PR_{\tau_a}(\zeta) > PR_{\tau_i}(\zeta)$, for some task $\tau_i \neq \tau_a$. Thus τ_a was allocated at time ζ and $\alpha = \alpha - 1$. Let $\zeta = \zeta - Q$. If sd_k is reached then $RE_{\tau_a}(sd_k) = N_a > 0$, i.e., the discrete schedule does not meet the fluid schedule at sd_k, a contradiction, otherwise if $\alpha - 1 == 1$ then go to Case 1.

Part 2) Sentence 2.

Since the hyper-period H is divided into $I = \frac{H}{Q}$ subintervals and task migrations and preemptions occur at the end of this subintervals then the number of task migrations and preemptions is bounded by I.

Complexity

The algorithm executes $I = \frac{H}{Q}$ times, where H is the hyper-period, thus the loop in step 7 of the algorithm runs I times. The instruction inside this loop runs in polynomial time in the size of the transitions of the TCPN. As mentioned in previous section, the execution on the TCPN is polynomial and therefore the algorithm is polynomial too.

3.6 Example

Let us consider the task set $\mathcal{T} = \{\tau_1, \tau_2, \tau_3\}$, with $\tau_1 = (9, 10, 10)$, $\tau_2 = (9, 10, 10)$, $\tau_3 = (8, 40, 40)$, running on two CPUs. The hyper-period is $H = 40$. The *fluid* schedule $m_i^{exec}(\zeta)$ is obtained by applying the *fluid* controller (Fig. 3.1). The fluid schedule and the error E_{τ_i} converge to zero, and therefore the *fluid* schedule based on the sliding-mode control meets the time constraints.

Now, the on-line Algorithm 1 discretizes the fluid schedule. The set of deadlines is $SD =$

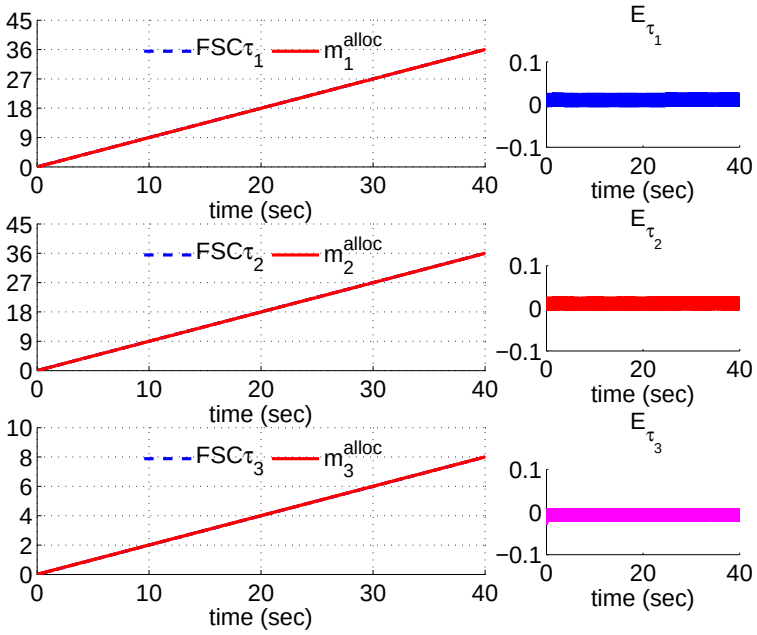


Figure 3.1: A feasible *fluid* schedule. The *execution error* E_{τ_i} converges to zero due the control action.

$\{10, 20, 30, 40\}$ and the quantum $Q = 1$. The computed discrete schedule appears in Fig. 3.2. At every $sd \in SD$ the tokens accrued in $m_i^{exec}(sd)$ meet the fluid schedule requirements. For instance, at $sd = 10$ the fluid schedule indicates that task τ_1 requires 9 time units. Analyzing Fig. 3.2, τ_1 is executed 1 time units in CPU_1 from $\zeta = 0$ to $\zeta = 1$, 3 time units in CPU_2 from $\zeta = 1$ to $\zeta = 4$ and 5 time units in CPU_1 from $\zeta = 5$ to $\zeta = 10$. Thus, τ_1 is executed 9 time units at $sd = 10$.

3.7 Conclusions

The approximation introduced in this Chapter shows that the modeling methodology of Ch. 2 allows the successful design and test of a HRT global scheduler which leverages a simple on-off controller. This is a first step in the search for a suitable method for temperature control under HRT constraints. The fluid functions are computed on deadline partitioning basis, but the scheduling decisions are taken on a quantum based, which was the first objective of the approach in order to allow for a fine thermal control in the future. These results were presented in [81]. However, there are two principal issues to be tackled yet, in addition to the inclusion of thermal constraints. First, to lower the computing overhead. Although the controller is simple, Alg. 1 produces a high number of task preemption and migration that

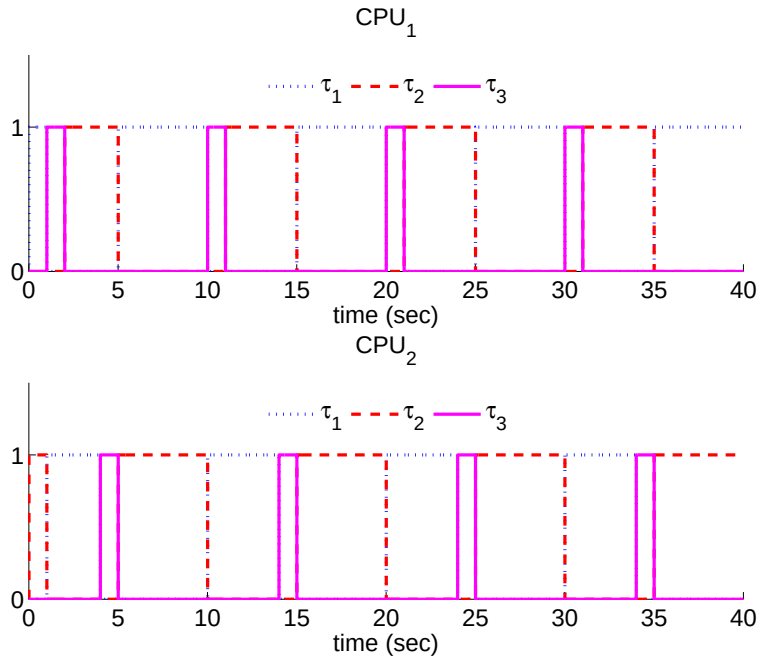


Figure 3.2: Discretization of the *fluid* schedule for CPU_1 and CPU_2.

compromises the performance of the system. Second, there is a single global job queue, actually a list ordered by job priority. Jobs are allocated to CPUs by priority order irrespective the CPU they were running on during the previous scheduling period. There chances for applying CPU affinity heuristics are slim: all jobs are preempted every Q, and a job could come back to the same CPU it was running on. The inclusion of thermal restrictions on this scheme led to a better solution.

Chapter 4

A HRT Thermal-Aware Fluid Scheduler

This chapter presents OLDTFS*, a global on-line fluid scheduler that meets both thermal and RT constraints, resorting to a feedback control technique. The thermal schedule feasibility is proved by a LPP that captures HRT and thermal constraints as linear constraints. If there exists a feasible solution, then the LPP solution represents the correct execution of tasks as a continuous linear functions over time (*fluid schedule functions*), honoring HRT and thermal constraints. Then, the fluid scheduler is discretized allowing control on context switching and migrations. The feedback controller that implements the proposed global scheduler allows the system to recover from disturbances such as CPU detentions due to environmental hazards causing energy interruptions or thermal peaks.*

4.1 Introduction

The previous chapters established the basis to cope with the inclusion of thermal constraints in HRT scheduling leveraging the advantages of TCPNs as a modeling and simulation tool. A single global TCPN models tasks, CPUs and thermal activity. The evolution of the complete system is described by Eqs. (2.3). The inclusion of thermal constraints entails a new compromise between overhead (context-switches and migrations) and precision, which is strongly related to selecting a discretization based on fixed (quantum) or variable (deadline-based) scheduling intervals. Alg. 1 in Ch. 3 provided such a trade-off.

In this chapter, we first determine the existence of a feasible HRT thermal-aware schedule.

Then, we present *OLDTFS*, a controller implementing the RT thermal-aware scheduler in two stages, an off-line and an on-line stage. The off-line stage alleviate the overhead of Alg. 1 by means of a Linear Programming Problem (LPP) which computes the fraction of each task job to be run at each CPU. This facilitates the management of a per-CPU queue in the ulterior on-line scheduler. Also, the controller is now able to deal with disturbances as long as there is some utilization slack. The overhead can be greatly reduced by using an opportunistic DP-fair under specific circumstances. The solution can be applied to a simulated system or to a physical system (Sec. 4.3.4). This approach has been accepted for publication [82].

4.2 HRT thermal-aware fluid schedule Feasibility

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

Definition 4.1 *A schedule is* thermal feasible *if it satisfies the required deadlines every hyperperiod and the temperature of the processors do not exceed a maximum operating value* $\boldsymbol{T_{max}} = [\boldsymbol{T_{max_1}}, \ldots, \boldsymbol{T_{max_m}}]$. *Moreover, since the schedule is periodic, the temperature of the processors must satisfy that temperatures at the start and at the end of the hyperperiod are equal.*

The feasibility analysis is accomplished in three parts. First, we present the temporal and CPU utilization restrictions. Second, we derive the thermal constraints. Both restrictions are described as linear functions of certain coefficients ${}_j\beta_i$ denoting fractions of task executions. Third, we pose a LPP to compute the fraction of each task job to be run at each CPU, subject to the constraints mentioned above, whose solution provides the values of the coefficients ${}_j\beta_i$. The continuous controller in Sec. 4.3 uses these coefficients.

4.2.1 Computation of the temporal fluid-schedule functions

The task *fluid* schedule function is computed as:

$$FSC_{\tau_i}(\zeta) = \frac{c_i}{\omega_i}\zeta \tag{4.1}$$

where ζ is the current time. This function represents the optimal *fluid* execution of task τ_i at time ζ [39] [38]. Eq. (4.1) is defined as optimal in the literature, although it just provides a feasible schedule for task τ_i. We obtain a *fluid* schedule function for each pair CPU_j and task τ_i as follows.

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

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

$$i_a = \sum_{j=1}^{m} {}_j\beta_n = \frac{H}{w_a} \quad \forall\ \tau_a\ \in \mathcal{T} \qquad [\textit{Temporal Constraint}] \tag{4.2}$$

In order to fulfill task deadlines, the CPU_j utilization must not exceed the capacity of the processors:

$$\sum_{\tau_i \in \mathcal{T}} \frac{c_i \times {}_j\beta_i}{H} \leq 1 \quad \forall\ CPU_j\ \in \mathcal{P} \qquad [\textit{CPU utilization Constraint}] \tag{4.3}$$

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

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

$$FSC_{\tau_i}(\zeta) = {}_1FSC_{\tau_i}(\zeta) + \dots + {}_mFSC_{\tau_i}(\zeta) = \frac{{}_1\beta_i \times c_i}{H}\zeta + \dots + \frac{{}_m\beta_i \times c_i}{H}\zeta \tag{4.4}$$

where ${}_jFSC_{\tau_i}(\zeta) = \frac{{}_j\beta_i \times c_i}{H}\zeta$ stands for the fluid schedule function of τ_i at time ζ in CPU_j. Eqs. (4.2) and (4.3) provide the temporal restrictions for computing the ${}_j\beta_a$ coefficients.

4.2.2 Thermal Analysis

The relationship between task allocation and temperature was formulated in Section 2.5.1 using Eqs. (2.3a) and (2.3d). Eq. (2.3d) can be rewritten by applying the following change of variable. Let $\boldsymbol{M_T^1 = m_T}$ and $\boldsymbol{M_T^2 = m_{\mathcal{P}}}$, thus $\boldsymbol{M_T = [M_T^1\ M_T^2]^T}$ denotes the state variables corresponding to the thermal behavior of processors and the task dynamic allocation respectively. Hence, the part of the model that represents the relationship between the temperature and the power consumption due to task allocation is:

$$\begin{aligned} \boldsymbol{\dot{M}_T} &= \boldsymbol{AM_T + Bw^{alloc} + B'm_a} \\ \boldsymbol{Y_T} &= \boldsymbol{S'M_T} \end{aligned} \tag{4.5}$$

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

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{C_T\Lambda_T\Pi_T} & \boldsymbol{C_{\mathcal{P}}^{exec}\Lambda^{exec}\Pi^{exec}} \\ \mathbf{0} & \boldsymbol{C_{\mathcal{P}}\Lambda_{\mathcal{P}}\Pi_{\mathcal{P}}} \end{bmatrix} \boldsymbol{B'} = \begin{bmatrix} \boldsymbol{C_a\Lambda_a\Pi_a m_a} \\ \mathbf{0} \end{bmatrix} \boldsymbol{B} = \begin{bmatrix} \mathbf{0} \\ \boldsymbol{C_{\mathcal{P}}^{alloc}} \end{bmatrix} \tag{4.6}$$

As stated before, vector $\boldsymbol{w^{alloc}}$ is the controlled flow of the *allocation transitions* (it stands for the task allocation rate to CPUs). The task allocated to CPU ($m_{\mathcal{P}}$) times $\boldsymbol{C_{\mathcal{P}}^{exec}\Lambda^{exec}\Pi^{exec}(m)}$ represents the dynamic computational power of the tasks under execution. The matrix $\boldsymbol{S' = [S\ 0]}$ is the output matrix which selects the components representing the temperature of the CPUs, and $\boldsymbol{Y_T}$ stands for the temperature of the components of the processor. The schedule must be periodic from a temporal and thermal point of view. Thus, the initial temperature must be equal to the final temperature (evaluated at the hyperperiod H) to meet the thermal feasibility condition, i.e.: $\boldsymbol{Y_T(H) = S'M_T(0)}$.

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

$$\begin{aligned} \boldsymbol{M_{T_{ss}}} &= \boldsymbol{-A^{-1}(Bw^{alloc} + B'm_a)} \\ \boldsymbol{Y_{T_{ss}}} &= \boldsymbol{S'M_{T_{ss}}} \end{aligned} \tag{4.7}$$

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

$$\boldsymbol{S'M_{T_{ss}} \leq T_{max}} \tag{4.8}$$

Combining Eqs. (4.7) and (4.8),

$$\boldsymbol{-S'A^{-1}Bw^{alloc} \leq T_{max} + S'A^{-1}B'm_a} \qquad [\textit{Thermal constraint}] \tag{4.9}$$

The previous equation provides the thermal constraints that must fulfill the allocation of tasks to the processors ($\boldsymbol{w^{alloc}}$). Now, $\boldsymbol{w^{alloc}}$ must be represented as a function of ${}_j\beta_i$ parameters. In steady state, the flows in transitions $\boldsymbol{t^{alloc}}$ and $\boldsymbol{t^{exec}}$ are equal. According to the temporal restrictions in Eq. (4.2), the flow required in $t^{alloc}_{i,j}$ must be $\frac{{}_j\beta_i \times c_i}{H}$, matching the number of jobs of task τ_i assigned to CPU_j per time unit. The next subsection computes the ${}_j\beta_i$ coefficients.

4.2.3 computation of the fluid execution

The computation of the fluid execution of tasks can be formulated as a LPP, to deal with the steady state temperature and the *maximum temperature* of each CPU. The problem consists in finding a solution for the ${}_j\beta_i$ subject to thermal, temporal and CPU utilization constraints, which translates into Eq. (4.10):

$$\begin{aligned}
&min \sum_{j=1}^{m} \sum_{\tau_a \in \mathcal{T}} {}_j\beta_a \\
&s.t. \\
&\quad \left.\begin{array}{ll} \boldsymbol{-SA^{-1}B}\left[\ \sum_{\tau_a} \frac{{}_1\beta_a c_a}{H},\ \cdots,\ \sum_{\tau_a} \frac{{}_m\beta_a c_a}{H}\ \right]^T \leq \boldsymbol{T_{max} + SA^{-1}B'm_a} & \textit{Thermal Constraint} \\ \sum_{j=1}^{m} {}_j\beta_n = \frac{H}{\omega_n} \quad \forall i = 1, \ldots, n & \textit{Temporal Constraint} \\ \sum_{\tau_a \in \mathcal{T}} \frac{c_a \times {}_m\beta_a}{H} \leq 1 \quad \forall j = 1, \ldots, m & \textit{CPU util. Constraint} \end{array}\right\}
\end{aligned} \tag{4.10}$$

The thermal constraint in Eq. (4.9) states that the temperature of the processors due to task execution must not violate the maximum allowed temperature. The estimated temporal constraints, Eqs. (4.2) and (4.3) ensure that the number of jobs within the hyperperiod allows for a feasible schedule. The last constraint ensures that the computation utilization of each CPU is ≤ 1.

