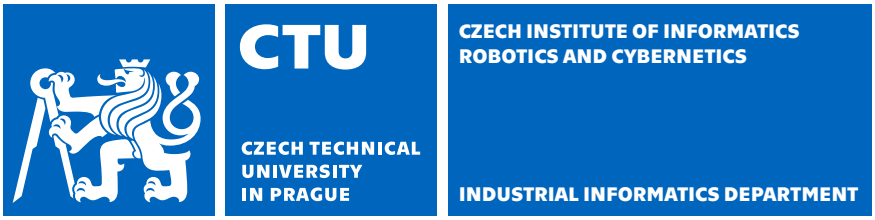

Thermal-Aware Scheduling for MPSoC in the Avionics Domain: Tooling and Initial Results

Ondřej Benedikt, Michal Sojka, Pavel Zaykov, David Hornof, Matěj Kafka, Přemysl Šůcha, Zdeněk Hanzálek

DOI: https://doi.org/10.1109/RTCSA52859.2021.00026
Cite as: O. Benedikt, M. Sojka, P. Zaykov, D. Hornof, M. Kafka, P. Šůcha, and Z. Hanzálek. Thermal-aware scheduling for mpsoc in the avionics domain: Tooling and initial results. In *2021 IEEE 27th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 159–168, 2021

Thermal-Aware Scheduling for MPSoC in the Avionics Domain: Tooling and Initial Results

Ondřej Benedikt, Michal Sojka, Pavel Zaykov*, David Hornof, Matěj Kafka, Přemysl Šůcha, Zdeněk Hanzálek

Czech Technical University in Prague
Prague, Czech Republic
ondrej.benedikt@cvut.cz

*Honeywell International s.r.o.
Brno, Czech Republic
pavel.zaykov@honeywell.com

***Abstract*—The demand for high-performance computing leads to the adoption of modern Multi-Processor System-on-Chip platforms in the avionics domain, where many applications are safety-critical. To fulfill the safety requirements, it is vital to avoid the platform's overheating. In this paper, we propose a task mapping method, MultiPAWS, for thermal-aware allocation of the safety-critical avionics workloads under time isolation constraints. With the help of MultiPAWS, we jointly find an optimal number of scheduling windows and their lengths and optimal mapping of the workload to these windows and available CPU cores. To guide the optimization, we introduce a thermal model based on power-characteristic coefficients, which we experimentally identify for a benchmark dataset on NXP i.MX8QuadMax platform (based on ARMv8 big.LITTLE architecture). Furthermore, to mimic the execution of safety-critical avionics applications, we introduce DEmOS, an open-source Linux-based scheduler. DEmOS provides a time-partitioned scheduling similar to the ARINC 653 standard. We use DEmOS for the experimental evaluation on the i.MX8 platform. The experimental results suggest that MultiPAWS achieves over a 12 % decrease of the platform temperature compared to the minimum-utilization-based approach. Moreover, we demonstrate how MultiPAWS can be used in design space exploration for finding the trade-off between the platform temperature and the length of the scheduling hyper-period.**

***Index Terms*—thermal-aware task mapping, MPSoC, safety-critical, ARINC 653**

I. Introduction

Many high-performance Multi-Processor Systems-on-Chip (MPSoC) have been adopted or considered for safety-critical domains such as aerospace. Apart from guaranteeing the high-performance, the safety-critical systems shall also operate under harsh environmental conditions such as dust, vibrations, and extended thermal ranges. In the context of safety and reliability, it is vital to operate within a given thermal envelope. One of the most straightforward ways to guarantee the thermal envelope is to introduce active cooling, commonly implemented by forced airflow. However, the active cooling significantly complicates the mechanical design and increases the cost. An alternative way is to employ passive cooling techniques, such as Dynamic Voltage and Frequency Scaling (DVFS), Dynamic Power Management (DPM), or thermal-aware task mapping. In this work, we focus solely on the latter-most.

We propose an optimization method (referred to as MultiPAWS) for thermal-aware static scheduling of a safety-critical workload under time isolation constraints. The method maps the applications on the computing resources and sequences them in time.

The optimization goal in MultiPAWS is to find a schedule by minimizing the platform's steady-state temperature while the workload is periodically executed. We demonstrate, both theoretically and experimentally, that for relatively short periods (compared to the platform's thermal time constants), the steady-state temperature minimization is essentially equivalent to the minimization of the average power consumption of the schedule. To guide the optimization, we introduce a power model based on power-characteristic coefficients, which are experimentally measured for each task.

To evaluate the MultiPAWS on a real computing platform, we needed an execution environment similar to the safety-critical avionics Real-Time Operating System (RTOS). Such RTOSes are proprietary [1]–[3]. More importantly, we target i.MX8QuadMax MPSoC by NXP [4] and such cutting-edge computing platforms are often not yet fully supported by the avionics RTOSes. Linux is usually the first operating system supported on such cutting-edge computing platforms. Therefore, we evaluate the MultiPAWS under Linux. To allow the execution of avionics workloads under Linux, we introduce an open-source tool that we referred as DEmOS [5]. DEmOS mimics the time execution of an avionics safety-critical ARINC-653 RTOS scheduler in a Linux environment and enables us to validate various optimization methods and thermal management techniques on a real computing platform.

To summarize, the contributions of this paper are as follows:

- We propose a MultiPAWS method designed for thermal-aware task mapping of the safety-critical workload under time partitioning constraints.
- We introduce the power-estimation model, which forms the basis of the MultiPAWS method and is based on the extensive benchmarking of various workloads.
- We empirically evaluated the proposed MultiPAWS method on a real computing platform (i.MX8QuadMax MPSoC by NXP) and demonstrated how MultiPAWS can be used in a design-space exploration for finding trade-

offs between platform temperature and scheduling hyper-period.
- We introduce an open-source tool (DEmOS) that allows the evaluation of various thermal-aware techniques with avionics workloads under Linux.

The remainder of the paper is organized as follows. The problem statement and consequent solution are formally described in Section II and in Section III, respectively. The DEmOS tool is covered in Section IV. The experimental results are summarized in Section V. Finally, we review related works in Section VI and conclude the paper with Section VII.

II. System Model and Problem Definition

In this section, we introduce the formal notation and describe the problem of the thermal-aware mapping of safety-critical tasks under windows isolation constraints. The hardware platform model is introduced in Section II-A and the task model in Section II-B. Then, we describe the thermal and power models in Sections II-C and II-D. We conclude this section by summarizing the problem definition in Section II-E.

A. Hardware Platform Model

The platform is composed of a set of computing *clusters* $\mathcal{C} = \{C_1, \ldots, C_m\}$. Each cluster C_k is associated with a number $c_k \in \mathbb{Z}_{>0}$ representing its *capacity*, i.e., the number of its computing elements (cores). The vector of capacities $(c_1, c_2, \ldots, c_m)$ is denoted by $\boldsymbol{c}$. The individual computing elements of each cluster are assumed to be identical, considering their performance as well as the power/thermal characteristics. All cores within a single cluster are running at the same clock frequency, while different clusters can be operating at different frequencies. We assume that the clock frequencies are given and fixed, i.e., DVFS is not considered.

B. Task Model

Figure 1 illustrates the notation of the task model. The basic unit of our task model is a partition, denoted as $\tau_i \in \mathcal{T}$, where $\mathcal{T}$ is a set of partitions $\mathcal{T} = \{\tau_1, \ldots, \tau_n\}$. We assume that a partition hosts a single, single-threaded process. Each partition needs to be scheduled within a *major frame*, i.e., within a time interval $[0, h]$, where h is the major frame length (also called *hyper-period*). We assume that each partition is ready at time zero and needs to finish at or before time h. Partitions are independent. Each partition needs to be mapped to a single cluster $C_k \in \mathcal{C}$ and a single time *window* W_j, defined by its offset and length within the major frame. Windows do not overlap and cover the whole major frame. The partitions assigned to one cluster and window are mapped to the individual cores in arbitrary order (at most one partition per core). Note that the number of windows is not known a priori and needs to be found.

Each partition τ_i and each cluster C_k has an associated execution time $e_{i,k} \in \mathbb{Z}_{>0} \cup \{\infty\}$ representing the number of time units needed to execute a partition τ_i in a given cluster C_k. If a partition τ_i cannot be executed in a cluster C_k, then we set $e_{i,k} \coloneqq \infty$. The execution times are given by matrix $E = [e_{i,k}]_{\tau_i \in \mathcal{T}, C_k \in \mathcal{C}}$.

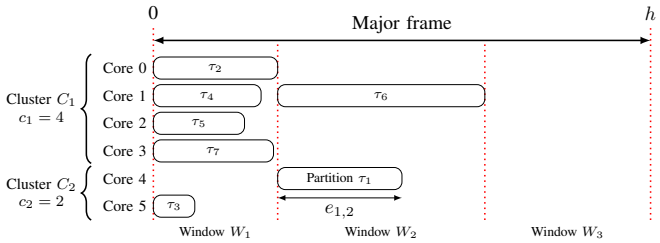

Fig. 1. A time-partitioned schedule with two CPU clusters, three windows, and seven partitions.

C. Thermal Model

Thermal models are often linked with the thermal-electrical analogy and RC circuits networks, capturing the transient behavior of the individual components of the target processor [6], [7]. The dependent physical parameters, including the thermal conductance and capacitance values, need to be identified to employ such a thermal model on a given processor practically. Moreover, the processor layout shall be known to model the heat dissipation and transfer among the cores. Precise temperature measurements from multiple places on a chip are needed to identify the parameters. However, modern complex MPSoCs often provide just a limited number of on-chip thermal sensors with limited resolution. In such a setting, it is impossible to distinguish how the individual computing resources thermally influence each other. Therefore, our model consists of a single thermal node that approximates the overall on-chip temperature.

Temperature evolution in time is primarily determined by thermodynamical parameters. It turns out that thermal time constants of the MPSoC platforms are much larger than periods of typical safety-critical avionics applications [8]. Therefore, in our model, we neglect the thermal dynamics and focus on a steady-state temperature only.

Assuming a single node and a steady-state temperature, the thermal model proposed in [6] simplifies to the following relation between the steady-state temperature T_∞ and the average power consumption P:

$$T_\infty = \beta P + T_{\text{amb}}, \tag{1}$$

where T_{amb} is the ambient temperature, and β represents the inverse of the chip's thermal conductance in [K/W]. Hence in this setting, minimization of the steady-state temperature can be achieved by minimizing the power consumption. We experimentally validate the linear correlation between P and T_∞ in Section V-C.

D. Power Model

The power models of CMOS devices were extensively studied in the past [9]–[11]. A widely-adopted approach is to model the *dynamic*, *static*, and *short-circuit* power, as described by Bambagini et al. [11]:

$$P = \underbrace{\alpha C V^2 f}_{\text{dynamic}} + \underbrace{V I_{\text{leak}}}_{\text{static}} + \underbrace{\alpha V I_{\text{short}}}_{\text{short-circuit}}, \tag{2}$$

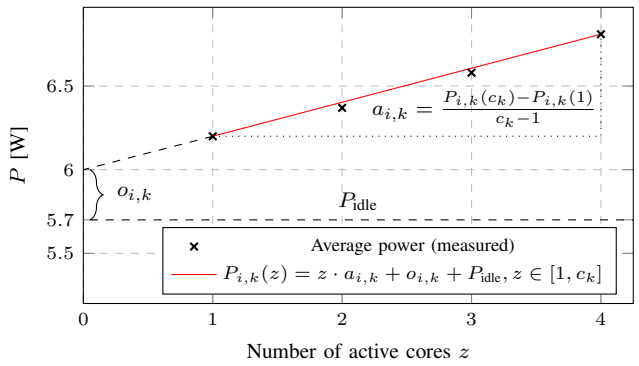

Fig. 2. Coefficients $o_{i,k}$ and $a_{i,k}$ for $C_k = \text{A53}$ and $\tau_i = \text{dijkstra}$.

where C is the load capacitance, α is the factor corresponding to the transistor switching activity, V is the supply voltage, f is the clock frequency, I_{leak} is the leakage current, and I_{short} is the current between the supply voltage and ground during the gate switching.

Such a model is sufficient for simple circuits, but using it for complex MPSoC platforms is not straightforward, e.g., due to multiple clock domains on the chip. Since we assume that the frequency of the individual clusters is fixed, we do not need to model the power-frequency relation. We use an empirical power model based on experimental measurements instead.

The model that we use is similar to the model proposed by Chen et al. [12], where each task is characterized by a number representing its activity factor. Contrary to [12], where only randomly generated power characteristics were used, we discuss how to obtain such characteristics for each task. Also, we improve the model by identifying two power characteristic numbers, instead of one, capturing the activity of the task and part of the system's power needed to execute it.

We characterize each partition $\tau_i \in \mathcal{T}$ by two numbers for each cluster $C_k \in \mathcal{C}$: the *activity coefficient* $a_{i,k}$ and the *offset coefficient* $o_{i,k}$, respectively. These coefficients (shown in Fig. 2) are obtained by benchmarking each partition at each cluster in the following way: for each partition $\tau_i \in \mathcal{T}$ and cluster $C_k \in \mathcal{C}$, we execute the partition in a loop at 1 and c_k cores of C_k. These two points then characterize a linear segment $P_{i,k}(z)$, approximating the average power consumption of task τ_i running on z cores of C_k for $z \in [1, c_k]$. The activity coefficient $a_{i,k}$ is then computed as the slope of $P_{i,k}(z)$. The offset coefficient $o_{i,k}$ is the offset of $P_{i,k}(z)$ with respect to the idle power consumption of the platform P_{idle}. When no cores are active ($z = 0$), the average power consumption is P_{idle}. The illustration for one particular benchmark is shown in Figure 2.

Note that more complex methods could be used to identify the power characteristics of the tasks. For example, the measurements could be done for all cores ($z \in \{1, \ldots, c_k\}$) while fitting the linear segment, e.g., using the least-squares method. Even more, piecewise-linear or more complex relations could be used to model the power-behavior of the partitions. Nevertheless, we use just a linear relation in this study since (i) linear relation approximates the behavior of most of the tested benchmarks reasonably well, as shown in Section V-D, (ii) it is simple – just two coefficients are needed to characterize each partition, which is beneficial when building the optimization models and algorithms, (iii) only two measurements are needed for each partition and cluster, which makes the benchmarking reasonably short.

In the remainder of this section, we describe how the task's characteristics are used to estimate the average power consumption. First, we assume to have a single window W of length l, and for each cluster $C_k \in \mathcal{C}$ a set of tasks $\mathcal{T}_k$ assigned to it. We estimate the steady-state average power consumption $P(W)$ of the window when being executed periodically:

$$P(W) = \sum_{C_k \in \mathcal{C}} \sum_{\tau_i \in \mathcal{T}_k} \left(a_{i,k} \cdot \frac{e_{i,k}}{l} \right) + \max_{\substack{C_k \in \mathcal{C} \\ \tau_i \in \mathcal{T}_k}} o_{i,k} + P_{\text{idle}}. \quad (3)$$

Note that if z tasks of the same type are executed within the window, and the execution of each one spans the whole window, then expression (3) can be reduced to $P_{i,k}(z)$.