Proposition 4.1 *Consider a system as defined in 1.19 and the thermal-aware fluid scheduler problem 1.8.1 for this system. If the linear programming problem LPP (4.10) defined for the*

system parameters has a feasible solution, then there exists a thermal-aware HRT feasible schedule.

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

To show the existence of such a scheduler (and complete the proof), we propose in the next section a continuous scheduler whose result is discretized in a later stage. It uses the functions ${}_jFSC_{\tau_i}(\zeta)$ (target function) and $m_{i,j}^{exec}(\zeta)$ (actual task execution) to define an error as the difference between these two quantities. By controlling task allocation with $\boldsymbol{w^{alloc}}$ the proposed scheduler brings the error down to zero. This is proved in proposition 4.2, where the Lyapunov functions guarantee that the error reaches a zero value.

4.3 Thermal-Aware HRT Fluid Scheduler Control

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

We leverage a sliding mode feedback controller to manage workload execution [89]. The purpose of the controller is to keep the *RT thermal fluid execution error* $\mathcal{E}_{T_{i,j}}(\zeta)$ equal to zero. This error is defined as the difference between the task fluid execution ${}_jFSC_{\tau_i}(\zeta)$ of a task τ_i in CPU_j and its actual execution percentage ($m_{i,j}^{exec}(\zeta)$ in Fig. 2.6a):

$$\mathcal{E}_{T_{i,j}}(\zeta) = {}_jFSC_{\tau_i}(\zeta) - m_{i,j}^{exec}(\zeta) \tag{4.11}$$

4.3.1 RT Thermal Sliding surface

In the sliding mode technique, a sliding surface $\mathcal{S}$ is first designed as a function of the system's state in such way that, if the system is controlled so that $\mathcal{S}$ is null then the error converges to zero. In order to construct the sliding surface, let $x_{i,j}^1 = \mathcal{E}_{T_{i,j}}(\zeta)$ and $x_{i,j}^2 = m_{i,j}^{busy}$. Then, the following system holds:

$$\begin{aligned} \dot{x^1}_{i,j} &= \varphi_{i,j} - \lambda_{i,j}^{exec} x_{i,j}^2 \\ \dot{x^2}_{i,j} &= {}_jw_i^{alloc} - \lambda_{i,j}^{exec} x_{i,j}^2 \end{aligned} \tag{4.12}$$

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

$$\mathcal{S}_{i,j}(\zeta) = \frac{K_1}{\lambda_{i,j}^{exec}} x_{i,j}^1 + \frac{\varphi_{i,j}}{\lambda_{i,j}^{exec}} - x_{i,j}^2 \tag{4.13}$$

where K_1 is a real positive number.

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

$$x_{i,j}^2 = -\frac{K_1}{\lambda_{i,j}^{exec}} x_{i,j}^1 - \frac{\varphi_{i,j}}{\lambda_{i,j}^{exec}} \tag{4.14}$$

Therefore

$$\dot{x^1}_{i,j} = -K_1 x_{i,j}^1 \tag{4.15}$$

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

4.3.2 Control law computation

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

$${}_j w_i^{alloc}(\zeta) = {}_j \hat{w}_i^{alloc}(\zeta) + \tfrac{K_1}{\lambda_{i,j}^{exec}} \varphi_{i,j} \tag{4.16}$$

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

Proposition 4.2 *Let $\mathcal{T}$ and $\mathcal{P}$ be the sets of n tasks and m processors, respectively. Let ${}_j FSC_{\tau_i}$ be fluid schedule function of task τ_i and CPU_j, obtained by solving the linear programming problem of Eq. (4.10). If the control law given by Eq. (4.16) is applied to the global system with $K_1 = \lambda_{i,j}^{exec}$ and $0 < K_2 < \varphi_{i,j}$ then each RT thermal fluid execution error $\mathcal{E}_{T_{i,j}}(\zeta)$ converges to zero.*

Proof **4.1** *The controlled flow of a transition $t_{i,j}^{alloc}$ is given by ${}_j w_i^{alloc}(\zeta)$ in Eq. (4.16), where ${}_j \hat{w}_i^{alloc}(\zeta)$ is the control action. Note that when ${}_j w_i^{alloc} > 0$, transition $t_{i,j}^{alloc}$ is fired, i.e., jobs of τ_i are being allocated to CPU_j.*

In order to prove the asymptotic stability of Eq. (4.11), a Lyapunov function can be defined, satisfying $V(0) = 0$, $V(x) > 0$ *and* $\dot{V}(x) < 0\ \forall x \neq 0$ *([90]). Let us consider the following quadratic candidate Lyapunov function* V*:*

$$V(\mathcal{S}_{1,1}, \ldots, \mathcal{S}_{n,m}) \;=\; \tfrac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{m}\mathcal{S}_{i,j}^2 \tag{4.17}$$

$V = 0$ iff *each* $\mathcal{S}_{i,j} = 0$*, and* $V > 0$ *if any* $\mathcal{S}_{i,j} \neq 0$*. Therefore,* V *can be considered a Lyapunov function (and Eq. (4.11) is asymptotic stable)* iff $\dot{V} < 0$ *for any* $\mathcal{S}_{i,j} \neq 0$*. To prove this, we first compute the derivative of* V*:*

$$\begin{aligned}
\dot{V} &= \sum_{i=1}^{n}\sum_{j=1}^{m}\mathcal{S}_{i,j}\dot{\mathcal{S}}_{i,j} = \sum_{i=1}^{n}\sum_{j=1}^{m}\mathcal{S}_{i,j}\left(-{}_j\hat{w}_i^{alloc} + x_{i,j}^2(\lambda_{i,j}^{exec} - K_1)\right)\\
&\leq \sum_{i=1}^{n}\sum_{j=1}^{m} -K_2\mathcal{S}_{i,j}sign(\mathcal{S}_{i,j}) + |\mathcal{S}_{i,j}||x_{i,j}^2||\lambda_{i,j}^{exec} - K_1| \qquad (4.18)\\
&\leq \sum_{i=1}^{n}\sum_{j=1}^{m} -|\mathcal{S}_{i,j}|\left(K_2 - |x_{i,j}^2||\lambda_{i,j}^{exec} - K_1|\right)
\end{aligned}$$

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

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

The correct selection of the gains K_1 and K_2 for the derived control law provided by Eq. (4.16) allows tracking the optimal fluid schedule for each τ_i and CPU_j, which is thermally feasible. Therefore, the fluid execution $\boldsymbol{m_{exec}}$ follows a fluid schedule.

4.3.3 OLDTFS (On-line discretization of the Thermal-Aware fluid schedule)

The Alg. 2 (*OLDTFS*) implements our thermal-aware RT fluid scheduler, aiming to balance a good thermal control and a suitable context-switch overhead. Abiding by the hypothesis

ALGORITHM 2: On-line discretization of Thermal fluid schedule (OLDTFS)

Input: The TCPN of the set of tasks $\mathcal{T}$, the ordered set SD where any $sd_k \in SD$ is lower or equal than H. The quantum Q. The fluid schedule ${}_jw_i^{alloc}$. The number of processors m

Output: The discrete schedule ${}_jW_i^{alloc}$

Initialize $i = 1$, $sd = sd_i$, $\zeta = 0$, $M_{i,j}^{exec}(\zeta) = 0\ \forall \tau_i \in \mathcal{T}$ and $\forall CPU_j \in \mathcal{P}$;

for $\zeta \leq H$ **do**
  All tasks are preempted from the processors;
  $RE_{i,j}(\zeta) = {}_jFSC_{\tau_i}(sd) - M_{i,j}^{exec}(\zeta)$; /* Compute remaining jobs */
  $ET_j(\zeta) = \{\tau_i | RE_{i,j}(\zeta) > 0 \forall CPU_j \in \mathcal{P}\}$; /* Compute the set of tasks to be executed */
  $PR_{i,j}(\zeta) = m_{i,j}^{exec}(\zeta) - M_{i,j}^{exec}(\zeta)$; /* Compute the priority for every task τ_i in $ET_j(\zeta)$ */
  for $j = 1$ *to* m **do**
    ${}_jW_i^{alloc} = 0,\ 1 \leq j \leq m,\ 1 \leq i \leq n$;
    τ_a = task with the highest priority value in $ET_j(\zeta)$; /* $ET_j(\zeta)$: task queue of CPU_j */
    ${}_jW_a^{alloc} = 1$; /* Set τ_a to run on CPU_j */
    ; /* Now remove τ_a from all tasks queues but $ET_j(\zeta)$: */
    Remove τ_a from $ET_k(\zeta)$ for all $1 \leq k \leq m$ and $k \neq j$;
    $M_{a,j}^{exec}(\zeta + Q) = M_{a,j}^{exec}(\zeta) + Q \times {}_jW_a^{alloc}$; /* Compute the discrete execution of τ_a */
    Remove τ_a from ET_j;
    Switch to τ_a in CPU_j ; /* Only if scheduling tasks in a real, physical system */
  end
  Simulate the CPU TCPN model from ζ *to* $\zeta + Q$; /* Solve Eqs. (2.3.e) to compute m^{exec} */
  $\zeta = \zeta + Q$; /* Update time */
  if $\zeta == sd$ **then**
    $i = i + 1, sd = sd_i$
  end
end

exposed in Sec. 1.7, we leverage the approach taken in the previous Ch. 3, which limits the fluid schedule computation to the set of deadlines, but places the scheduling points on a quantum basis. Later we will show that under precise circumstances it is possible to limit the scheduling points to the set of deadlines as well, and still accomplish the RT and thermal constraints without requiring a fixed quantum, all the more lowering the overhead. *OLDTFS* yields a discrete schedule that closely tracks the fluid one ($\boldsymbol{m_{exec}}$) by computing a schedule up to the hyperperiod (from time zero to time H). We first define the set of deadlines SD and quantum Q as in [81].

For every time interval $[sd_k,\ sd_{k+1}]$ (where $sd_{k+1} \leq H$ and $k =, 0, 1, \ldots$), at time ζ, if ${}_jFSC_{\tau_i}(sd_k) > M_{i,j}^{exec}(\zeta)$, τ_i must be allocated to CPU_j so that it runs to the point required by the fluid scheduler, to warrant that the k-th job of τ_i completes before its k-th deadline.

The thermal-compliant HRT fluid schedule (${}_jFSC_{\tau_i}(sd_k)$) was computed in the off-line stage. We define the *remaining jobs execution* until the next deadline in the whole set of deadlines as:

$$RE_{i,j}(\zeta) = {}_jFSC_{\tau_i}(sd_k) - M^{exec}_{i,j}(\zeta)$$

for each task τ_i and CPU_j (line 4), and the *task priority function* as (line 6 of Alg. 2):

$$PR_{i,j}(\zeta) = m^{exec}_{i,j}(\zeta) - M^{exec}_{i,j}(\zeta)$$

All tasks τ_i such that $RE_{i,j}(\zeta) > 0$ must be allocated to CPU_j before the next quantum. The discretized time that task τ_i must run on CPU_j starting at time $\zeta + Q$ is given by the following equation (line 13):

$$M^{exec}_{i,j}(\zeta + Q) = M^{exec}_{i,j}(\zeta) + Q \times {}_jW^{alloc}_i \tag{4.19}$$

where

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

Thus, Eq. (4.19) yields the execution time slice of each task job in a CPUs when dispatched for the next interval. We can leverage the TCPN to entirely simulate a physical system, or to exclusively compute the fluid schedule, discretized on a quantum basis in Alg. 2. In the first case, besides accounting for the runtime, we have to actually dispatch the task on a specific CPU (statement in line 15).

4.3.4 Overview of the Thermal-Aware RT scheduler

Fig. 4.1 summarizes the three parts of the proposed scheduler highlighting the system control signals. The dotted boxes above and below (A, B) exemplify the evolution of the principal system's input and output signals during normal operation (A) or under disturbance (such as a CPU detention, B).

During the off-line stage, the scheduler is build up as described in Ch. 2, from the TCPN model, according to the thermal parameters of the materials of the system, the number of prisms, and the parameters of the CPUs and HRT task set, obtaining its state equation. Then, the constraints of the LPP are stated from this model, and the LPP solution provides the coefficients for the fluid scheduling functions ${}_jFSC_{\tau_i}(\zeta)$ (S1), which guarantee the accomplishment of the thermal and HRT constraints in H (hyperperiod).

As described in Sec. 2.6, all this process has been fully automated, as part of a publicly available simulation framework [80] that includes a number of schedulers that can be simulated out-of-the-box.

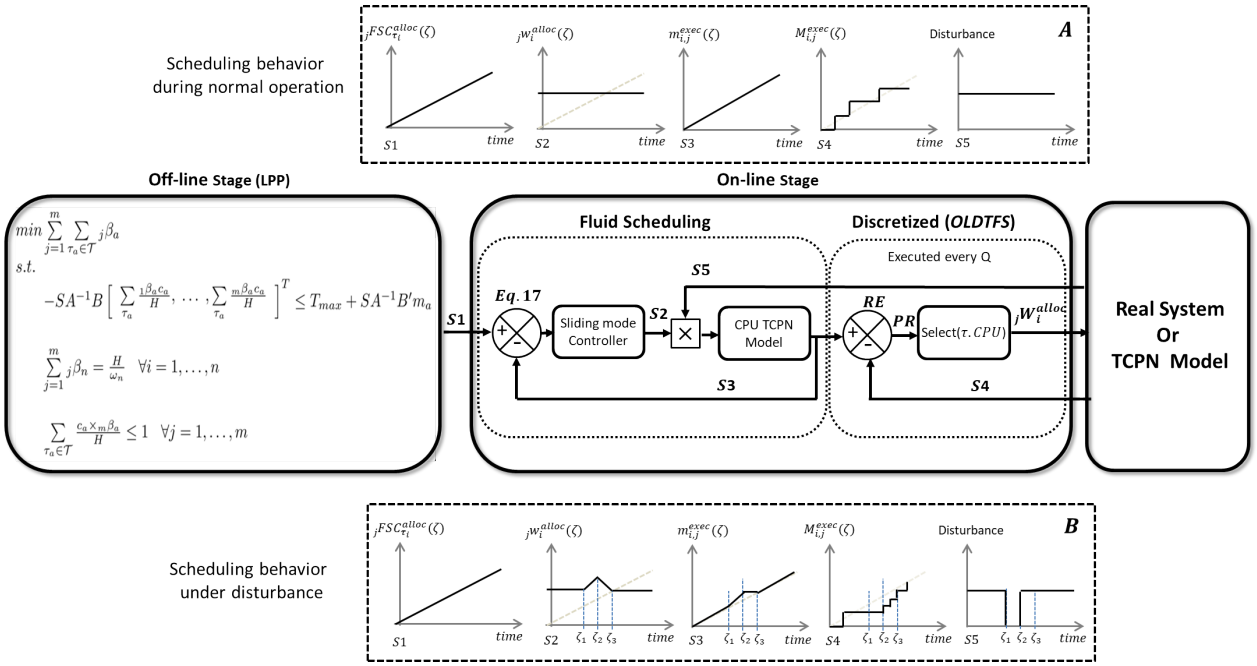


Figure 4.1: Thermal-Aware HRT Fluid Scheduler overview. To simplify the view we respectively name ${}_jFSC_{\tau_i}(\zeta)$, ${}_jw_i^{alloc}$, $m_{i,j}^{exec}$ and $M_{i,j}^{exec}$ as the signal arrays $S1, S2, S3, S4$. $S5$ provides the feedback to capture disturbances.

The on-line stage starts with the sliding mode controller allocating tasks to CPUs in the TCPN model. The controller throttles the fluid marking ${}_jw_i^{alloc}$ (S2), which is proportional to the error $\mathcal{E}_{T_{i,j}}(\zeta)$ (Eq. 4.11), to make S3 (the current fluid schedule, $m_{i,j}^{exec}$ in the TCPN) follow S1 (the fluid schedule calculated off-line), closing $\mathcal{E}_{T_{i,j}}(\zeta)$.

During normal operation S3 will always follow S1 (upper dotted box A in Fig. 4.1). Alg. 2 iterates every quantum Q. It computes the error as the difference between the fluid execution time S3 (provided by the TCPN model) and the actual discrete execution time S4 (Alg. 2 line 4), dynamically adjusts priorities accordingly (line 6) and selects, allocates and dispatches the jobs until the next scheduling point (line 18).

Under disturbance, S5 is asserted, which modifies the dynamics of the TCPN model. For example, if a disturbance can lead a CPU to a halt, the flow of transitions $t_{i,j}^{exec}$ (representing the active execution of task τ_i on CPU_j, Fig. 2.6) is computed as $f_{i,j}^{exec} = (\lambda_{i,j}^{exec} m_{i,j}^{busy}) \times S5$. For example (Fig. 4.1 box B) halting CPU_j at time ζ_1 ($S5 = 0$) will halt the simulated CPU in the TCPN model, increasing S2 so that $m_{i,j}^{exec}$ (S3) higher that ${}_jFSC_{\tau_i}(\zeta)$ (S1). The controller will increase S2 so that S3 increases accordingly until ζ_2. By ζ_3, S1 has reached its normal level and S3 has caught up with the S1 line.

OLDTFS discretizes S3 by increasing the ratio at which the tasks smitten by the CPU halt are dispatched in successive quanta (compare S4 in box A and B). A scheduler lacking a continuous controller like this are not capable of recovering from disturbances. A sliding mode controller is specially suitable when dealing with off-line and on-line signals.

Complexity

The algorithm executes $I = H/Q$ times (H is the hyperperiod), thus the outer loop in Alg. 2 runs I times. The instructions inside this loop run in polynomial time in the size of the number of tasks and CPUs. All the instructions in the inner loop (lines $7 - 14$) run in polynomial time. Moreover, Eq. (2.3.e) (Alg. 2 line 15) can be rewritten in a discrete state space representation. Hence the TCPN is simulated in polynomial time, and therefore the algorithm is polynomial too. Note that Eqs. (2.3a) and (2.3b) are not computed in the algorithm because they are not required to solve $\boldsymbol{m}^{\boldsymbol{exec}}$.

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

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

If the linear programming problem in Eq. (4.10) has a solution, it means that the coefficients ${}_j\beta_i$ of the solution meet the thermal, temporal and CPU utilization constraints in a steady-state temperature. These coefficients can now be used to develop an algorithm based on *DP-Fair*. However, in contrast with the deadline partitioning technique, based on the task local utilization, we leverage the coefficients ${}_j\beta_i$ as the task share that must run on each CPU for each time slice, defined by all the deadlines of all tasks in the system. The algorithm is opportunistic in that it prioritizes the thermal constraint and makes the processors to become idle when the temperature of the CPUs approaches the temperature limit. When resuming execution, the algorithm catches up and meet the temporal constraints.

4.4 Simulation Results

The following experiments show the ability of the proposed scheduler to meet HRT and thermal constraints and to deal with disturbances. Also, we compare *OLDTFS* with a straightforward implementation of a *DP-Fair* scheduler [37].

4.4.1 Experimental setup

We consider a package with two homogeneous $1cm \times 1cm$ microprocessors mounted over a $5cm \times 5cm$ copper heat spreader. The microprocessors and the copper heat spreader are respectively $0.5mm$ and $1mm$ thick. The temperature of the surrounding air is fixed and set to 35^oC, in the range of an environment type C according to [91]. The convection coefficient of the heat spreader is $h = 0.001\frac{W}{mm^{2o}C}$. The isotropic thermal properties of the materials in the package are shown in Table I. We assume CPUs with caches and speculative mechanisms non-existent or turned-off, and tasks running at a fixed frequency $F = 1GHz$. Thus, WCET, deadline and period time bounds can be stated more accurately. We include scheduling and context switch overhead in the WCET. We have used for the thermal model a mesh of 25×25 prisms for the heat spreader, 5×5 prisms per CPU, totaling 675 prisms.

The experiments are performed using the simulation framework presented in Sec. 2.7, which automates the description of the TCPN model including the thermal parameters of the system and the generation of task sets. To solve the LPP in Eq. (4.10) during the off-line stage we use the `lingprog` (*simplex* algorithm) function of MatLab [88]. We obtain the predicted task execution ($\dot{\boldsymbol{m}}_{\boldsymbol{exec}}$) during the on-line stage with the ODE45 solver. All the experiments can be reproduced with the publicly available package [80].

4.4.2 *OLDTFS* results

The first experiment considers a thermo-hydraulic system. Both the water level in a tank and its temperature are controlled by a certain algorithm, resulting in a periodic task set $\mathcal{T} = \{\tau_1, \tau_2, \tau_3\}$ running on a MPSoC with two cores; the maximum operating temperature of both cores is $T_{max_{1,2}} = 100^oC$. Tasks τ_2 and τ_3 control the level and temperature respectively. Task τ_1 (acquisition task) is used to read the sensors every sample period. We have computed the sample period according to Shannon's theorem. The relevant task parameters are WCET (in CPU cycles), sample time ω and deadline d (with $\omega = d$ in this case), resulting in $\tau_1 = (2 \times 10^9, 4, 4, 6.4)$, $\tau_2 = (5 \times 10^9, 8, 8, 8)$, $\tau_3 = (6 \times 10^9, 12, 12, 9.6)$, and the consumed energy e. Task independence is achieved by the correct computation of the sample time. If ω_1 is two or more times shorter than ω_2 and ω_3, then the level and temperature variables are known before the execution of τ_2 and τ_3, i.e. tasks are independent of each other.

Fig. 4.2 depicts the schedule obtained by the on-line discretization of the fluid temporal scheduler. Fig. 4.3 shows the evolution of temperature in both CPUs until the hyperperiod, using two different values for the quantum. Temporal and thermal constraints are met with both values, but temperature variations are much narrower with $Q = 0.05$ than with the computed quantum $Q = 0.5$. A theoretical infinitesimal quantum would match the optimal thermal solution with infinitesimal context switches.

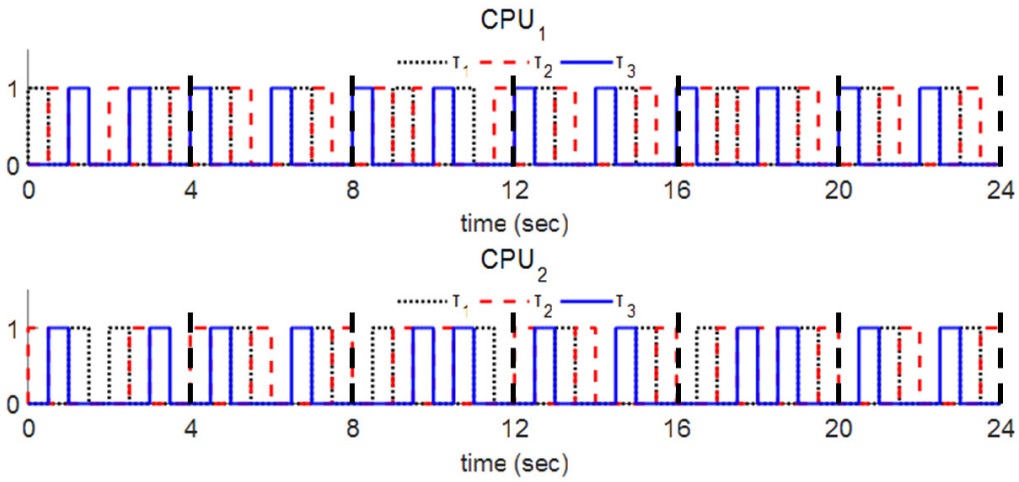


Figure 4.2: Schedule computed by *OLDTFS* algorithm for the example of subsection 4.4.2. Its execution produce a rising in the CPUs temperature depicted in Fig. 4.3.

A second experiment shows the sensitivity of *OLDTFS* to computation utilization and its ability to cope with a significant number of tasks pushing temperature well over the limit. We leverage the algorithm UUniFast [87] to generate 10^3 task sets with computation utilization varying from 1 to 2 to evaluate the thermal feasibility, and only considering task sets which are computationally feasible.

Fig. 4.4 shows the maximum temperature per CPU reached during the first hyperperiod, while varying the computation utilization (x-axis). The maximum temperature does not only depend on the computational utilization, but on the power consumption of each task set too (see LPP in Eq. 4.10). *OLDTFS* keeps the maximum temperature under control along the whole range of utilization values. The smaller the quantum, the finer the thermal control at the cost of a higher overhead.

4.4.3 *OLDTFS* vs. *DP-Fair*

We have seen that *OLDTFS* allows a great control on temperature, but finding a balance to keep a low overhead requires experimenting with different quanta. *DP-Fair* only schedules tasks on the set of deadlines, which generally yields fewer scheduling points than when using a quantum. However, a baseline *DP-Fair* can fail to keep the temperature inside a safe region.

The following example makes evident this observation. It considers two identical CPUs and three tasks such that $\mathcal{T} = \{\tau_1, \tau_2, \tau_3\}$, where $\tau_1 = (2 \times 10^9, 4, 4, 6.4)$, $\tau_2 = (5 \times$

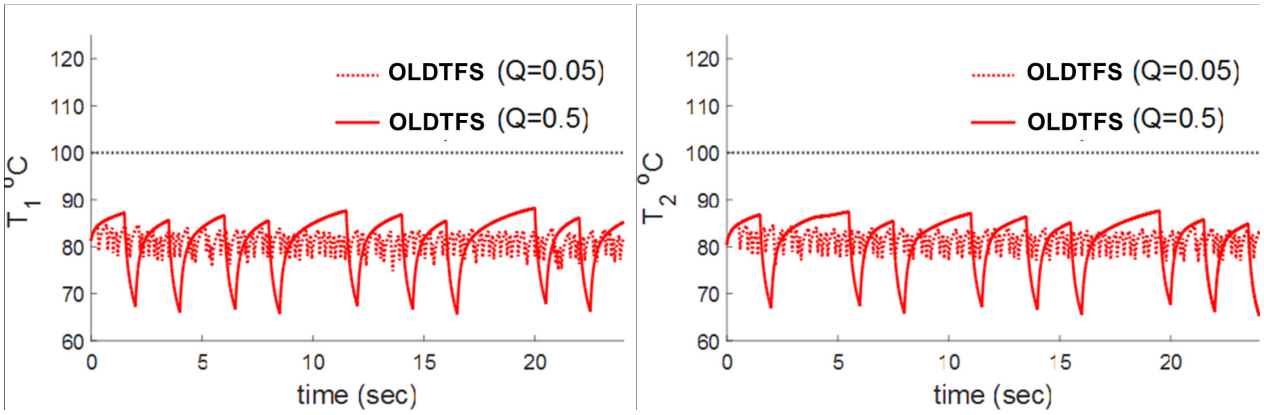


Figure 4.3: CPUs temperature evolution due to the execution of the computed schedule in the example of subsection 4.4.2, for different quantums. The initial conditions of CPU_1 and CPU_2 are $m_T(0)[1] = 81.5664^oC$, $m_T(0)[2] = 80.7352^oC$ respectively. Note that the temperature does not exceed the bound $T_{max} = 100^oC$. As expected, a smaller quantum means a lower temperature variation.

$10, 8, 8, 8)$, $\tau_3 = (6 \times 10, 12, 12, 9.6)$. The CPU utilization is $U = 1.625$, and therefore the scheduling problem has a solution on a two processor platform.

Both *DP-Fair* and *OLDTFS* compute feasible schedules. Fig. 4.5 shows the temperature evolution for both schedules. *OLDTFS* evenly distributes the execution and idle time of the CPUs over the scheduling points, keeping temperature under the 90°C bound. In contrast, *DP-Fair* runs all the tasks as soon as possible during the scheduling points, with accrued idle time at the end of each scheduled point, which leads to a temperature violation. The number of context switches and migrations in *DP-Fair* is much lower than in *OLDTFS* nonetheless.

4.4.4 Thermal-aware Opportunistic *DP-Fair*

Fig. 4.6 compares the temperature variations yielded by *OLDTFS* with a small quantum and by the thermal-aware opportunistic *DP-Fair* presented in Sec. 4.3.5. The temperature bound is set to $T_{max} = 100^oC$ for a set of task $\mathcal{T} = \{\tau_1, \tau_2, \tau_3\}$, where $\tau_1 = (2 \times 10^9, 4, 4, 9.6), \tau_2 = (5 \times 10^9, 8, 8, 9.6), \tau_3 = (6 \times 10^9, 12, 12, 9.6)$, and $\mathcal{P} = \{CPU_1, CPU_2\}$. The hyperperiod is $H = 24$ and the CPU utilization is $U = 1.62$. Both algorithms meet the thermal constraint, but *OLDTFS* reaches a lower temperature.

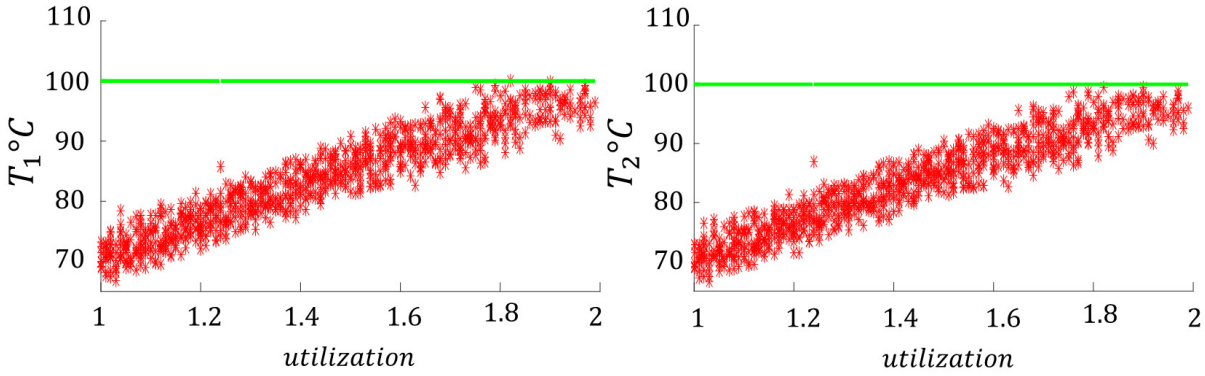


Figure 4.4: Maximum temperature of processors considering 10^3 tasks sets, generated by the UniFast algorithm, with utilization varying 1-2. Each marker represents the maximum temperature of a task set with the corresponding utilization.

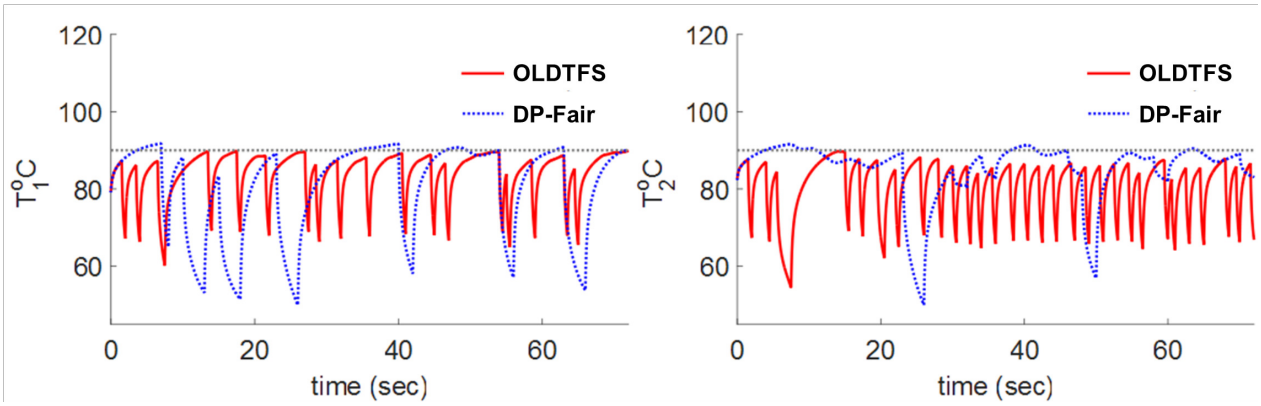


Figure 4.5: Temperature evolution in both CPUs for the example in Sec. 4.4.3. *DP-Fair* produce a temporal feasible schedule, but unlike *OLDTFS* it violates the thermal bound.