For a schedule of multiple windows $W_j \in \mathcal{W}$ where W_j has length l_j, we estimate the average power consumption to be as follows:

$$\sum_{W_j \in \mathcal{W}} \frac{l_j}{h} P(W_j) + \frac{h - \sum_{W_j \in \mathcal{W}} l_j}{h} \cdot P_{\text{idle}}. \quad (4)$$

Note that the assignment of tasks to the individual windows and clusters is unknown.

E. Problem Definition

The input is given by a tuple $(\mathcal{C}, \boldsymbol{c}, \mathcal{T}, E, A, O, h)$, i.e., the cluster set $\mathcal{C}$, vector of cluster's capacities $\boldsymbol{c}$, set of partitions $\mathcal{T}$, matrix of their execution times E, matrices of their activity and offset coefficients A and O, and the major frame length h. The goal is to find solution $\sigma = (q, \boldsymbol{l}, a^{(w)}, a^{(c)})$, where $q \in \mathbb{Z}_{>0}$ is the number of windows, $\boldsymbol{l} = (l_1, \ldots, l_q) \in \mathbb{Z}^q_{>0}$ is the vector of windows' lengths, $a^{(w)} : \mathcal{T} \to \mathcal{W}$ is a total function mapping the partitions to windows, and $a^{(c)} : \mathcal{T} \to \mathcal{C}$ is a total function mapping the partitions to resource clusters. We denote the set of windows as $\mathcal{W}$, where $\mathcal{W} = \{W_1, W_2, \ldots, W_q\}$.

A solution σ is called *feasible* if the following three types of constraints are satisfied:

(i) each window $W_j \in \mathcal{W}$ is at least as long as the longest partition assigned to it:

$$l_j \geq \max_{\substack{\tau_i \in \mathcal{T} \\ C_k \in \mathcal{C}}} \left\{ e_{i,k} \mid a^{(w)}(\tau_i) = W_j \wedge a^{(c)}(\tau_i) = C_k \right\}, \quad (5)$$

(ii) the total length of all windows is at most equal to the major frame length:

$$\sum_{W_j \in \mathcal{W}} l_j \leq h, \quad (6)$$

and (iii) at most c_k partitions are assigned to cluster $C_k \in \mathcal{C}$ in each window $W_j \in \mathcal{W}$:

$$\sum_{\tau_i \in \mathcal{T} : a^{(w)}(\tau_i) = W_j} \mathbb{1}_{[a^{(c)}(\tau_i) = C_k]} \leq c_k. \tag{7}$$

We seek such a feasible solution that minimizes the steady-state temperature of the platform reached after its repeated execution. The minimization of the steady-state temperature is achieved by minimizing the average power consumption (see Section II-C). To approximate the energy consumed by the execution of a single window, we use the power model (3)–(4). By using this model, the average power consumption of solution σ can be expressed as follows:

$$\frac{1}{h} \sum_{W_j \in \mathcal{W}} \left(\sum_{\tau_i \in \mathcal{T} : a^{(w)}(\tau_i) = W_j} \sum_{C_k \in \mathcal{C} : a^{(c)}(\tau_i) = C_k} (a_{i,k} \cdot e_{i,k}) + \max_{\substack{\tau_i \in \mathcal{T} \\ C_k \in \mathcal{C}}} \left\{ o_{i,k} \cdot l_j \,|\, a^{(w)}(\tau_i) = W_j \wedge a^{(c)}(\tau_i) = C_k \right\} \right). \tag{8}$$

Note that constant P_{idle} was omitted since it does not have an impact on the optimization. We seek the *optimal* solution, i.e., a feasible solution minimizing (8).

III. Proposed Solution Method

To solve the optimization problem, we propose an Integer Linear Programming (ILP) model called Multi-Processor Power-Aware Windowed-Scheduler (MultiPAWS). The MultiPAWS model has the following three types of variables.

First, we set the maximal number of windows in the schedule $\ell_{\max} = n$ (in the worst-case, there is one window per partition). Having this bound, we introduce a non-negative variable $\hat{l}_j \in \mathbb{Z}_{\geq 0}$ for each potential window $W_j \in \mathcal{W}_{\max} = \{W_1, W_2, \ldots, W_{\ell_{\max}}\}$ representing the window's length.

Second, for each partition $\tau_i \in \mathcal{T}$, the potential window $W_j \in \mathcal{W}_{\max}$ and cluster $C_k \in \mathcal{C}$, we introduce a binary variable $x_{i,j,k} \in \{0, 1\}$ modeling the mapping of a task τ_i to a window W_j and a cluster C_k.

Third, we employ auxiliary variable $y_{i,j,k} \in \mathbb{R}_{\geq 0}$ for each partition $\tau_i \in \mathcal{T}$, potential window $W_j \in \mathcal{W}_{\max}$, and cluster $C_k \in \mathcal{C}$ to capture the influence of the offset coefficient $o_{i,k}$ of a partition τ_i on a cluster C_k and in a window W_j.

The complete mathematical model MultiPAWS follows:

$$\min \frac{1}{h} \sum_{W_j \in \mathcal{W}_{\max}} \left(\sum_{\tau_i \in \mathcal{T}} \sum_{C_k \in \mathcal{C}} (x_{i,j,k} \cdot a_{i,k} \cdot e_{i,k}) + \max_{\substack{\tau_i \in \mathcal{T} \\ C_k \in \mathcal{C}}} \{ y_{i,j,k} \} \right) \tag{9}$$

subject to:

$$\hat{l}_j \geq x_{i,j,k} \cdot e_{i,k} \quad \forall \tau_i \in \mathcal{T}, W_j \in \mathcal{W}_{\max}, C_k \in \mathcal{C}, \tag{10}$$

$$\sum_{W_j \in \mathcal{W}} \hat{l}_j \leq h, \tag{11}$$

$$\sum_{\tau_i \in \mathcal{T}} x_{i,j,k} \leq c_k \quad \forall W_j \in \mathcal{W}_{\max}, C_k \in \mathcal{C}, \tag{12}$$

$$\sum_{W_j \in \mathcal{W}} \sum_{C_k \in \mathcal{C}} x_{i,j,k} = 1 \quad \forall \tau_i \in \mathcal{T}, \tag{13}$$

$$x_{i,j,k} = 1 \Rightarrow y_{i,j,k} = o_{i,k} \cdot \hat{l}_j \quad \forall W_j \in \mathcal{W}_{\max}, C_k \in \mathcal{C}. \tag{14}$$

Constraints (10) ensure that each window is at least as long as the longest partition assigned to it, as specified by (5). Constraint (11) is a direct equivalent of (6) constraining the total length of all windows by major frame length h. Constraints (12) implement the cluster capacities as required by (7). Expression (13) guarantees that each partition will be assigned to exactly one cluster and window. Together, these constraints ensure that the schedule found by the solver is feasible.

Finally, the implication (14) sets auxiliary variables $y_{i,j,k}$, which are used in the objective (9). Suppose that partition τ_i is assigned to window W_j and cluster C_k. In that case, the auxiliary variable $y_{i,j,k}$ is set to be equal to the product of the partition's offset coefficient $o_{i,k}$ and the window's length $\hat{l}_j$; otherwise, the value is arbitrary (and the solver will force it to 0, which is, in that case, optimal w.r.t. (9)). Objective (9) is a direct translation of expression (8) in terms of variables $x_{i,j,k}$ and $y_{i,j,k}$.

Note that the model MultiPAWS as described by (9)–(14) is non-linear due to the maximum in (9) and implication in (14). Nevertheless, both of these expressions can be linearized (the first one by introducing additional variables for each window, and the second one by rewriting with 'big M'), and the modern solvers even do the linearization internally. Thus, for better readability, we leave the model in the non-linear form.