4.4.5 Disturbance recovering with *OLDTFS*

The feedback controller embedded in the scheduler can recover the system from disturbances such as CPU detentions due to hazardous environmental conditions, energy interruptions and other eventualities, as we discussed when describing the overall structure and operation of the scheduler (Sec. 4.3.4, Fig. 4.1 box B). As a proof of concept, we consider three tasks $\mathcal{T} = \{\tau_1, \tau_2, \tau_3\}$, where $\tau_1 = (2 \times 10^9, 4, 4, 6.4), \tau_2 = (5 \times 10^9, 8, 8, 8), \tau_3 = (6 \times 10^9, 12, 12, 9.6)$ running on two processors. A disturbance causes CPU_1 to halt during the time interval $[5, 7]$, resuming task execution in $\zeta > 7$. Fig. 4.7 represents the task execution according to our scheduler (Fig. 4.1). The three plots on the left correspond to the allocation of the three tasks on CPU_1. During the time interval $[5, 7]$, $m_{1,i}^{exec}(\zeta)$ (S3) is unable to track the fluid

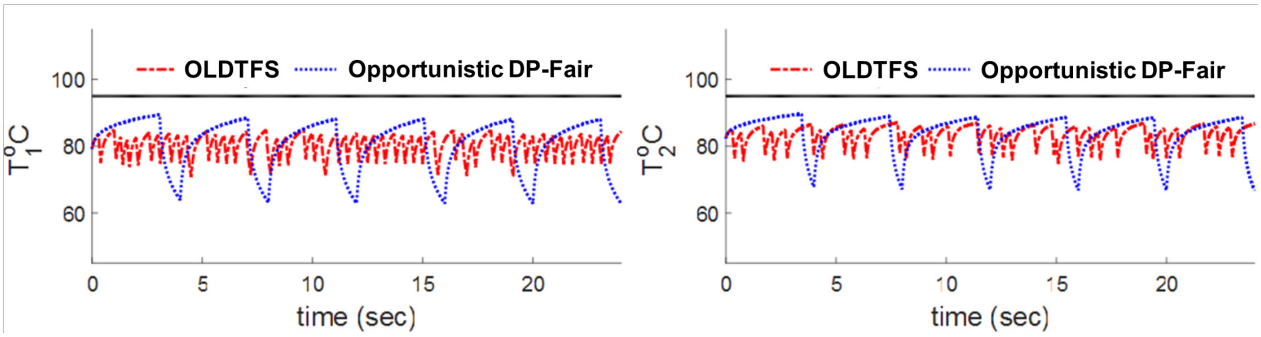


Figure 4.6: Temperature evolution for the example in Sec. 4.4.4 by applying the *OLDTFS* and the opportunistic *DP-Fair* algorithm, which is a blend of the DP-Fair algorithm and the proposed fluid scheduling functions. Both algorithms yields a thermal and temporal feasible schedule, but context switches are higher in *OLDTFS*.

schedule (${}_1FSC_i(\zeta)$, S3), and appears constant (flat) in the plot. When $\zeta > 7$, the controller starts increasing the task execution rate (without never exceeding a 100% CPU utilization): $m_{1,i}^{exec}(\zeta)$ increases continuously and the discretized schedule ($M_{1,i}^{exec}(\zeta)$, red dots) follows up on a quantum basis.

4.5 Conclusions

The approach in Ch. 3 set a solid ground to add and deal with thermal constraints by exploring the balance between fixed (quantum) interval scheduling and variable (deadline based) interval scheduling, including a simple on-off controller. However, establishing an optimal quantum to limit the overhead can be onerous, and the on-line allocation method in Alg. 1 requires heuristics to avoid unnecessary context switches and migrations. The approach taken in this Chapter moves key decisions to the off-line stage, and leverages the controller during the on-line stage to minimize the fluid error even in the presence of disturbances. We also show that a pure deadline partitioning approach can be sufficient to keep the temperature under control, although temperature variations are higher than when relying on the *OLDTFS* on-line scheduler. This scheduling approach has been accepted for publication [82].

Managing aperiodic tasks while honoring the constraints of the HRT task set under thermal restrictions introduces a new level of difficulty, especially in battery-operated o satellite systems in which power draw can be an issue. This problem is addressed in the next chapter.

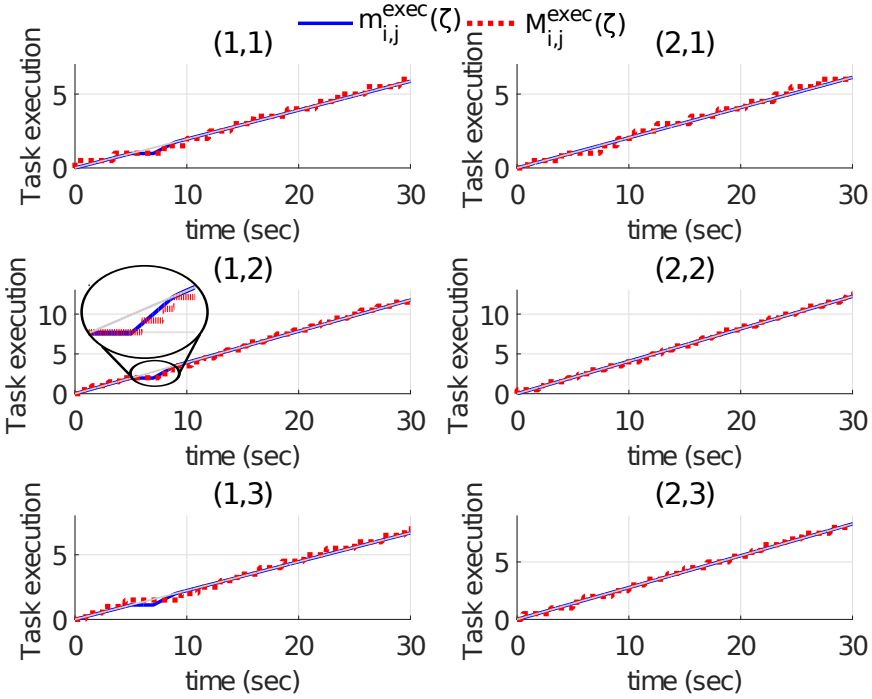


Figure 4.7: Task execution of the example presented in Sec. 4.4.5. A disturbance is introduced during 5-7 s, when the CPU_1 stops task execution due to an external and uncontrollable interruption. Note that when the interruption is released, CPU_1 increases task execution by using the idle spaces; when the fluid schedule function is reached (its reference), then the CPU_1 continues with its normal task execution.

Chapter 5

Thermal, Energy-Aware RT Scheduling with aperiodic task and disturbance management

This Chapter introduces a cascade control scheme to manage the execution of aperiodic tasks arriving asynchronously to a HRT system with thermal constraints in which energy consumption must be minimized, while dealing with disturbances and parametric variations. The proposed controller assumes that there exists a feasible fluid schedule, the aperiodic task parameters are accessible upon task arrival, and that the CPU clock frequency can be throttled.

5.1 Introduction

The two previous chapters introduced two HRT multiprocessor scheduling algorithms designed and tested using a methodology based on TCPNs. Alg. 1 is based on a fluid scheduler and a sliding mode controller to guarantee that temporal constraints are honoured. Alg. 2 goes one step further and is capable of keeping the temperature below a fixed bound while always meeting HRT constraints and managing disturbances. In battery-operated devices, however, the power draw must be minimized to extend the battery lifespan and the autonomy of the device. Also, managing SRT aperiodic tasks is a desirable feature, as long as it warrants the requirements of the HRT task set in the system.

This chapter proposes the integration of a PID controller and an Aperiodic Task Manager (ATM) module into the thermal-aware RT scheduler of Ch. 4. The purpose of this new controller is to manage SRT aperiodic tasks by throttling frequency. This conforms a cascade control in which the PID controler manages frequency based on the *continuous utilization* of the system, whereas the sliding mode controller manages disturbances based on the fluid error as explained in Sec. 4.3.1. Both controllers warrant the fulfillment of the thermal constraint.

The rationale for this chapter was summarized in Sec. 1.7.4, on the intuition that given a minimum frequency which ensures the correct execution of a HRT task set at maximum CPU utilization, any frequency increase (up to a maximum determined by an upper temperature boundary), will lower the total utilization of the system. This will free CPU time as a slack that can be exploited to accommodate aperiodic tasks when they arrive. This intuition was leveraged in [83], where a sizeable computing overhead upon the arrival of an aperiodic task is required. The approach taken in this chapter substantially lowers that computing overhead.

The first section (Sec. 5.2) introduces the principal features of the *METARTS* algorithm, proposed and detailed in [83]. The *METARTS* algorithm provides the minimum frequency required to safely run a HRT task set with thermal constraints, the maximum frequency at which the system can run respecting the temperature limit, and the per-task per-deadline interval CPU cycles each task must run to meet the thermal and HRT constraints. These results from *METARTS* are leveraged in the scheme proposed in this chapter. Sec. 5.3 presents an off-line algorithm that yields a schedule for the HRT task set according to a FPZL policy, later used by the *ATM*. Sec. 5.4 introduces the concepts of *free equivalent system* and *instantaneous system utilization*, the condition to accept or reject an incoming SRT aperiodic task, and the PID controller. Sec. 5.5 presents the *ATM* and the integration of all the control and scheduling elements into the scheduling scheme. Sec. 5.6 shows the simulation results of a proof of concept. Sec. 5.7 closes the chapter with some final conclusions.

5.2 The METARTS algorithm

The *METARTS* scheduler solves the following problem.

Problem 5.2.1 *Given the task set $\mathcal{T}$ and CPU set $\mathcal{P}$, the* Minimum Energy Thermal Aware Real Time Scheduler *(*METARTS*) problem consists in designing an algorithm to allocate the tasks in $\mathcal{T}$ to the m identical CPUs within the hyperperiod H, such that the deadlines for $\mathcal{T}$ are always satisfied and the CPU temperatures are always kept below a given temperature bound $\boldsymbol{T_{max}}$ and the consumed energy is minimum.*

Typically, the clock frequency of modern CPUs is adjusted according to a number of safe values $\mathcal{F} = [F_1, \ldots, F_{max}]$. We normalize this set as $\phi = [\phi_{min} = \frac{F_1}{F_{max}}, \ldots, 1]$.

In *METARTS*, the parameter $\boldsymbol{C_P^{exec}}$ of Eq. (2.3a) is rewritten as $F^3\boldsymbol{C_P'^{exec}}$ to indicate that this parameter depends on the CPU clock frequency. The same steady state analysis for temperature that led to Eq. (4.9), leads now to the following thermal constraint equation:

$$-\boldsymbol{S}\boldsymbol{A}^{-1}F^3\boldsymbol{B}\boldsymbol{w}^{alloc} \leq \boldsymbol{T_{max}} + \boldsymbol{S}\boldsymbol{A}^{-1}\boldsymbol{B'}\boldsymbol{m_a} \tag{5.1}$$

This equation, which depends on the CPU clock frequency, provides the thermal constraints that the allocation of tasks to the processors ($\boldsymbol{w^{alloc}}$) must fulfill. The normalized operation frequency for minimum consumption of energy is:

$$\Phi^* = \max\{\phi_{min}, \frac{1}{m}\sum_{i=1}^{n}\frac{cc_i}{\omega_i F_{max}}\} \tag{5.2}$$

The normalized frequency Φ^* meets the temporal constraints. The actual frequency is computed as:

$$F^* = \min\{F \in \mathcal{F} | F \geq \Phi^* F_{max}\} \tag{5.3}$$

The *maximum thermal frequency* F^+ is the greatest frequency at which all CPUs could operate at 100% of utilization without violating the thermal constraint. F^+ is computed by solving the next programming problem.

$$\begin{array}{l} max \quad F^+ \\ s.t. \\ \quad -\boldsymbol{SA^{-1}F^{+^3}B}\left[\ \frac{CC_1}{F^+H}, \ \ldots\ , \frac{CC_m}{F^+H}\ \right]^T \leq \boldsymbol{T_{max}} + \boldsymbol{SA^{-1}B'm_a} \\ \quad \frac{CC_j}{F^+H} = 1 \qquad \forall j = 1, \ldots, m \\ \quad F^* \leq F^+ \leq F_{max} \end{array} \tag{5.4}$$

The first constraint sets the thermal constraint. CC_j represents the cycles that CPU_j must execute per hyperperiod. Since all CPUs must work at their maximum capacity, the second constraint forces the CPU utilization be 100%. The last constraint bounds F to the actual clock frequency range of CPUs. The solution for F^+ has to be in the set $\mathcal{F}$ of discrete frequencies. Thus the processor frequency is updated as $F^+ = \max\{F \in \mathcal{F} | F \leq F^+\}$.

The algorithm uses the set of deadlines SD and scheduling interval I_{SD}^k presented in Sec. 3.5 to find the number of cycles (x_i^k) that each job of task τ_i must run at the $k-th$ scheduling interval, solving the following LPP:

$$
\begin{aligned}
&\min \quad \sum_{i=1}^{n} x_i^k \quad \forall k = 1, \ldots, \alpha \\
&s.t \\
&\quad \forall k \sum_{i=1}^{n} x_i^k = m * |I_{SD}^k| * F^* \\
&\quad \text{if } r_i = 0 \ \sum_{\gamma=1}^{k} x_i^\gamma = q * cc_i \\
&\quad \text{if } r_i \neq 0 \ \sum_{\gamma=1}^{k} x_i^\gamma \geq -q * cc_i^* + max\{0, \sum_{\gamma=1}^{k} |I_{SD}^\gamma| * F^* - cc_i^*\}
\end{aligned}
\tag{5.5}
$$

The constraints are represented by a unimodular matrix, and therefore the previous problems provide integer solutions [83].

5.3 Off-line job allocation

The clock frequencies F^*, F^+ and the x_i^k cycles computed off-line in Eqs. (5.3-5.5) determine that task τ_i^k must be allocated x_i^k cycles at F^* clock frequency during the I_{SD}^k interval to accomplish the HRT and thermal constraints, and that the frequency could be throttled up to F^+ without violating the thermal constraint. However, this off-line computation does not determine which job must be allocated to which CPU at which time.

The actual allocation is performed in [83] by means of an on-line scheduler, using a fixed-priority Zero-Laxity (FPZL) policy [84]. This allocation policy is simple, and the frequencies and cycles provided by the off-line stage guarantee the fulfillment of HRT and thermal constraints. However, the aperiodic scheduler included in [83] that manages the arrival of aperiodic tasks requires a sizeable amount of on-line computation.

Therefore, we propose to leverage the ability of the TCPN model of tasks and CPUs to obtain an off-line discrete schedule of the HRT task set at minimum frequency (F^*) over the hyper-period. This will allow for a lighter on-line management of aperiodic tasks later.

As in [83], Alg. 3 relies on a FPZL policy, and uses the clock frequencies F^*, F^+ and the x_i^k cycles computed in Eqs. (5.3-5.5), to perform job allocation. Therefore, the schedule meets the required HRT and thermal constraints. However, the schedule is now obtained off-line by simulating the TCPN model of tasks and CPUs (denoted *TCPN**) along the hyper-period. The algorithm yields the values of the allocation function ${}_jW_i(\zeta)$, which is equal to one if the job is allocated (and dispatched) at time ζ, and equal to zero otherwise. The value of this allocation function changes at each scheduling event. Scheduling events are either a zero-laxity event or a job termination event.

Whenever an event occurs, job priorities are updated according to a FPZL policy in a set

ALGORITHM 3: FPZL off-line Scheduler

Input:
I_{SD}^k – *Scheduling (deadline) intervals*;
X^k – *Task cycles per deadline interval*;
F^* – *CPU frequency*;
H – Hyper-period;
$TCPN^*$ – TCPN model of tasks and CPUs (no thermal module);
step – Time Step Simulation
Output: ${}_jW_i(\zeta)$ *Task allocation function (it determines a feasible schedule)*

Initialize
$k = 0$;
$\zeta = 0$
$m_i^{exec}(0) = 0$ The actual executed tasks;
while $\zeta \leq H$ **do**
    Compute the set of job laxities $SL(\zeta)$ as in Eq. (5.6) Compute task priorities sorting $SL(\zeta)$ by descending order ; `// priority= (1,2,3)`
    Allocate the m tasks with highest priority:
    ${}_jW_i(\zeta) = 1$ if τ_i is executed in CPU_j, otherwise ${}_jW_i = 0$;
    Compute the flow ${}_jw_i^{alloc}(\zeta)$ as in Eq. (5.7)
    Compute $\boldsymbol{m_{exec}}(\zeta)$ *by simulating the TCPN* model*
    $\zeta = \zeta + \text{step}$ *;* `// Update time (step is the time step simulation)`
    if $\zeta \geq sd_{k+1}$ **then**
        $k = k + 1$ *;* `// Update scheduling interval` $I_{SD}^k = [sd_k, sd_{k+1}]$
end

SL as follows:

$$SL(\zeta) = \{sl_{\tau_i} | sl_{\tau_i} = \frac{x_i^k}{F^*}\zeta + x_i^k - \frac{x_i^k}{F^*}sd_{k+1} - \frac{m_i^{exec}(\zeta) - m_i^{exec}(sd_k)}{F^*}, x_i^k \in X^k\} \tag{5.6}$$

Jobs reaching their zero-laxity time are given the maximum priority (= 1). Jobs being executed and with laxity different from zero receive priority equal to 2. The remaining jobs receive priority level equal to 3 (the lowest one). Thus, zero laxity tasks have the highest priority and must be executed immediately.

Step 7 in Alg. 3 computes the allocation function ${}_jW_i(\zeta)$. This function is equal to one if the job is allocated and dispatched at time ζ, and equal to zero otherwise. In order to advance the simulation of the *TCPN**, ${}_jw_i^{alloc}(\zeta)$ has to be computed according to the following equation:

$${}_jw_i^{alloc}(\zeta) = \begin{cases} (x_i^k - m_i^{exec}(\zeta))\lambda_i & \text{if } {}_jW_i(\zeta) = 1 \\ 0 & \text{otherwise} \end{cases} \tag{5.7}$$

The last equation represents the flow through the allocation transitions of the *TCPN** model.

5.4 Aperiodic task management based on the free equivalent system

The METARTS algorithm (Sec. 5.2) provides F^* and F^+. The HRT task set can safely run at F^* for minimum power draw, at 100% utilization. In a real case, the fact that $F^* > (\Phi^* \times F_{max})$ can make tasks run faster, leading to a near-optimal utilization, slightly below the theoretical 100% [83]. This utilization slack will never cause thermal or HRT constraint violations, and therefore, in which follows, we assume that the theoretical 100% is always achieved. Throttling the CPU frequency up in the range $[F^* \dots F^+]$ will lower utilization all the more, freeing up CPU capacity (increasing the slack) to allocate aperiodic tasks. In this proposal, we compute the free CPU utilization slack to derive a scheduling algorithm that accepts or rejects SRT aperiodic tasks while complying with the constraints of the HRT task set.

5.4.1 Computing a free equivalent system

Assuming that a feasible schedule for the HRT task set is known, that the system utilization is 100% because the CPUs are running at frequency F^*, and that the maximum frequency at which the system can run honoring the thermal constraints is F^+, then the *maximum free CPU utilization* (U_{max}) represents the maximum workload that the system supports, in addition to the HRT set of tasks, without compromising the temporal and thermal constraints of the system.

$$U_{max} = m - \sum_{\tau_i \in HRT} \frac{cc_i}{F^+ w_i} \tag{5.8}$$

The m CPUs are homogeneous and the schedule for the HRT task set achieves a 100% utilization at F^*, therefore the maximum capacity per CPU is $\frac{U_{max}}{m}$. Hence, the *equivalent free system* is a set of m homogeneous processors, where each processor has a $\frac{U_{max}}{m}$ capacity.

U_{max} is obtained at the beginning of the hyper-period $\zeta = 0$. This value decreases linearly as time increases. Hence, the maximum free CPU utilization at time ζ is:

$$LU_{max}(\zeta) = -\frac{U_{max}}{H}\zeta + U_{max} \tag{5.9}$$

5.4.2 Executed task cycles

In this scenario in which the frequency can be throttled between F^* and F^+ to achieve a maximum CPU utilization, every task in the HRT task set has executed the following amount of cycles at time ζ:

$$cc_i(\zeta) = \int_0^{\zeta} F(\zeta) Ex_i(\zeta) d\zeta \tag{5.10}$$

where $F^* \leq F(\zeta) \leq F^+$ and

$$Ex_i(\zeta) = \begin{cases} 0 & \text{if } \tau_i \text{ is not being executed at time } \zeta \\ 1 & \text{otherwise} \end{cases} \tag{5.11}$$

Hence, the time consumed in the set of CPUs by task τ_i at time ζ is:

$$e_i(\zeta) = \int_0^{\zeta} Ex_i(\zeta)d\zeta \tag{5.12}$$

Then the task utilization of τ_i at time ζ is:

$$u_i(\zeta) = \frac{e_i(\zeta)}{\zeta} \tag{5.13}$$

The scheduler will always throttle the frequency to achieve a system utilization as close as possible to 100% at any time ζ, therefore:

$$U(\zeta) = \sum_{\tau_i \in HRT} u_i(\zeta) = m \tag{5.14}$$

5.4.3 Condition for the acceptance of an aperiodic task

The parameters of an aperiodic task τ_a are known at time arrival ζ_a, and therefore its utilization at time ζ_a is also known:

$$u_a(\zeta) = \frac{cc_a}{\omega_a F(\zeta)} \tag{5.15}$$

Depending on the task utilization and the maximum free CPU utilization at time ζ, τ_a can be accepted or rejected. Formally, if

$$LU_{max}(\zeta) - u_a(\zeta) \geq 0 \tag{5.16}$$

the task is accepted since the system is still able to run τ_a honoring all task deadlines.

5.4.4 Instantaneous system utilization

Assume that the system is operating in the scheduling interval $I_{SD}^k = [sd_k,\ sd_{k+1}]$, where sd_k, sd_{k+1} are the starting and ending time of the interval, and the current time is $sd_k < \zeta < sd_{k+1}$ when an aperiodic task τ_a arrives to the system at time ζ_a. Then, an aperiodic task is accepted or rejected according to Eq. (5.16).

When τ_a is accepted, the frequency must be updated to meet task deadlines and minimize energy. First, we must compute the remaining CPU cycles of each task in the interval I_{SD}^k, at time ζ_a, $sd_k < \zeta_a < sd_{k+1}$:

$$cc_i^R(\zeta_a) = cc_i(sd_{k+1}) - cc_i(\zeta_a) \tag{5.17}$$

Also the remaining time is computed.

$$\zeta^R = sd_{k+1} - \zeta_a \tag{5.18}$$

The remaining time of the aperiodic task is computed as follows.

$$\zeta_a^R = min(sd_{k+1} - \zeta_a, d_a - \zeta_a)$$

In this section we are assuming that task τ_a must execute its cc_a CPU cycles in sd_{k+1}.

Then, the following equation can be solved for $F(\zeta)$:

$$m = \sum_{\tau_i \in A(\zeta<\zeta_a)} \frac{cc_i^R(\zeta)}{\zeta^R F(\zeta)} + \sum_{\tau_a \in A(\zeta=\zeta_a)} \frac{cc_a(sd_{k+1})}{\zeta_a^R F(\zeta)} \tag{5.19}$$

resulting in:

$$F(\zeta) = \sum_{\tau_i \in A(\zeta<\zeta_a)} \frac{cc_i^R(\zeta)}{\zeta^R m} + \sum_{\tau_a \in A(\zeta=\zeta_a)} \frac{cc_a(sd_{k+1})}{\zeta_a^R m} \tag{5.20}$$

where $A(\zeta)$ is the set of tasks that are active at time ζ.

Computing this equation on-line would lead to an inefficient scheduler. The next subsection introduces a control scheme where a PID controller computes $F(\zeta)$ in a simpler way.

5.4.5 A PID control scheme

Fig. 5.1 depicts the architecture of the proposed scheduler. The reference m is the number of CPUs and represents the 100% system capacity. The PID controller measures the utilization error and computes the CPU frequency $F(\zeta)$ making this error equal to zero to achieve 100% of system utilization. The system block includes the characteristics of tasks and CPUs. The scheduler block computes a feasible schedule and generates the task starting and stopping times as outputs. The utilization block receives the task starting and stopping times, and the task parameters from the scheduler and system blocks, respectively, to compute the actual

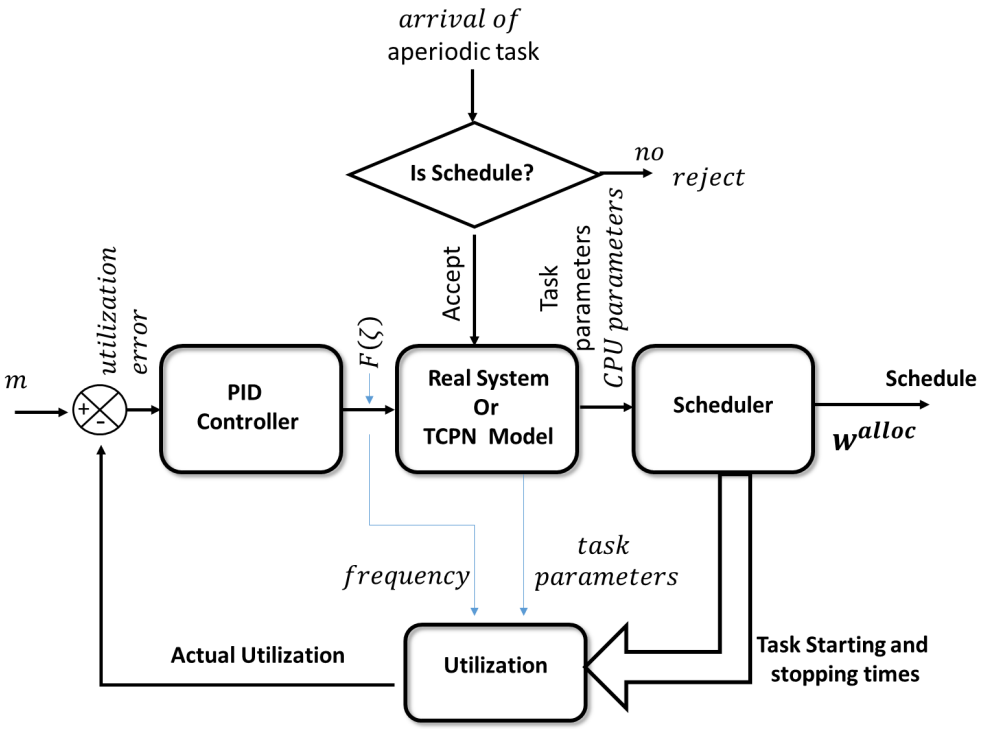


Figure 5.1: Scheduler managing aperiodic tasks. It includes a frequency adaptive mechanism guaranteeing a 100% CPU utilization

system utilization using the left side of Eq. (5.19). Upon arrival of an aperiodic task, the system determines if the task is schedulable or should be rejected. Schedulable tasks are added to the task set, from which they are removed after completion. Next, we first describe the PID controller block in Fig. 5.1, and then the scheduler block.

PID controller

The basic PID controller responds to error by a correction that is an algebraic superposition of three actions (or forces). The first is the proportional action. It changes power in proportion to the value of the error and in the direction that reduces the error. The second is the integral action. It adjusts power incrementally, in proportion to the time integral of previous errors. This action tends to accumulate a slowly-changing bias that becomes constant when the error becomes zero, hence maintaining the system at the zero-error state. The last component is the derivative action. It adjusts power in proportion to the rate of change of error in the direction that reduces the rate of change, damping the response to avoid overshooting. At any time ζ, the output of the controller is the same instant frequency given in Eq. (5.20),

now calculated as the weighted sum of the three terms above:

$$F(\zeta) = KP\left(e(\zeta) + \frac{1}{KI}\int_0^{\zeta} e(\zeta)d\zeta + KD\frac{de(\zeta)}{d\zeta}\right) \tag{5.21}$$

where KP, KI and KD are constants that need to be tuned according to stability analysis. The error $e(\zeta)$ is computed as:

$$e(\zeta) = m - \left(\sum_{\tau_i \in A(\zeta<\zeta_a)} \frac{cc_i^R(\zeta)}{\zeta^R F(\zeta)} + \sum_{\tau_a \in A(\zeta=\zeta_a)} \frac{cc_a(sd_{k+1})}{\zeta_a^R F(\zeta)}\right) \tag{5.22}$$

Eq. (5.21) can be discretized as follows:

$$F(\zeta) \approx KP\left(e(\zeta) + \frac{1}{KI}\sum_{\varrho=1}^{\zeta} e(\varrho\Gamma)\Gamma + KD\frac{e(\zeta) - e(\zeta-\Gamma)}{\Gamma}\right) \tag{5.23}$$

Here, Γ is the discretization period. Then,

$$F(\zeta-\Gamma) \approx KP\left(e(\zeta-\Gamma) + \frac{1}{KI}\sum_{\varrho=1}^{\zeta-\Gamma} e(\varrho\Gamma)\Gamma + KD\left(\frac{e(\zeta-\Gamma) - e(\zeta-2\Gamma)}{\Gamma}\right)\right) \tag{5.24}$$

The combination of the last two equations lead to the following discrete calculation of the PID:

$$F(\zeta) = F(\zeta-\Gamma) + \alpha_1 e(\zeta) + \alpha_2 e(\zeta-\Gamma) + \alpha_3 e(\zeta-2\Gamma) \tag{5.25}$$

This equation provides the same instant frequency $F(\zeta)$ than Eq. (5.20), in a way than can be leveraged in an on-line scehduler. The constants α_i are computed as follows:

$\alpha_1 = KP(1 + \frac{\Gamma}{KI} + \frac{KD}{\Gamma})$, $\alpha_2 = KP(-1 - \frac{2KD}{\Gamma})$, $\alpha_3 = KP\frac{KD}{\Gamma}$.

It is important to remark that the computed frequency is the minimum frequency required to guarantee that no task deadlines are missed, and therefore this scheme is accepting on-line aperiodic tasks for execution while minimizing the power draw.

Scheduler

The scheduler block in Fig. 5.1 can be implemented in different ways. For example, an on-line version of Alg. 3 could replace the simulation of the *TCPN** in line 10 to read task execution

times ($\boldsymbol{m_{exec}}(\zeta)$) from a real system directly. Alternatively, the next section presents a proposal to adapt the scheduling scheme of Ch. 4, which uses a sliding mode controller along with the OLDTFS on-line scheduler. In this way, not only the scheduler is able to manage aperiodic tasks, but also to recover the system from disturbances entailing CPU detentions, or to deal with parametric variations in the system model.

5.5 Aperiodic task management and fluid scheduler

This section introduces a scheduling proposal that integrates the PID controller presented in Sec. 5.4.5 (Fig. 5.1) into the thermal-aware RT scheduler presented in Sec. 4.3.4. The purpose is to leverage the ability of the PID controller to manage aperiodic tasks along with the ability of the thermal-aware RT scheduler to recover from disturbances. The scheduler in Sec. 4.3.4 resorts to a fluid scheduler with a sliding mode controller, which is then discretized with Alg. 2 (Fig. 4.1). Next, we describe the computation of the new fluid functions and the new adapted sliding mode control law. No changes are required in Alg. 2 to obtain the discretized output.