To speed-up the solution process, we introduce the symmetry-breaking constraints that do not affect the solution quality, but prune part of the search space. Since the windows are independent, we enforce their order, according to their lengths:

$$\hat{l}_j \geq \hat{l}_{j+1} \quad \forall j \in \{1, 2, \ldots, \ell_{\max} - 1\}. \tag{15}$$

A. Reconstruction of the Solution

Let us denote the optimal values of $x_{i,j,k}$ and $\hat{l}_j$ found by the solver by $x^\star_{i,j,k}$ and $\hat{l}^\star_j$, respectively. After solving the MultiPAWS model, solution $\sigma = (q, \boldsymbol{l}, a^{(w)}, a^{(c)})$ is reconstructed in the following way:

(i) set q as the number of potential windows, to which at least one partition was assigned by the solver:

$$q = \left| \left\{ W_j \in \mathcal{W}_{\max} \,\middle|\, \sum_{\tau_i \in \mathcal{T}} \sum_{C_k \in \mathcal{C}} x^\star_{i,j,k} \geq 1 \right\} \right|, \tag{16}$$

(ii) set the windows lengths $\boldsymbol{l}$ according to the values of the corresponding variables $\hat{l}_j^\star$:

$$\boldsymbol{l} = \left(\hat{l}_j^\star \,\middle|\, W_j \in \mathcal{W}_{\max}, \sum_{\tau_i \in \mathcal{T}} \sum_{C_k \in \mathcal{C}} x_{i,j,k}^\star \geq 1 \right), \tag{17}$$

(iii) assign the partition τ_i to window according to the value of $x_{i,j,k}^\star$:

$$\forall \tau_i \in \mathcal{T}, W_j \in \mathcal{W}_{\max} : a^{(w)}(\tau_i) = W_j \Leftrightarrow \sum_{C_k \in \mathcal{C}} x_{i,j,k}^\star = 1, \tag{18}$$

(iv) assign the partition τ_i to cluster according to the value of $x_{i,j,k}^\star$:

$$\forall \tau_i \in \mathcal{T}, C_k \in \mathcal{C} : a^{(c)}(\tau_i) = C_k \in \mathcal{C} \Leftrightarrow \sum_{W_j \in \mathcal{W}_{\max}} x_{i,j,k}^\star = 1. \tag{19}$$

IV. Time-Partitioned Schedulers and DEmOS Tool

To execute partitions $\mathcal{T}$ on a real hardware following a given schedule, we developed an open-source tool called DEmOS [5]. DEmOS provides the scheduling of partitions following the ARINC 653 concept in a Linux environment. The DEmOS considers the ARINC 653 specifics of the Deos™ RTOS implementation.

A. ARINC 653

ARINC 653 (abbreviated as A653) is a specification of the operating environment for the application software used within Integrated Modular Avionics (IMA). More specifically, it defines the interface between the Operating System and the applications [13]. The A653 services can be grouped into the following categories: partition management, process management, time management, inter-partition communication, intra-partition communication, and health monitoring. The DEmOS tool implements only scheduling, which is a part of partition management services.

The partition management allows partitioning the execution in space (memory partitioning) and time (temporal partitioning). Therefore, partitions with different criticality levels can execute in the same system without affecting one another spatially or temporally.

Partitions are scheduled deterministically on a fixed, cyclic basis. Each full cycle of the schedule is called a *major frame*. Partitions mapped onto one or more scheduling *windows* within a major frame. Each window is being defined by its offset from the start of the major frame and expected duration. The order of partition activation is defined at configuration time using configuration tables [13].

The original ARINC standard published in 2010 did not address its use in multi-core processor avionics systems. However, since there was strong market demand, the standard has evolved, and ARINC 653 Part 1 Supplement 4 (ARINC653P1-4), published in 2015 [14], mentions multi-core processors and supports parallel execution of partitions on multiple cores [15].

B. Deos RTOS

Deos™ [1] by DDC-I is one of the popular RTOSes in the avionics domain. Deos™ supports ARINC-653 time and space partitioning and has been certified in numerous safety-critical products to DO-178 DAL-A. In each partition, Deos™ RTOS supports one of the following three schedulers – harmonic Rate Monotonic, ARINC-653, and POSIX (leveraging a para-virtualized RTEMS instance).

In a multi-core processor, a single Deos™ RTOS instance handles all processor cores. A window has an ID, fixed length, and spawns across all cores. Each core within the window has its own scheduler (e.g., A653, POSIX).

User applications (processes with threads) are mapped to the windows, cores, and schedulers during the system configuration. In the context of an IMA architecture, Deos™ provides a set of mechanisms that allow a single multi-core platform to host multiple applications of different criticality levels.

In many cases, just two levels are used: Safety-Critical (SC) and Best-Effort (BE). Deos™ extends the A653 scheduling scheme by allowing SC and BE partitions to share a core in a window. Specifically, SC partitions are always granted for execution within a given window, and BE partitions can be optionally scheduled once all SC partitions in the window complete. In this paper, we focus only on the scheduling of the SC partitions. The motivation is that under the worst conditions, the BE partitions will not be executed to avoid platform overheating.

C. DEmOS Tool

DEmOS partially mimics the time execution of a Deos™ RTOS scheduler (with windows and partitions) in the Linux environment. Although DEmOS partially implements the ARINC-653 scheduling scheme, the terminology used in this section differs from ARINC-653 terminology. Specifically, an ARINC process represents a single thread of execution. In contrast, we use the term process for a Linux process, which can be composed of one or more threads of execution. Similarly, the ARINC partition is a collection of ARINC processes, which all share an address space, whereas DEmOS partition is a collection of Linux processes, which do not share an address space. DEmOS implements only the scheduling; it does not aim to be A653 API compatible and provide all A653 services.

DEmOS is a user-space program that uses the Linux *cgroups* mechanism to implement the partitions as groups of Linux processes. Scheduling happens at the cgroup level according to the configured schedule. The cgroup *freeze* functionality is used to prevent individual processes from executing outside of their windows or after exhausting their budgets. The *cpuset* cgroup controller limits the execution of processes to the configured CPU cores. Besides this, DEmOS does neither limit use of other Linux services nor change Linux scheduling policies. Therefore, processes running under DEmOS are free to use any Linux scheduling policy such as SCHED_FIFO and can internally spawn multiple threads.

```
windows:
  - length: 100
    sc_partition: [{cmd: prg1 --arg=value}]
  - length: 200
    slices:
      - {cpu: 0,
         sc_partition: [{cmd: prg2, budget: 20}],
         be_partition: [{cmd: prg3}]}
      - {cpu: 1, sc_partition: [{cmd: prg4}, {cmd: prg5}]}
```

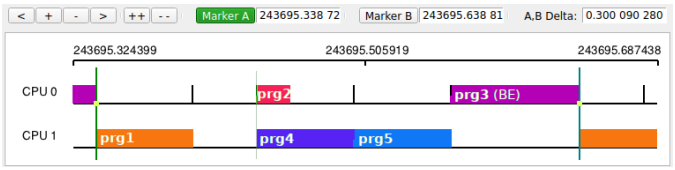

Fig. 3. Exemplary DEmOS configuration file (top) and a trace of the DEmOS execution with that configuration (recorded with trace-cmd and visualized with the kernelshark tool).

DEmOS is configured via YAML-formatted files, which specify parameters for the windows, partitions, and processes. The configuration file can be either written manually or auto-generated by higher-level tools, such as the MultiPAWS implementation used in this paper.

Figure 3 shows an exemplary configuration file for a dual-core system (CPU0 and CPU1) with two windows (spanning 100 and 200 ms, respectively) and five programs (prg1 – prg5). These programs are single-threaded user-defined applications that execute a `while(1)` loop. The bottom part of the figure shows a time trace of how DEmOS executes the programs. The major frame, composed of the two windows, is delimited with thick green vertical lines. No CPU restrictions were configured for prg1 (orange), and we see in the trace that Linux chose to execute it on CPU1. The other processes running in the second window are restricted to run on specific CPUs. Process prg2 (on CPU0, red) has explicitly set a budget so it cannot run longer than 20 ms. The other safety-critical (SC) processes have a default budget, which DEmOS configures so that all processes in safety-critical partitions occupy 60 % of window length. This is why there is a gap on CPU1 between prg1 (orange) and prg4 (dark blue) and why both prg4 and prg5 (light blue) stop their execution after approximately 60 ms. The best-effort process (prg3, magenta) executes in the remaining 40 % of the second window. The trace also shows (with black vertical lines) that DEmOS itself executed on CPU0 but only when action was required on one of the CPUs.

DEmOS can run arbitrary (unmodified) programs and ensures that they run only in configured windows and on configured CPUs. However, it might be beneficial for a program to use `demos-sch` library, which allows it to notify DEmOS about important events. Currently, applications can notify DEmOS about the completion of their initialization phase and the completion of a periodic job. The initialization phase is not a subject of strict window-based scheduling. The periodic job completion notification allows DEmOS to schedule the following process in the partition before exhaustion of the previous process's budget.

Although we do not use it in this paper, DEmOS supports frequency scaling (DVFS). This support allows creating custom *power policies*, which change operational frequencies of individual CPU clusters (or cores if supported by the hardware).

V. Experimental Evaluation

In this section, we first describe the experimental platform and benchmark applications. Then, we provide experimental results for both the proposed power model (3)-(4) and the scheduling method MultiPAWS.

A. Experimental Platform

As an experimental platform, we choose the i.MX8 MPSoC [4], the latest generation of i.MX family by NXP, which is nowadays among the prominent computing MPSoCs for the safety-critical domain. In one of its most performant variations, the i.MX8 processor offers two CPU clusters and two identical onboard GPUs (Vivante GC7000). The first CPU cluster has four ARM Cortex-A53 cores, while the second has two ARM Cortex-A72 cores. The interested reader may find further information about our testbed setup, measurements methodology, as well as some preliminary results of the platform benchmarking in [8].

Throughout all experiments, the computing clusters were set to be operating at the highest possible frequency, which is 1600 MHz for the A72 cluster, and 1200 MHz for the A53 cluster, respectively. The idle power of the platform P_{idle} was empirically measured to be $P_{\text{idle}} = 5.7$ W.

B. Benchmark Applications

In the evaluation, we apply the following benchmarks: (i) a subset of Tacle-bench suite [16] consisting of 6 benchmarks (dijkstra, fft, prime, sha, susan, test3), (ii) a software 3D rendering tool known as tinyrenderer [17], and (iii) memory stressing benchmark membench [8]. Membench is used in two configurations – membench-1M (stressing the L2 cache) and membench-4M (stressing the main memory).

For each benchmark b, we report its execution time per iteration on each i.MX8 cluster together with the ratio between them (denoted by γ_b) in Table I.

TABLE I
AVERAGE EXECUTION TIME NEEDED FOR A SINGLE ITERATION MEASURED FOR THE TESTED BENCHMARKS.

	Time per iteration		Ratio
Benchmark b	C_k = A53	C_k = A72	γ_i
dijkstra	17.5 ms	10.8 ms	1.62
fft	0.332 ms	0.114 ms	2.91
prime	0.372 µs	0.199 µs	1.87
sha	0.454 ms	0.170 ms	2.67
susan	20.5 ms	5.63 ms	3.64
test3	67.1 ms	34.8 ms	1.92
tinyrenderer	953 ms	409 ms	2.33
membench-1M	814 ms	606 ms	1.34
membench-4M	7.15 s	5.45 s	1.31

We also report the activity and offset coefficients in Figure 4, which were obtained as described in Section II-D.

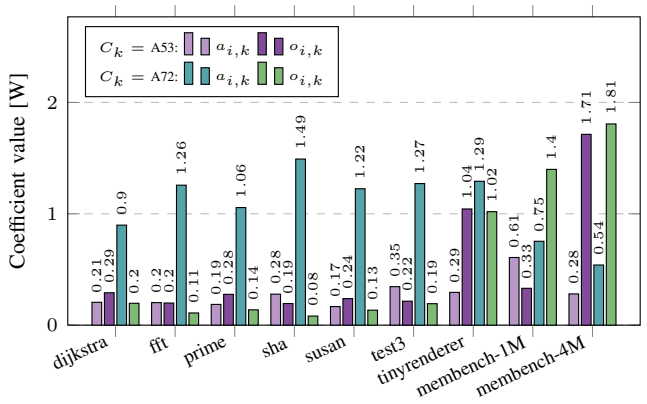

Fig. 4. Measured coefficients of the tested benchmarks.

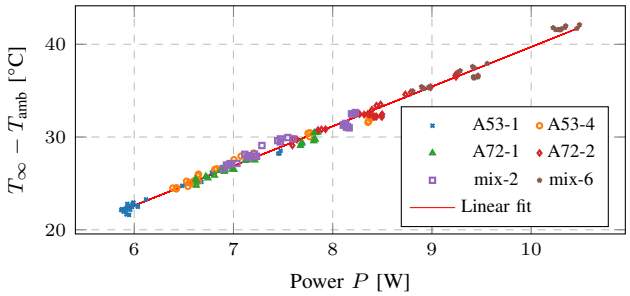

Fig. 5. Relation between the measured power and relative temperature; linear fit: $(T_\infty - T_{\text{amb}}) \simeq 4.31P - 3.21$.

C. Average Power and Steady-State Temperature

When identifying the coefficients, each benchmark was executed on 1 and 4 A53 cores and 1 and 2 A72 cores, respectively. Additionally, we executed the benchmarks in mix-2 and mix-6 configurations, where in the first case, the benchmark was running at one A53 core and one A72 core, while in the second case, the benchmark was running at all 6 cores of the i.MX8 platform. Each measurement was repeated three times – after 15 minutes of each benchmark running, the relative steady-state temperature $T_\infty - T_{\text{amb}}$ was recorded. The resulting relation between the average power consumption and the relative steady-state temperature for various benchmarks running under various configurations is shown in Figure 5. Clearly, we observe the linear relation between the average power consumption and the steady-state temperature, as suggested by (1).

D. Power Estimation Evaluation

We evaluate the power model (3) for every single benchmark running in a loop in the following 4 configurations: A53-2 (two active A53 cores), A53-3 (three active A53 cores), mix-2 (one A53 and one A72 core active), and mix-6 (four A53 and two A72 cores active). Note that for configurations A53-1, A53-4, A72-1 and A72-2, and a single benchmark running, the prediction error is zero.

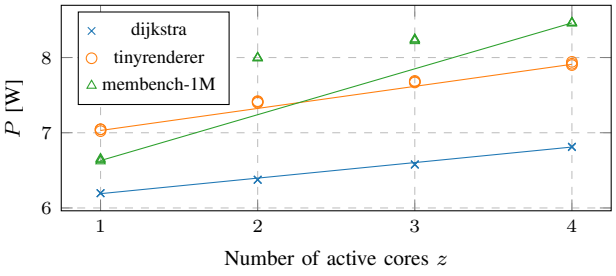

Fig. 6. Power of various benchmarks with respect to the number of active cores z and the corresponding segments of $P_{i,k}(z)$.