Fluid execution function

The fluid execution function for a task τ_i in CPU_j is computed by integrating the allocation function ${}_jW_i(\zeta)$ obtained in Alg. 3 as follows:

$$ {}_jFSC_{\tau_i}(\zeta) \;=\; \tfrac{F^*}{F_{max}} \int_0^{\zeta} {}_jW_i(\zeta)d\zeta \tag{5.26} $$

where F^* is the minimum frequency that minimizes the energy consumption, ζ is the current time, and $0 \leq \zeta \leq H$. This *fluid* function provides a fluid schedule that fulfills temporal and thermal constraints. The difference of this function with respect to the function that was obtained in Eq. 4.1 is that it represents the execution of a task τ_i in CPU_j as a line with a slope F^*, becoming 0 when the task is not executed in CPU_j. The scheduler will track this function to make all tasks meet the thermal and temporal requirements. As in the scheduler in Ch. 4, a sliding mode controller is in charge of reducing the fluid error (the difference between the actual execution time and the fluid execution time), which can appear upon disturbance. However, the sliding mode control law that was presented in Sec. 4.3.1 now depends on the new fluid task fluid execution ${}_jFSC_{\tau_i}(\zeta)$ (Eq. 5.26). The new form of the control law is presented next.

Control law computation

Following a similar analysis to Proposition 4.2, the new control law ${}_j w_i^{alloc}(\zeta)$ is computed as

$$ {}_j w_i^{alloc}(\zeta) = {}_j \hat{w}_i^{alloc}(\zeta) + \frac{K_1}{\lambda_{i,j}^{exec}} \frac{{}_j FSC_{\tau_i}(H)}{HF(\zeta)} \tag{5.27} $$

where ${}_j \hat{w}_i^{alloc}(\zeta) = K_2 sign(\mathcal{S}_{i,j}(\zeta))$ and $\mathcal{S}_{i,j}(\zeta) = \frac{K_1}{\lambda_{i,j}^{exec}} x_{i,j}^1 + \frac{{}_j FSC_{\tau_i}(H)}{HF(\zeta)\lambda_{i,j}^{exec}} - x_{i,j}^2$. $sign(x) = 1$ if $x \geq 0$; 0 otherwise.

Note that $F(\zeta)$ is the frequency computed by the PID controller in Eq. (5.25).

Discretization by OLDTFS

The vector $\boldsymbol{w}^{\boldsymbol{alloc}}(\zeta)$ represents a fluid schedule, yielding continuous values in vector $\boldsymbol{m}_{\boldsymbol{exec}}(\zeta)$ and generating an infinitesimal number of context switchings and migrations. The Alg. 2 in Sec. 4.3.3 (*OLDTFS*) can be used to discretize the fluid schedule. *OLDTFS* provides a discrete schedule that closely tracks the fluid schedule ($\boldsymbol{m}_{\boldsymbol{exec}}(\zeta)$) by computing a schedule up to the hyperperiod (from time zero to time H).

5.5.1 The Aperiodic Task Manager (*ATM*)

The management of aperiodic tasks in this scheduling system requires two operations: throttling frequency and accept or reject aperiodic tasks. The first operation is performed by the PID controller (Sec. 5.4.5). The second operation is performed by the Aperiodic Task Manager (*ATM*), described in this section. The *ATM* uses Eq. (5.16) to accept and dispatch the SRT aperiodic tasks arriving to the system if they do not jeopardize the constraints of the HRT task set and the thermal limit, or to reject them otherwise. First, we analyze how to compute the zero-laxity of the HRT that is running when a SRT aperiodic task arrives. Then, we will explain the *ATM* algorithm.

Zero-laxity computation

Upon acceptance of an aperiodic task, the PID controller computes a new operating CPU clock frequency F^*. The ATM grants the highest priority to the accepted aperiodic task, which runs either to completion or until a HRT task reaches its zero laxity.

The red dashed line in Fig. 5.2 shows the fluid execution of the HRT task τ_i in CPU_j (${}_j FSC_{\tau_i}(\zeta)$) at the normalized frequency $\phi^* = \frac{F^*}{F_{max}}$. The green solid line represents the

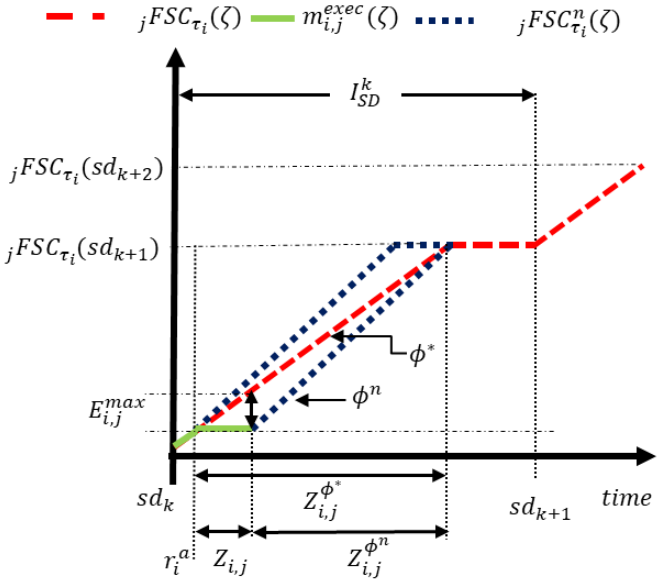


Figure 5.2: Task execution of task τ_i in CPU_j. At time $\zeta = r_i^a$ an aperiodic task τ_i^a arrives to the system.

actual execution $m_{i,j}^{exec}$ of τ_i, and the blue dotted lines represent its fluid execution $_jFSC^n_{\tau_i}(\zeta)$ at any possible normalized frequency $\phi^n = \frac{F(\zeta)}{F_{max}}$ (computed by the PID control).

In the figure, an aperiodic task τ_i^a arrives to the system at $\zeta = r_i^a$ demanding cc_i^a CPU cycles. At this time $\zeta = r_i^a$, task τ_i is running in CPU_j (green solid line). Since τ_i^a is given the highest priority task when it arrives, it is immediatly dipatched, preempting τ_i (flat green line). At $\zeta = r_i^a + Z_{i,j}$, the maximum error $E_{i,j}^{max}$ is equal to $\mathcal{E}_{i,j}(\zeta)$ (where $\mathcal{E}_{T_{i,j}}(\zeta) = {}_jFSC_{\tau_i}(\zeta) - m_{i,j}^{exec}(\zeta)$ as defined in Eq. 4.11). This triggers a zero-laxity event: τ_i must resume execution lest it miss its HRT deadline, and τ_i^a must be preempted accordingly.

The challenge at this point is to calculate $E_{i,j}^{max}$ upon arrival of τ_i^a. This can be done with the known signals ($_jFSC_{\tau_i}$ and $m_{i,j}^{exec}(\zeta)$). Analyzing Fig. 5.2 we can derive the following equation:

$$Z_{i,j} = Z_{i,j}^{\phi^*} - Z_{i,j}^{\phi^n} \tag{5.28}$$

where $Z_{i,j}^{\phi} = \frac{{}_jFSC^*_{\tau_i}(sd_{k+1}) - m_{i,j}^{exec}(\zeta)}{\phi*}$ and $Z_{i,j}^{\phi^n} = \frac{{}_jFSC^n_{\tau_i}(sd_k) - m_{i,j}^{exec}(\zeta)}{\phi^n}$.

Then, the maximum error between the fluid execution function $_jFSC_{\tau_i}(\zeta)$ and the actual execution $m_{i,j}^{exec}(\zeta)$ at time $\zeta = r_i^a$ is computed as:

ALGORITHM 4: *ATM*

Input: parameters of τ_i^a: (r_i^a, cc_i^a, d_i^a), $\mathcal{E}_{i,j}(\zeta)$
Output: Aperiodic task execution

```
1  Initialize E_{i,j}^{max} = 0;
2  while true do
3      if ζ = r_i^a and LU_max(ζ) − u_a(ζ) ≥ 0 then
4          Preempt task in CPU_j with the lowest priority;
5          Accept and dispatch aperiodic task;
6          Compute E_{i,j}^{max} using Eq. (5.29) ;        // Maximum error allowed for task τ_i^a
7      else
8          Reject aperiodic task;
9      end
10     if E_{i,j}(ζ) − E_{i,j}^{max} == 0 then
11         preempt τ_i^a, and continue with the normal behaviour of τ_i;
12     end
13 end
```

$$E_{i,j}^{max} = \phi^* Z_{i,j} \tag{5.29}$$

The ATM algorithm

Alg. 4 shows the Aperiodic Task Manager (*ATM*). A scheduling event occurs when an aperiodic task arrives and when a HRT task reaches its zero laxity (i.e., when $\mathcal{E}_{i,j} - E_{i,j}^{max} = 0$). If the aperiodic task is accepted (step 3), the HRT task τ_i that is currently running in CPU_j is preempted. The *ATM* computes the maximum execution error $E_{i,j}^{max}$ that will trigger the zero-laxity event ($\mathcal{E}_{i,j} - E_{i,j}^{max} = 0$). The zero-laxity event preempts the aperiodic task τ_i^a and resumes τ_i at the adjusted CPU frequency $F(\zeta)$.

5.5.2 Overview of the complete scheduler

Fig. 5.3 provides an overview of the complete scheduling approach. An off-line stage computes the fluid functions ${}_jFSC_{\tau_i}(\zeta)$ (signal vector S1). The on-line scheduler holds the same fluid scheduler and the discretized scheduler (*OLDTFS*) used in Ch. 4 (see Fig. 4.1), which work the same way here. Differences are that the fluid functions computed off-line are now given as a function of $F(\zeta)$ (Eq. 5.26), that the frequency is now dynamic, throttled by the PID controller (Eq. 5.25), and that a new module, the *ATM*, rejects or accepts incoming SRT aperiodic tasks.

The upper box A in the figure shows the behavior of the system during normal operation. There is no frequency switch because there is no incoming aperiodic task (or if any, the

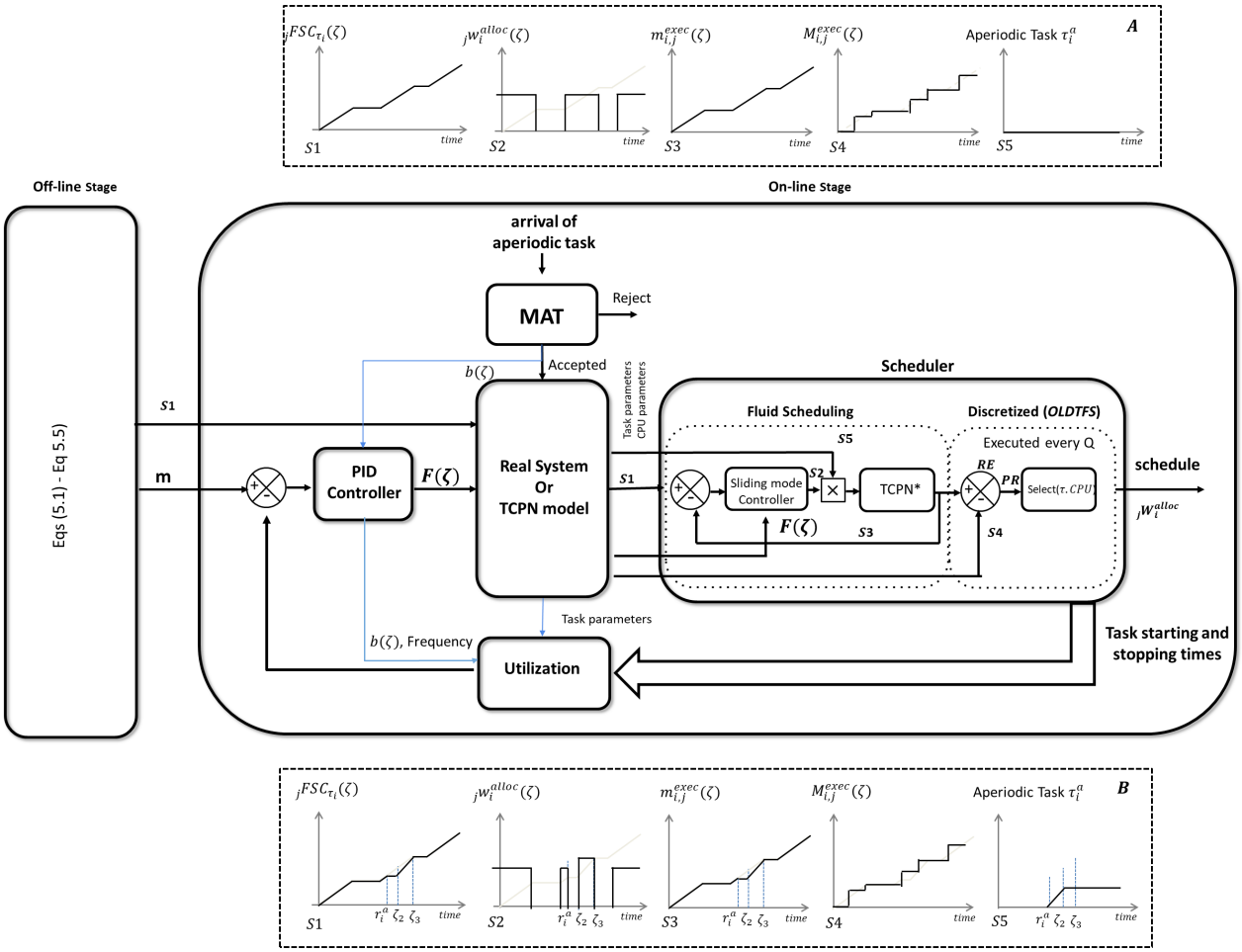


Figure 5.3: Frequency control OLDTFS overview.

aperiodic task has been rejected by the ATM). S3 (the execution time of τ_i) as given by the *TCPN** model (the internal model of tasks and CPUs in the scheduler) matches the fluid scheduler, as it happened in the thermal-aware RT scheduler of Ch. 4 during normal operation, no disturbances (Fig. 4.1, box A).

The lower box A in Fig. 5.3 depicts the behavior of the system upon arrival —and acceptance, according to Eq. (5.16)— of a SRT aperiodic task (τ_i^a) at time $\zeta_a = r_i^a$, demanding cc_i^a cycles, with utilization u_a. Accepting τ_i^a implies that a HRT task τ_i in CPU_j is preempted, and τ_i^a is immediately dispatched. To accept τ_i^a, the *ATM* (Eq. 5.16) takes into account that the execution of the aperiodic task fits into the schedule because the frequency can be safely throttled up —honoring the thermal constraint— so that the preempted HRT task can meet its deadline once it resumes execution.

At time ζ_2, either τ_i^a ends or is preempted because τ_i reaches its zero-laxity condition. At this point, the PID controller computes a new convenient frequency $F(\zeta)$ according to Eq. (5.25). The increase in frequency is can be seen in the variation of S2 in Fig. 5.3, box B. At this new (higher) frequency, τ_i resumes execution in CPU_j. The slope of S1 in Box B ($_jFSC_{\tau_i}$) increases from ζ_2 to ζ_3 with respect to the slope of this function in box A, meaning that τ_i is running faster, until catching up the fluid slope expected when running at the normal (minimum) frequency (F^*).

The scenario depicted in Fig. 5.3, box B, does only show the arrival —an acceptance— of an aperiodic task. In contrast with the thermal-aware controller in Fig. 4.1, there is now a cascade control composed of the PID controller (frequency) and the sliding mode controller (still in charge of dealing with disturbances or parametric variations). Therefore, we could still add different scenarios with an incoming aperiodic task (the situation isolated in Fig. 5.3, box B) and disturbances (the situation depicted in Fig. 4.1, box B).

5.6 Simulation Results

A proof of concept shows in this section how the proposed scheduler deals with aperiodic tasks. Let us consider the HRT task set $\mathcal{T} = \{\tau_1, \tau_2, \tau_3\}$, where $\tau_1 = (2000, 4)$, $\tau_2 = (5000, 8)$, $\tau_3 = (6000, 12)$; the hyperperiod is $H = 24$. These tasks run on two homogeneous $1cm \times 1cm$ silicon microprocessors mounted over a $5cm \times 5cm$ cooper heat spreader, where the thickness of the silicon microprocessors and the cooper heat spreader is $0.5mm$ and $1mm$ respectively. The isotropic thermal properties of the materials are taken from [86]. The processor supports four operating frequency levels $\mathcal{F} = \{0.5, 0.85, 0.95, 1\}KHz$. The temperature of the surrounding air is constant and set to 45^oC. The maximum operating temperature level is set to $T_{max_{1,2}} = 50^oC$. The simulations presented in this dissertation consider CPUs with caches and speculative mechanisms non-existent or turned off.

In the offline stage, the *METARTS* algorithm obtains the minimum frequency for the HRT task as $\phi^* = 0.8125$ according to Eq. (5.2). Hence, the selected discrete frequency is $F^* = 0.85kHz$ (Eq. 5.3). Eq. (5.4) provides the maximum clock frequency ($F^+ = 1kHz$), so that the *METARTS* problem has a feasible solution. Then, solving the LPP in Eq. (5.5) for F^* yields the CPU cycles of each task x_i^k to be executed at each interval. Alg. 3 determines which jobs must be allocated to which CPU (i.e. the values of the allocation function $jW_i(\zeta)$). The fluid execution function for a task τ_i in CPU_j is computed by using Eq. (5.26). This function provides the fluid schedule at the discrete frequency F^* that fulfills temporal and thermal constraints. Moreover, the fluid schedule is used as a set point in the online stage.

Fig. 5.4 depicts the outcome of the online scheduler proposed in Sec. 5.5 when an aperiodic task $\tau_1^a = (2000, 10)$ arrives at $\zeta = 2$ with an absolute deadline at $\zeta = 12$. The *ATM* deals with this incoming aperiodic task, accepting τ_1^a at $\zeta = 2$. At the interval $[2, 8]$ the PID

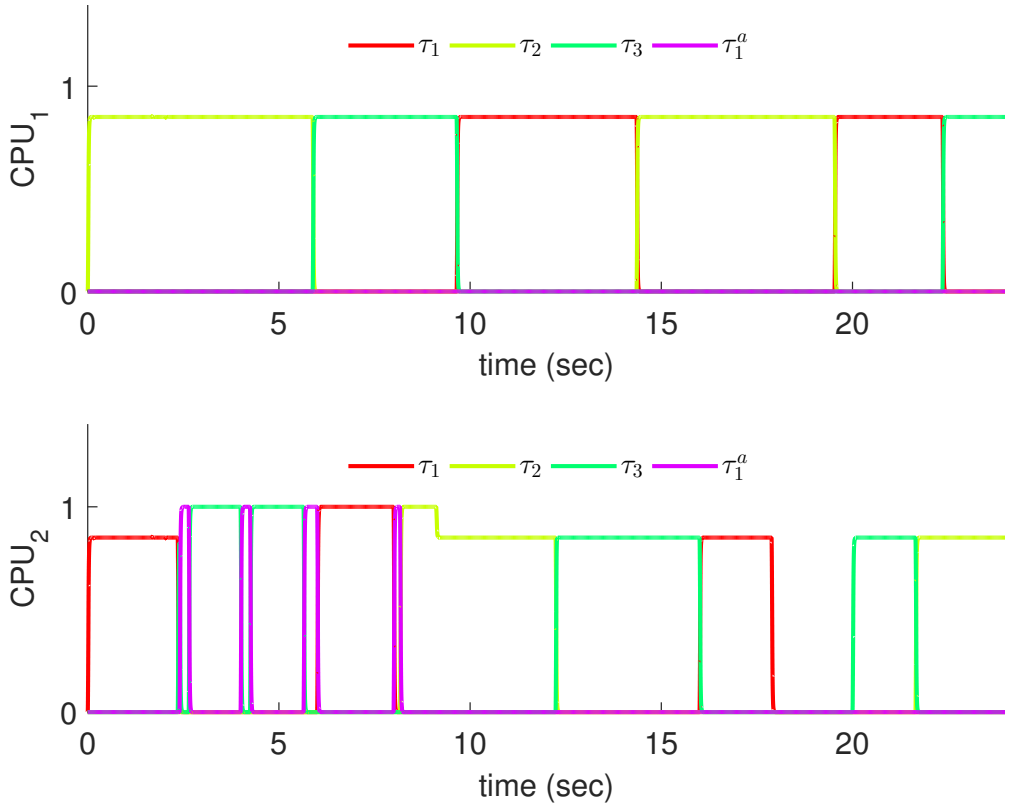


Figure 5.4: Task execution of the example. The y-axis shows the utilization of the CPU.

controller throttles the frequency up to $F(\zeta) = 1Khz$, τ_1^a is dispatched and runs in CPU_2 during this interval. However, the fact that $F^* > (\Phi^* \times F_{max})$ makes tasks allocated to CPU_2 run faster in this simulation. This translates into the slack that appears in the lowest plot of the figure (interval $[18, 20]$).

5.7 Conclusions

The scheduling scheme introduced in Ch. 4 can be easily empowered by incorporating the ability to manage SRT aperiodic tasks while honoring the thermal constraints of the system and the constraints of the HRT task set.

A cascade control is a straightforward solution that requires minimal changes in the scheme, leveraging the results provided by the *METARTS* algorithm [83]. The cascade control is composed of a new PID controller along with the sliding mode controller already present in the scheduling scheme of Ch. 4.

The PID controller computes the CPU frequency required to speed up HRT taks resuming execution upon acceptance of a SRT aperiodic tasks in the range $[F^*, \ldots, F^+]$ in order to free up CPU capacity and allocate aperiodic tasks. This approach avoids the recalculation upon arrival of aperiodic tasks which appears in [83]. The sliding mode controller is still able to deal with disturbances and parametric variations. The result is a resilient RT multiprocessor scheduler able to minimize energy consumption and guarantee thermal and temporal constraints.

Chapter 6

Conclusions

6.1 Summary

6.1.1 TCPNs as a modelling tool for RT scheduling

A first objective in the Thesis has been to explore the advantages and possible drawbacks of using TCPNs to model RT task sets running on a multiprocessor, accounting for thermal and energy constraints. Although Petri nets have been leveraged for decades in different fields for scheduling purposes including RT restrictions, as far as we know this is the first work based on TCPNs specifically addressed to RT scheduling on multiprocessors.

As a hypothesis, the continuous nature of the thermal problem and RT multiprocessor schedulers based on global, fluid approaches, could well be modeled by means of a TCPN. On the other hand, multiprocessors provide as much power as design complexity, and the modular nature of TCPNs could also help at this point. Also, the experience of using control techniques along with TCPNs in other fields could be leveraged in the RT multiprocessor scheduling arena.

As a proof of concept, results in this Thesis are promising in that we have found a suitable methodology, automated with a publicly available tool, to design and test RT multiprocessor schedulers encompassing thermal and energy aspects. Other than making the design process easier (gathering task, CPU and thermal behavior in a single formalism), a simplified TCPN model of tasks and CPUs (*TCPN**, without thermal components) requires few calculations to solve the state equation, and constitutes a feasible tool to derive the fluid error and leverage on-off controllers for a better system's resilience.

The simulation framework implemented to support the experimental parts of this Thesis constitutes the basis of a more ambitious joint project that is currently under development [80].

6.1.2 Control integration

The problem considered in this Thesis involves controlling three different variables: time, temperature and energy (the latter by operating on CPU frequency), subject to specific constraints. Moreover, asynchronous unexpected events such as incoming aperiodic tasks or disturbances are considered. From a bare control theory point of view, this constitutes a complex problem, which can not be solved with a single controller. Fortunately, there are solid tools to bring this problem down to a simpler control problem, amenable to be tackled by means of feedback controllers. Feedback controllers can deal with a wide range of behaviors related to RT tasks with thermal and energy constraints.

A first tool to face this problem is the global TCPN model obtained in Ch. 2, which leads to a state space equation that allows controlling the workload execution, the temperature and the energy consumption of the system.

A second tool comes from the RT scheduling theory. By the reasons discussed in Secs. 1.6 and 1.7.2, global fluid schedulers constitute a first choice, because they are amenable to be modeled with a TCPN, and especially because they are HRT and SRT optimal. Fluid schedulers trigger an infinitesimal number of task preemptions and migrations and must be discretized. Thus, the objective in Ch. 3 is twofold. First, to analyze the compromise between a quantum-based and a deadline interval-based discretization. Second, to use a controller to bring the fluid error down to zero with a low computing overhead. This led to exploiting a continuous sliding mode controller in the scheduler, which fulfills the HRT requirements.

The inclusion of temperature in Ch. 4 as an additional variable to control required a different approach. The solution to Eq. (4.10) represents the fraction of each job to be run at each CPU during the hyperperiod to meet all thermal, time and utilization constraints. That means that no controller is required to fulfill system requirements during normal operation. This opens up the chance to leverage a controller to improve on-line resilience. Therefore, the sliding mode controller that was successfully used in Ch. 3 to close the fluid error, is now targeted to the dynamic management of disturbances at fixed CPU frequency as long as the system keeps under maximum utilization (Fig. 4.1).

Dealing with aperiodic tasks introduces a new complexity level. The solution to the *METARTS* problem [83] provides the minimum frequency (F^*) at with the system can run to accomplish the requirements a the HRT task set. It also provides the maximum frequency at which the system can run (F^+) and the job cycles per deadline interval to meet thermal and HRT constraints.

Upon these results, Ch. 5 leverages the ability to throttle the current frequency $F(\zeta)$ to decrease the instantaneous utilization as per the HRT task set, freeing up a slack to accommodate incoming aperiodic tasks up to the limit imposed by F^+. The problem is that calculating the new $F(\zeta)$ upon arrival of an aperiodic task can be onerous. However, it

boils down to a simple linear computation if derived from the PID control law presented in Sec. 5.4.5. Leveraging the scheduling scheme of Ch. 4 leads to a cascade control where the PID controller manages $F(\zeta)$ and the sliding mode controller deals with disturbances and parametric variations.

6.1.3 Resilient, energy-efficient RT scheduling

This Thesis has followed an incremental approach, starting with a RT global scheduler with control integration (Ch. 3) which results were presented in [81]. Lessons learned are that the discretization of fluid schedulers leads to difficult compromises, and on the bright side, that TCPNs a really a powerful modeling tool in this context. Over the basis of this scheduler, Ch. 4 includes a thermal constraint and leverages the sliding mode controller to deal with disturbances and parametric variations inside specific utilization limits. This scheduler has been accepted for publication [82].

The results draw a few important conclusions. First, computing off-line the per-task, per-CPU time share leads to a lighter on-line discretized scheduler. Second, ensuring thermal, time and utilization constraints by solving off-line an LPP, allows the use of a simple on-off controller to deal with disturbances and parametric variations, thus improving the resilience of the system. Third, the fluid functions (which are computed off-line) can be tracked on-line by solving the state equation of a simple TCPN model of tasks and CPUs ($TCPN^*$), according to an allocation vector managed by the on-off controller to minimize the fluid error upon disturbance. Last, that the on-off control can lead to an unbounded number of context switches and migrations.

The final step consists on considering the management of SRT aperiodic tasks and disturbances while observing HRT and thermal constraints, which constitutes a complex problem. The approach taken in [83] ensured the safe execution of an HRT task set with minimum power draw, and the ability of throttling up CPU frequency up to a maximum F^+ that still keeps the system under the maximum allowed temperature. Per-task, per-deadline interval cycles computed off-line as time-dependent functions allowed for an on-line scheduler with a low overhead. However, recomputing these functions upon the arrival of an aperiodic task requires a tough calculation.

Thus, Ch. 5 starts from the scheduling scheme of Ch. 4 and leverages a simple PID controller to compute the frequency required to safely run a HRT job which was preempted upon acceptance of an incoming aperiodic task, profiting from the results obtained by solving the *METARTS* problem in [83]. The PID controller and the sliding mode controller conforms a cascade control that endows the system with a higher resilience with a low computing overhead.

As a bonus, the Chapter also introduces the idea of performing task allocation off-line

(Alg. 3) using a FPZL policy over the per-task, per-deadline interval cycles provided by the solution to the *METARTS* problem for the HRT task set. In other words, starting with a global fluid scheduling algorithm, it is possible to obtain a partition which is HRT optimal (the optimality is ensured by the HRT optimality of the fluid schedule), thermal-safe, and with takes a minimum power draw. Being an off-line partition, off-line heuristics can be applied to minimize context switches and migrations.

6.2 Open questions and Future Work

The assumptions taken in this work are commonplace in RT scheduling (Sec. 1.4.1), and provides a convenient starting point when designing and testing multiprocessor schedulers. On the other hand, as mentioned in Sec. 1.6, the strict safety levels demanded by the automotive and aerospace industries impose scheduling models strongly deterministic at all levels, assuming a loss of optimality in resource usage, such as partitioned scheduling locally managed by a cyclic executive.

However, there is a growing interest in matching real-world needs and advances in RT scheduling theory as the industry progresses in embracing multiprocessor for RT systems. Reasons are that real systems are facing more and more the problem of hardware overprovisioning, along with problems derived from contention in memory accesses which are difficult to bound, among others. It has already been claimed that RT algorithms other than the cyclic executive and schedulability analysis beyond RMS or DMS are perfectly viable, although not even EDF (first proposed in 1974 [92]) is being implemented in certified RTOS [10].

Closing the gap between industry and RT research can be done in two ways. On the one hand, by exploiting theoretical models and results with an eye on real-world constraints. On the other hand, by directly starting with the paradigms and workloads used in the industry.

The first way can be undertaken from this Thesis perspective by leveraging Alg. 3 for example. It will be worth exploring heuristics to reduce context switching an migrations over a schedule obtained off-line, which warrants thermal, time and utilization constraints.

Concerning the second way, the modeling power of TCPNs could well be tried to tackle the contention problem which appears when scheduling the Minor Frame (MIF) in a cyclic executives [93].

Other than that, the basis established in this Thesis has given foot to several on-going projects. One of them is the already mentioned simulation framework [80]. Another important one is the implementation of the schedulers in LitmusRT [10], running on a platform instrumented for thermal and power measuring. Along with the focus on the reduction of the overhead especially when applying the on-off control, testing on-the field the capabilities of the schedulers will provide interesting results.

Bibliography

[1] S. Abinesh, M. Kathiresh, and R. Neelavenik, "Analysis of multi-core architecture for automotive applications," in *2014 International Conference on Embedded Systems (ICES)*, pp. 76–79, July 2014.

[2] K. Vipin, "Cannoc: An open-source noc architecture for ecu consolidation," in *2018 IEEE 61st International Midwest Symposium on Circuits and Systems (MWSCAS)*, pp. 940–943, Aug 2018.

[3] M. K. Berhe, "Ergonomic temperature limits for handheld electronic devices," in *ASME 2007 InterPACK Conference collocated with the ASME/JSME 2007 Thermal Engineering Heat Transfer Summer Conference*, ASME, 2007.

[4] 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 Dynamic Systems*, vol. 21(4), pp. 427–497, December 2011.

[5] R. David and H. Alla, "Discrete, continuous and hybrid Petri nets (david, r. and alla, h.; 2004)," *Control Systems, IEEE*, vol. 28, pp. 81–84, June 2008.

[6] IEEE Technical Committee on Real-Time Systems, "Terminolgy and notation," 2019. `http://sites.ieee.org/tcrts/education/terminology-and-notation/`.

[7] G. Buttazzo, *Hard real-time computing systems: predictable scheduling algorithms and applications*, vol. 24. Springer Science & Business Media, 2011.

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

[9] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM (JACM)*, vol. 20, no. 1, pp. 46–61, 1973.

[10] B. B. Brandenburg, *Scheduling and Locking in Multiprocessor Real-time Operating Systems*. PhD thesis, University of North Carolina at Chapel Hill, Chapel Hill, NC, USA, 2011. AAI3502550.

[11] J. L. Hennessy and D. A. Patterson, *Computer Architecture, Fifth Edition: A Quantitative Approach*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 5th ed., 2011.

[12] A. Burns and A. Wellings, *Real-Time Systems and Programming Languages: Ada, Real-Time Java and C/Real-Time POSIX*. Addison-Wesley Educational Publishers Inc, 2009.

[13] B. Dinechin, D. van Amstel, M. Poulhies, and G. Lager, "Time-critical computing on a single-chip massively parallel processor," in *Proceedings -Design, Automation and Test in Europe*, pp. 1–6, 03 2014.