We measure the error ϵ defined as

$$\epsilon := \frac{P_{\text{measured}} - P_{\text{estimated}}}{P_{\text{estimated}}} \cdot 100. \tag{20}$$

The results are listed in Table II. For most of the benchmarks, except membench-1M, the estimation error is relatively low: –0.40 % on average, and at most –2.04 % for tinyrenderer running under configuration mix-6.

TABLE II
POWER PREDICTION ERROR OF A SINGLE BENCHMARK RUNNING ON DIFFERENT CLUSTERS IN DIFFERENT DEMOS CONFIGURATIONS.

	Error ϵ [%]			
τ_i	A53-2	A53-3	mix-2	mix-6
dijkstra	-0.41	-0.43	-1.07	0.40
fft	0.08	-0.18	-0.84	-0.67
prime	-0.26	-0.40	-1.52	0.53
sha	-0.42	-0.19	-0.96	-0.34
susan	-0.54	-0.23	-1.31	-0.15
test3	-0.18	-1.98	-1.11	-0.83
tinyrenderer	1.06	0.68	0.41	-2.04
membench-1M	9.38	4.63	-2.37	-16.15
membench-4M	0.94	0.78	0.37	-2.04

The benchmark membench-1M is an exception – when running on multiple cores, the individual instances of this benchmark are not independent since all of them compete for the L2 cache. This memory stressing benchmark is set such that one instance fills the whole L2 cache. Therefore, when multiple instances run in parallel, congestion occurs, negatively impacting the power consumption as well as the temperature. The relation between the average power consumption and the number of active cores for three different benchmarks is shown in Figure 6. Although the prediction, represented by the line segment, is not very accurate for membench, we argue that the power model (3) is a preliminary implementation, which will be improved in the future. Also, the membench is a rather specific benchmark, only stressing the memory. Most of the real-world CPU applications would behave more like the tinyrenderer benchmark.

E. Thermal-Aware Scheduling

Finally, we evaluate the MultiPAWS model. We compare it with two other methods: util-LTF and RA-LTF. These two methods work in two stages – first, the allocation of the tasks

to clusters is decided, then the isolation windows are created, and the tasks are statically scheduled.

For util-LTF, the allocation is done such that the utilization of the system is minimized (i.e., the total idle time of all cores is maximized). This is achieved by solving an ILP model. For RA-LTF, the allocation to clusters is decided randomly (both clusters have the same chance of being chosen).

The second stage is common for both util-LFT and RA-LTF – the tasks are scheduled one by one in order of their non-increasing execution times (hence LTF, i.e., the longest task first), minimizing the completion time of the last task in the major frame.

Due to the fact that each experiment is very time-consuming (a long time is needed for the steady-state temperature to stabilize), we compare the three methods, MultiPAWS, util-LTF, and RA-LTF, on a set of 6 instances only. For each instance, 25 partitions were created. Within each partition, one of the tested benchmarks, see Section V-B, (picked randomly with repetitions) was executed. For each partition τ_i, the execution time (in milliseconds) for the A72 cluster was generated uniformly from interval [40,160]. Then the execution time for A53 was scaled according to the ratio γ_b, see Table I. The major frame length was set to $\frac{\bar{p} \cdot n}{\kappa}$, where $n := 25$ for the tested instances, $\bar{p}$ is the average execution time (across all clusters), and $\kappa := 3.5$ is empirical constant setting the schedules not too tight to be infeasible, but not too loose to be trivial.

For each instance and tested method, three independent runs of the experiment were conducted. The relative steady-state temperature $T_\infty - T_{\text{amb}}$ was measured after 30 minutes when the temperature was already stable. Three different schedules, RA-LTF1, RA-LTF2 and RA-LTF3, were generated by the random assignment policy RA-LTF to illustrate its behavior. For six instances and five methods (MultiPAWS, util-LTF and RA-LTF1, RA-LTF2, and RA-LTF3), this gives 45 hours of the measurements running.

The results are summarized in Table III, reporting the average relative steady-state temperature and the standard deviation of the three runs. We see that MultiPAWS is able to reduce the relative steady-state temperature by up to 12 % compared with util-LTF and 7 % compared to RA-LTF.

TABLE III
COMPARISON OF THE RELATIVE STEADY-STATE TEMPERATURES $T_\infty - T_{\text{AMB}}$ OF TESTED METHODS ON BENCHMARK INSTANCES.

	$T_\infty - T_{\text{amb}}$ [°C]				
	MultiPAWS	util-LTF	RA-LTF-1	RA-LTF-2	RA-LTF-3
#1	30.6 ± 0.3	34.9 ± 0.5	33.0 ± 0.4	32.7 ± 0.3	32.0 ± 0.3
#2	29.5 ± 0.6	33.9 ± 0.3	32.1 ± 0.4	32.4 ± 0.1	32.3 ± 0.3
#3	29.9 ± 0.2	33.7 ± 0.3	31.4 ± 0.4	30.9 ± 0.7	31.6 ± 0.5
#4	30.2 ± 0.1	34.1 ± 0.2	31.9 ± 0.5	32.8 ± 0.2	30.7 ± 0.1
#5	30.1 ± 0.2	34.6 ± 0.2	33.5 ± 0.0	34.2 ± 0.1	33.2 ± 0.1
#6	30.2 ± 0.2	34.6 ± 0.1	32.6 ± 0.2	32.7 ± 0.2	32.8 ± 0.0

The schedules for instance #1 are visualized in Figure 7. We can observe that util-LTF is able to compactly schedule the tasks on A72 cores, which are faster and more powerful; however, at the price of increased temperature of the chip. Contrary to that, MultiPAWS, guided by the tasks' characteristics, utilizes the resources better by allocating the tasks to slower A53 cores and balancing the windows. Schedules produced by RA-LTF are somewhere in between – partially utilizing both clusters.

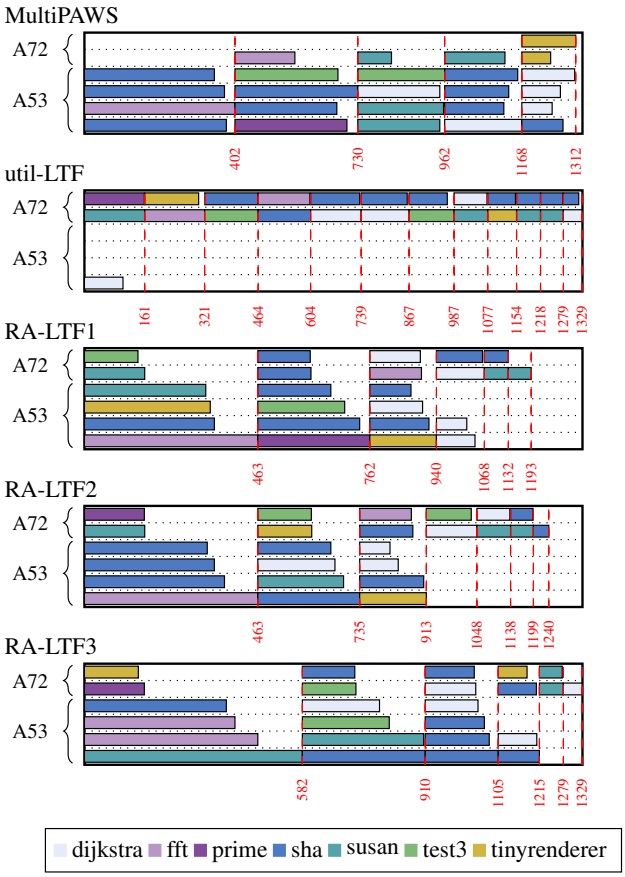

Fig. 7. Schedules provided by different methods for instance #1 with fixed major frame length $h = 1330$.

Note that even for these 25 tasks, finding such schedules, including the numbers of windows and allocations, manually might become quite challenging, especially when the major frame is tight.

F. Influence of the Major Frame Length on the Steady-State Temperature

The energy-time trade-off is the topic of many works, e.g. [18]. Here, we examine the potential trade-off between the major frame length and the relative steady-state temperature under time-partitioning constraints. This study is conducted with instance #1. We experiment with $h \in \{930, 1130, 1330, 1530, 1730, 1930\}$. Schedules for $h = 1330$ found by MultiPAWS and util-LTF are shown in Figure 7. Further, we illustrate the schedules found by MultiPAWS for different major frame lengths in Figure 8.

This time, we compare MultiPAWS, MultiPAWS-fixed, and util-LTF methods. Method MultiPAWS-fixed takes the schedule found by MultiPAWS for $h = 930$ and fixes it – i.e., after time 930, nothing is being executed, and the platform is idling (cooling). The results are shown in Fig. 9. The numbers

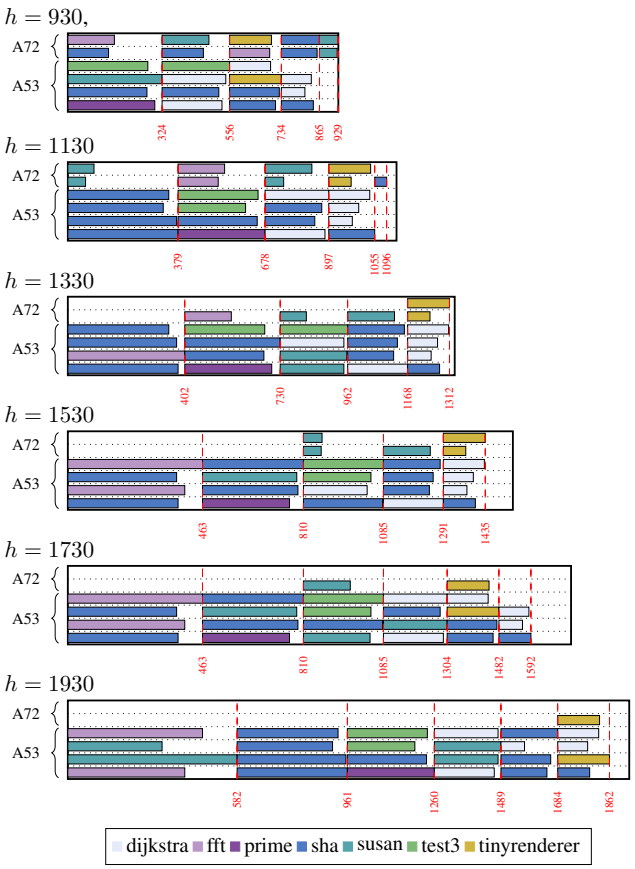

Fig. 8. Different schedules for instance #1 by MultiPAWS with changing major frame length $h \in \{930, 1130, 1330, 1530, 1730, 1930\}$.

reported for each data point represent the total utilization of all clusters.

One can see that a smart allocation and scheduling of the tasks combined with the potential increase of the major frame length can bring substantial benefits.

Clearly, when increasing the major frame length, there is more space for the thermal-aware allocation. The MultiPAWS method achieves the highest savings among the tested methods. Compared to MultiPAWS-fixed, we see the advantage of the tasks re-allocation and re-scheduling. The util-LTF method shows the lowest potential utilization of the platform, compensated by higher temperatures.

To put it into numbers, for this particular instance, we observe that by prolonging the major frame length $h = 930$ by about 21 %, MultiPAWS can improve the relative steady-state temperature by about 10 %, which is about 3 % better than rescaling the hyper-period (MultiPAWS-fixed). Making the major frame about two times of its original length (from $h = 930$ to $h = 1930$), we can obtain up to 26 % improvement in terms of the steady-state temperature and about 7 % improvement compared to MultiPAWS-fixed.

VI. Related Work

Thermal/power-aware task mapping on heterogeneous MPSoC has already received attention in the literature [12], [19]–

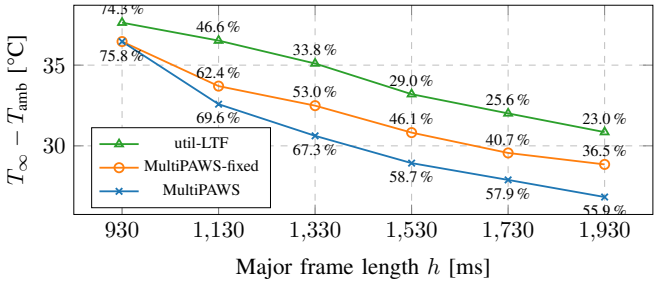

Fig. 9. Relation between the major frame length h and the relative steady-state temperature $T_\infty - T_{\text{amb}}$ for MultiPAWS method on instance 2; total utilization is reported as a percentage for each schedule.

[23]. Although several authors, such as [12], [22], [24], propose a mathematical model to solve the problem, the prevailing approach is to employ heuristics. A typical heuristics, such as those in [20], [21], work in several steps – first, the tasks are sorted according to some rule (e.g., power-intensive tasks first). Then, taking the tasks one by one, each task is mapped to a best-available resource w.r.t. some objective criterion. We argue that such a 'greedy' approach is not well-suited for time-partitioned scheduling, where the isolation windows make the individual cores and resources dependent on each other. Finding a schedule in a greedy way might not be feasible.

Suyyagh et al. [23] solve the power-aware task mapping problem with heuristics too, but similarly to our work, they profile the workload to guide the heuristics. However, their approach is not suitable for safety-critical applications and ARINC-653 scheduling.

In the avionics domain, the increased complexity of integrated modular avionics and multi-core platforms leads to the need to use model-based approaches that employ optimization techniques to find schedules and other system configuration parameters. Antonante et al. [25] use the optimization-based approach in the context of ARINC-653 scheduling for multi-core platforms. Similarly, Han et al. [26] use a model-based approach with optimization to solve ARINC-653 scheduling problem but only for a single-core platform. None of the mentioned works deals with the thermal aspects.

Balsini et al. [27] extend the real-time scheduling simulator RTSIM to simulate CPU power consumption of a heterogeneous computing platform. They use a similar approach to power modeling as in this paper – power models are fitted from data experimentally measured on real hardware. The difference is that they focus on simulation of event-triggered RTOS scheduling, whereas our approach allows to synthesize time-triggered schedules for avionics applications.

With respect to executing ARINC-653 workloads in Linux environments, both kernel-level and user-level approaches are used. Notable kernel-level ARINC schedulers are [28], [29]. As these are not a part of the mainline Linux kernel, their use on modern hardware platforms would require significant porting effort. A user-level ARINC-653 emulator was developed by Dubey et al. [30] and later published under a non-open-source (see https://opensource.org/osd-annotated) license at

https://github.com/adubey14/arinc653emulator. Compared to DEmOS, the emulator uses POSIX signals to start/stop the partitions, which is known not to be 100 % reliable. DEmOS relies on cgroups, which are guaranteed to work reliably. The emulator implements significant portion of scheduling-unrelated ARINC APIs, which DEmOS does not try to provide. On the other hand, DEmOS supports DVFS management, which is not available in the emulator.

VII. Conclusions

We proposed a task mapping method, MultiPAWS, for thermal-aware allocation of avionics safety-critical workloads under ARINC-653 time isolation constraints. Based on the extensive benchmarking of various workloads, we introduced a power-estimation model and identified its parameters. The model is the basis of our MultiPAWS optimization method.

Furthermore, we introduced an open-source tool DEmOS that allows the evaluation of thermal-aware techniques by mimicking the A653 scheduler under Linux.