[14] R. Trüb, G. Giannopoulou, A. Tretter, and L. Thiele, "Implementation of partitioned mixed-criticality scheduling on a multi-core platform," *ACM Trans. Embed. Comput. Syst.*, vol. 16, pp. 122:1–122:21, Sept. 2017.

[15] Aeroflex Gaisler, "Quad core leon4 sparc v8 processor leon4 ngmp draft data sheet and users manual," 2011.

[16] M. Pinedo, "Scheduling: Theory, and systems," 2008.

[17] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM (JACM)*, vol. 20, no. 1, pp. 46–61, 1973.

[18] A. Silberschatz, P. B. Galvin, and G. Gagne, *Operating System Concepts*. Wiley Publishing, 9th ed., 2012.

[19] A. W. Alan Burns, *Sistemas de tiempo real y lenguajes de programación - 3^a edición*. Pearson Educacion, 2005.

[20] M. R. Garey and D. S. Johnson, *Computers and intractability*. New York: W.H. Freemanand Company, 1979.

[21] J. Carpenter, S. Funk, P. Holman, A. Srinivasan, J. Anderson, and S. Baruah, "A categorization of real-time multiprocessor scheduling problems and algorithms," in *HANDBOOK ON SCHEDULING ALGORITHMS, METHODS, AND MODELS*, Chapman Hall/CRC, Boca, 2004.

[22] W. A. Horn, "Some simple scheduling algorithms," *Naval Research Logistics (NRL)*, vol. 21, no. 1, pp. 177–185, 1974.

[23] A. Srinivasan, *Efficient and Flexible Fair Scheduling of Real-time Tasks on Multiprocessors*. PhD thesis, The University of North Carolina at Chapel Hill, 2003. AAI3112080.

[24] 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 Systems*, vol. 47, no. 5, pp. 389–429, 2011.

[25] J. Lee, A. Easwaran, and I. Shin, "Laxity dynamics and llf schedulability analysis on multiprocessor platforms," *Real-Time Syst.*, vol. 48, pp. 716–749, Nov. 2012.

[26] J. Lee, A. Easwaran, I. Shin, and I. Lee, "Zero-laxity based real-time multiprocessor scheduling," *J. Syst. Softw.*, vol. 84, pp. 2324–2333, Dec. 2011.

[27] T. P. Baker, "A comparison of global and partitioned edf schedulability tests for multiprocessors," in *In International Conf. on Real-Time and Network Systems*, Citeseer, 2005.

[28] R. I. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems," *ACM computing surveys (CSUR)*, vol. 43, no. 4, p. 35, 2011.

[29] J. M. Calandrino, J. H. Anderson, and D. P. Baumberger, "A hybrid real-time scheduling approach for large-scale multicore platforms," in *Proceedings of the 19th Euromicro Conference on Real-Time Systems*, ECRTS '07, (Washington, DC, USA), pp. 247–258, IEEE Computer Society, 2007.

[30] D.-I. Oh and T. P. Bakker, "Utilization bounds for n-processor rate monotone scheduling with static processor assignments," *Real-Time Systems*, vol. 15, no. 2, pp. 183–192, 1998.

[31] M. Bertogna and M. Cirinei, "Response-time analysis for globally scheduled symmetric multiprocessor platforms," in *Real-Time Systems Symposium, 2007. RTSS 2007. 28th IEEE International*, pp. 149–160, IEEE, 2007.

[32] J. L. Goossens and S. Funk, "Priority-driven scheduling of periodic task systems on multiprocessors," *Real-Time Systems*, pp. 2–3, 2003.

[33] M. Bertogna, M. Cirinei, and G. Lipari, "Improved schedulability analysis of edf on multiprocessor platforms," in *Proceedings of the 17th Euromicro Conference on Real-Time Systems*, ECRTS '05, (Washington, DC, USA), pp. 209–218, IEEE Computer Society, 2005.

[34] S. Kato and N. Yamasaki, "Real-time scheduling with task splitting on multiprocessors," in *Proceedings of the 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, RTCSA '07, (Washington, DC, USA), pp. 441–450, IEEE Computer Society, 2007.

[35] J. H. Anderson and A. Srinivasan, "Mixed pfair/erfair scheduling of asynchronous periodic tasks," in *Real-Time Systems, 13th Euromicro Conference on, 2001.*, pp. 76–85, IEEE, 2001.

[36] H. Cho, B. Ravindran, and E. D. Jensen, "An optimal real-time scheduling algorithm for multiprocessors," in *Real-Time Systems Symposium, 2006. RTSS'06. 27th IEEE International*, pp. 101–110, IEEE, 2006.

[37] A. Chandra, M. Adler, and P. Shenoy, "Deadline fair scheduling: bridging the theory and practice of proportionate pair scheduling in multiprocessor systems," in *Real-Time Technology and Applications Symposium, 2001. Proceedings. Seventh IEEE*, pp. 3–14, IEEE, 2001.

[38] D. Zhu, D. Mossé, and R. Melhem, "Multiple-resource periodic scheduling problem: how much fairness is necessary?," in *Real-Time Systems Symposium, 2003. RTSS 2003. 24th IEEE*, pp. 142–151, IEEE, 2003.

[39] 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, 1996.

[40] S. K. Baruah, J. E. Gehrke, and C. G. Plaxton, "Fast scheduling of periodic tasks on multiple resources," in *ipps*, p. 280, IEEE, 1995.

[41] N. Fisher, J. Goossens, and S. Baruah, "Optimal online multiprocessor scheduling of sporadic real-time tasks is impossible," *Real-Time Syst.*, vol. 45, pp. 26–71, June 2010.

[42] G. Nelissen, V. Berten, J. Goossens, and D. Milojevic, "Reducing preemptions and migrations in real-time multiprocessor scheduling algorithms by releasing the fairness," in *2011 IEEE 17th International Conference on Embedded and Real-Time Computing Systems and Applications*, vol. 1, pp. 15–24, Aug 2011.

[43] S. K. Lee, "On-line multiprocessor scheduling algorithms for real-time tasks," in *TENCON'94. IEEE Region 10's Ninth Annual International Conference. Theme: Frontiers of Computer Technology. Proceedings of 1994*, pp. 607–611, IEEE, 1994.

[44] 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 Systems*, vol. 47, no. 5, p. 389, 2011.

[45] A. P. Chandrakasan, S. Sheng, and R. W. Brodersen, "Low-power cmos digital design," *IEICE Transactions on Electronics*, vol. 75, no. 4, pp. 371–382, 1992.

[46] W. Hai, J. Wan, S. Tan, C. Zhang, H. Tang, Y. Yuan, K. Huang, and Z. Zhang, "A fast leakage-aware full-chip transient thermal estimation method," *IEEE Transactions on Computers*, 2018.

[47] W. Nebel and D. Helms, "On leakage currents. sources and reduction for transistors, gates, memories and digital systems," in *Tutorial at the ACM/IEEE international symposium on Low power electronics and design*, 2013.

[48] Y. Liu, R. P. Dick, L. Shang, and H. Yang, "Accurate temperature-dependent integrated circuit leakage power estimation is easy," in *2007 Design, Automation Test in Europe Conference Exhibition*, pp. 1–6, April 2007.

[49] M. B. Taylor and S. Swanson, "Dark silicon [guest editors' introduction]," *IEEE Micro*, vol. 33, pp. 6–7, Sept 2013.

[50] D. J. Jamieson, A. D. Mansell, J. A. Staniforth, and D. W. Tebb, "Application of finite difference techniques for the thermal modelling of power electronic switching devices," in *Power Electronics and Variable-Speed Drives, 1994. Fifth International Conference on*, pp. 313–318, Oct 1994.

[51] S. Rao, *The finite element method in engineering*. Butterworth-Heinemann, Elsevier, 2011.

[52] S. Liu, J. Zhang, Q. Wu, and Q. Qiu, "Thermal-aware job allocation and scheduling for three dimensional chip multiprocessor," in *Quality Electronic Design (ISQED), 2010 11th International Symposium on*, pp. 390–398, March 2010.

[53] S. Zhang and K. Chatha, "Approximation algorithm for the temperature-aware scheduling problem," in *Computer-Aided Design, 2007. ICCAD 2007. IEEE/ACM International Conference on*, pp. 281–288, Nov 2007.

[54] T. Chantem, X. Hu, and R. Dick, "Temperature-aware scheduling and assignment for hard real-time applications on MPSoCs," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 19, pp. 1884–1897, Oct 2011.

[55] K. Skadron, M. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan, "Temperature-aware microarchitecture," in *Computer Architecture, 2003. Proceedings. 30th Annual International Symposium on*, pp. 2–13, June 2003.

[56] W. Huang, S. Ghosh, S. Velusamy, K. Sankaranarayanan, K. Skadron, and M. R. Stan, "Hotspot: A compact thermal modeling methodology for early-stage VLSI design," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 14, no. 5, pp. 501–513, 2006.

[57] J. Kong, S. W. Chung, and K. Skadron, "Recent thermal management techniques for microprocessors," *ACM Computing Surveys*, vol. 44, no. 3, pp. 13:1–13:42, 2014.

[58] S. Murali, A. Mutapcic, D. Atienza, R. Gupta, S. Boyd, L. Benini, and G. De Micheli, "Temperature control of high-performance multi-core platforms using convex optimization," in *Design, Automation and Test in Europe*, pp. 110–115, 2008.

[59] J.-J. Chen, C.-M. Hung, and T.-W. Kuo, "On the minimization fo the instantaneous temperature for periodic real-time tasks," in *13th IEEE Real Time and Embedded Technology and Applications Symposium (RTAS'07)*, pp. 236–248, IEEE, 2007.

[60] L. Schor, I. Bacivarov, H. Yang, and L. Thiele, "Worst-case temperature guarantees for real-time applications on multi-core systems," in *2012 IEEE 18th Real Time and Embedded Technology and Applications Symposium*, pp. 87–96, IEEE, 2012.

[61] Y. Fu, N. Kottenstette, Y. Chen, C. Lu, X. D. Koutsoukos, and H. Wang, "Feedback thermal control for real-time systems," in *2010 16th IEEE Real-Time and Embedded Technology and Applications Symposium*, pp. 111–120, IEEE, 2010.

[62] J. Donald and M. Martonosi, "Techniques for multicore thermal management: Classification and new exploration," in *ACM SIGARCH Computer Architecture News*, vol. 34, pp. 78–88, IEEE Computer Society, 2006.

[63] F. Zanini, D. Atienza, and G. De Micheli, "A control theory approach for thermal balancing of mpsoc," in *2009 Asia and South Pacific Design Automation Conference*, pp. 37–42, IEEE, 2009.

[64] X. Fu, X. Wang, and E. Puster, "Dynamic thermal and timeliness guarantees for distributed real-time embedded systems," in *2009 15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pp. 403–412, IEEE, 2009.

[65] Y. Fu, N. Kottenstette, C. Lu, and X. D. Koutsoukos, "Feedback thermal control of real-time systems on multicore processors," in *Proceedings of the tenth ACM international conference on Embedded software*, pp. 113–122, ACM, 2012.

[66] A. Soria-Lopez, P. Mejia-Alvarez, and J. Cornejo, "Feedback scheduling of power-aware soft real-time tasks," in *Sixth Mexican International Conference on Computer Science (ENC'05)*, pp. 266–273, IEEE, 2005.

[67] 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 Transactions on Embedded Computing Systems (TECS)*, vol. 15, no. 3, p. 49, 2016.

[68] P. Regnier, G. Lima, E. Massa, G. Levin, and S. Brandt, "Run: Optimal multiprocessor real-time scheduling via reduction to uniprocessor," in *Proceedings of the 2011 IEEE*

32Nd Real-Time Systems Symposium, RTSS '11, (Washington, DC, USA), pp. 104–115, IEEE Computer Society, 2011.

[69] P. Regnier, G. Lima, E. Massa, G. Levin, and S. A. Brandt, "Multiprocessor scheduling by reduction to uniprocessor: an original optimal approach," *Real-Time Systems*, vol. 49, no. 4, pp. 436–474, 2013.

[70] E. Massa, G. Lima, P. Regnier, G. Levin, and S. A. Brandt, "Quasi-partitioned scheduling: optimality and adaptation in multiprocessor real-time systems," *Real-Time Systems*, vol. 52, no. 5, pp. 566–597, 2016.

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

[72] D. Casini, A. Biondi, and G. Buttazzo, "Semi-Partitioned Scheduling of Dynamic Real-Time Workload: A Practical Approach Based on Analysis-Driven Load Balancing," in *29th Euromicro Conference on Real-Time Systems (ECRTS 2017)* (M. Bertogna, ed.), vol. 76 of *Leibniz International Proceedings in Informatics (LIPIcs)*, (Dagstuhl, Germany), pp. 13:1–13:23, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017.

[73] B. B. Brandenburg and M. Gül, "Global scheduling not required: Simple, near-optimal multiprocessor real-time scheduling with semi-partitioned reservation," in *IEEE Real-Time Systems Symposium (RTSS 2016)*, pp. 99–110, 2016.

[74] ARINC, "Specification 651: Design guide for integrated modular avionics," 1997.

[75] N. Diniz and J. Rufino, "Arinc 653 in space," 2005.

[76] AUTOSAR, "Specification of rte software," 2017.

[77] G. Fernandez, J. Jalle, J. Abella, E. Quiñones, T. Vardanega, and F. J. Cazorla, "Computing safe contention bounds for multicore resources with round-robin and fifo arbitration," *IEEE Trans. Comput.*, vol. 66, pp. 586–600, Apr. 2017.

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

[79] G. Desirena-López, L. Vazquez, A. Ramírez-Ireviño, and D. Gómez-Gutiérrez, "Thermal modelling for temperature control in mpsoc's using timed continuous petri nets," in *Control Applications (CCA), 2014 IEEE Conference on*, pp. 2135–2140, IEEE, 2014.

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

[81] G. Desirena-Lopez, J. L. Briz, C. R. Vázquez, A. Ramírez-Treviño, and D. Gómez-Gutiérrez, "On-line scheduling in multiprocessor systems based on continuous control using timed continuous petri nets," in *13th International Workshop on Discrete Event Systems*, p. To appear, 2016.

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

[83] L. Rubio-Anguiano, G. Desirena-López, A. Ramírez-Treviño, and J. Briz, "Energy-efficient thermal-aware scheduling for rt tasks using tcpn," *IFAC-PapersOnLine*, vol. 51, no. 7, pp. 236 – 242, 2018. 14th IFAC Workshop on Discrete Event Systems WODES 2018.

[84] R. I. Davis and A. Burns, "Fpzl schedulability analysis," in *Proceedings of the 2011 17th IEEE Real-Time and Embedded Technology and Applications Symposium*, RTAS '11, (Washington, DC, USA), pp. 245–256, IEEE Computer Society, 2011.

[85] Y. Liu, R. P. Dick, L. Shang, and H. Yang, "Accurate temperature-dependent integrated circuit leakage power estimation is easy," in *Proceedings of the conference on Design, automation and test in Europe*, pp. 1526–1531, EDA Consortium, 2007.

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

[87] E. Bini and G. C. Buttazzo, "Measuring the performance of schedulability tests," *Real-Time Systems*, vol. 30, no. 1-2, pp. 129–154, 2005.

[88] MATLAB, *version 9.4 (R2018a)*. Natick, Massachusetts: The MathWorks Inc., 2018.

[89] V. Utkin, J. Guldner, and J. Shi, *Sliding mode control in electro-mechanical systems*, vol. 34. CRC press, 2009.

[90] H. K. Khalil and J. Grizzle, *Nonlinear systems*, vol. 3. Prentice hall New Jersey, 1996.

[91] ASHRAE, *Thermal guidelines for data Processing Environments*. ASHRAE Datacom Series, 2012.

[92] W. Horn, "Some simple scheduling algorithms," *Naval Research Logistics Quarterly*, vol. 21, no. 1, pp. 177–185, 1974.

[93] H. Kim, D. De Niz, B. Andersson, M. Klein, O. Mutlu, and R. Rajkumar, "Bounding and reducing memory interference in cots-based multi-core systems," *Real-Time Syst.*, vol. 52, pp. 356–395, May 2016.