We experimentally evaluated MultiPAWS on a real MPSoC platform, NXP i.MX8QuadMax. Even without using DVFS, the MultiPAWS decreased MPSoC temperature by 12 % compared to another method. Furthermore, we demonstrated how MultiPAWS could be used in design-space exploration for finding trade-offs between platform temperature and scheduling hyper-period. For example, we observed that when increasing the major frame length by 21 %, MultiPAWS reduced platform temperature by 10 %, which was 3 % better than plain rescaling of the hyper-period.

Acknowledgment

This work was supported by the THERMAC project, which has received funding from the European Union through the Clean Sky 2 Joint Undertaking, under the H2020 Framework Programme (H2020-CS2-CFP08-2018-01), grant agreements No 832011 and No 945535.

References

[1] DDC-I. (2021) Deos, a Time & Space Partitioned, Multi-core Enabled, DO-178C DAL A Certifiable RTOS. [Online]. Available: https://www.ddci.com/products_deos_do_178c_arinc_653/

[2] SYSGO, "PikeOS RTOS & Hypervisor." [Online]. Available: https://www.sysgo.com/pikeos

[3] Wind River, "VxWorks Safety Platforms." [Online]. Available: https://www.windriver.com/products/vxworks/safety-platforms

[4] NXP. (2021) i.MX 8QuadMax/QuadPlus Multisensory Enablement Kit. [Online]. Available: https://www.nxp.com/design/development-boards/i-mx-evaluation-and-development-boards/i-mx-8quadmax-multisensory-enablement-kit-mek:MCIMX8QM-CPU

[5] "DEmOS," Apr. 2021. [Online]. Available: https://github.com/CTU-IIG/demos-sched

[6] S. Pagani, H. Khdr, W. Munawar, J.-J. Chen, M. Shafique, M. Li, and J. Henkel, "TSP: Thermal safe power: Efficient power budgeting for many-core systems in dark silicon," in *Int'l Conf. on Hardware/Software Codesign and System Synthesis (CODES)*, 2014.

[7] J. Perez Rodriguez and P. Meumeu Yomsi, "Thermal-aware schedulability analysis for fixed-priority non-preemptive real-time systems," in *Int'l Conf. on Real-Time Systems Symposium (RTSS)*, 2019, pp. 154–166.

[8] M. Sojka, O. Benedikt, Z. Hanzálek, and P. Zaykov, "Testbed for thermal and performance analysis in MPSoC systems," in *Int'l Conf. on Computer Science and Information Systems (FedCSIS)*, 2020, pp. 683–692.

[9] S. Zhuravlev, J. C. Saez, S. Blagodurov, A. Fedorova, and M. Prieto, "Survey of energy-cognizant scheduling techniques," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 7, pp. 1447–1464, 2013.

[10] M. E. Gerards, J. L. Hurink, and P. K. Hölzenspies, "A survey of offline algorithms for energy minimization under deadline constraints," *J. of Scheduling*, vol. 19, no. 1, p. 3–19, Feb. 2016.

[11] M. Bambagini, M. Marinoni, H. Aydin, and G. Buttazzo, "Energy-aware scheduling for real-time systems: A survey," *ACM Trans. Embed. Comput. Syst.*, vol. 15, no. 1, Jan. 2016.

[12] J. Chen, A. Schranzhofer, and L. Thiele, "Energy minimization for periodic real-time tasks on heterogeneous processing units," in *Int'l Symp. on Parallel Distributed Processing*, 2009, pp. 1–12.

[13] "ARINC specification 653P1-2: Avionics Application Software Standard Interface, Part 1 – Required Services," 2006.

[14] "Avionics Application Software Standard Interface, Part 1, Required Services, ARINC Specification 653 Part 1 Supplement 4 (653P1-4)," 2015.

[15] P. Parkinson, "Update on using multicore processors with a commercial ARINC 653 implementation," in *Aviation Electronics Europe*, Apr. 2017. [Online]. Available: https://resources.windriver.com/i/841405/4

[16] H. Falk, S. Altmeyer, P. Hellinckx, B. Lisper, W. Puffitsch, C. Rochange, M. Schoeberl, R. B. Sørensen, P. Wägemann, and S. Wegener, "TACLeBench: A benchmark collection to support worst-case execution time research," in *Int'l Workshop on Worst-Case Execution Time Analysis (WCET)*, M. Schoeberl, Ed., vol. 55, 2016, pp. 2:1–2:10.

[17] D. V. Sokolov, "Tiny renderer or how OpenGL works: software rendering in 500 lines of code," https://github.com/ssloy/tinyrenderer, 2020.

[18] V. W. Freeh, D. K. Lowenthal, F. Pan, N. Kappiah, R. Springer, B. L. Rountree, and M. E. Femal, "Analyzing the energy-time trade-off in high-performance computing applications," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 6, pp. 835–848, 2007.

[19] M. Chrobak, C. Dürr, M. Hurand, and J. Robert, "Algorithms for temperature-aware task scheduling in microprocessor systems," in *Algorithmic Aspects in Information and Management*, R. Fleischer and J. Xu, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 120–130.

[20] C.-F. Kuo and Y.-F. Lu, "Task assignment with energy efficiency considerations for non-DVS heterogeneous multiprocessor systems," *ACM SIGAPP Applied Computing Review*, vol. 14, no. 4, p. 8–18, 2015.

[21] J. Zhou, T. Wei, M. Chen, J. Yan, X. S. Hu, and Y. Ma, "Thermal-aware task scheduling for energy minimization in heterogeneous real-time mpsoc systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 8, pp. 1269–1282, 2016.

[22] J. Zhou, J. Sun, P. Cong, Z. Liu, X. Zhou, T. Wei, and S. Hu, "Security-critical energy-aware task scheduling for heterogeneous real-time mpsocs in iot," *IEEE Transactions on Services Computing*, vol. 13, no. 4, pp. 745–758, 2020.

[23] A. Suyyagh and Z. Zilic, "Energy and task-aware partitioning on single-isa clustered heterogeneous processors," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 2, pp. 306–317, 2020.

[24] A. Rudi, A. Bartolini, A. Lodi, and L. Benini, "Optimum: Thermal-aware task allocation for heterogeneous many-core devices," in *2014 International Conference on High Performance Computing Simulation (HPCS)*, Jul. 2014, pp. 82–87.

[25] P. Antonante, J. Valverde-Alcalá, S. Basagiannis, and M. Di Natale, "Safe Implementation of Mixed-Criticality Applications in Multicore Platforms: A Model-Based Design Approach," in *Computer Safety, Reliability, and Security*, 2017, pp. 141–156.

[26] P. Han, Z. Zhai, and L. Zhang, "A model-based approach to optimizing partition scheduling of integrated modular avionics systems," *Electronics*, vol. 9, no. 8, 2020.

[27] A. Balsini, L. Pannocchi, and T. Cucinotta, "Modeling and simulation of power consumption and execution times for real-time tasks on embedded heterogeneous architectures," *ACM SIGBED Review*, vol. 16, no. 3, pp. 51–56, Nov. 2019.

[28] S. Han and H.-W. Jin, "Kernel-level ARINC 653 partitioning for Linux," in *Int'l Conf. on Applied Computing*, ser. SAC, 2012, pp. 1632–1637.

[29] C. Kown, D. Kim, H. Joe, and H. Kim, "Linux-based memory efficient ARINC 653 partition scheduler," in *Int'l Conf. on IEEE Emerging Technology and Factory Automation (ETFA)*, Sep. 2014, pp. 1–5.

[30] A. Dubey, G. Karsai, and N. Mahadevan, "A component model for hard real-time systems: CCM with ARINC-653," *Software: Practice and Experience*, vol. 41, no. 12, pp. 1517–1550, 2011.