A Multi-Level DPM Approach for Real-Time DAG Tasks in Heterogeneous Processors

Federico Reghenzani*[†¶], Ashikahmed Bhuiyan[‡§¶], William Fornaciari*, Zhishan Guo[‡]
*Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Italy
[†]ESTEC, European Space Agency, Netherlands
[‡]Department of Electrical and Computer Engineering, University of Central Florida, USA
[§]Department of Computer Science, West Chester University of Pennsylvania, USA

2021 IEEE Real-Time Systems Symposium (RTSS) | 978-1-6654-2802-6/21/$31.00 ©2021 IEEE | DOI: 10.1109/RTSS52674.2021.00014

Abstract—**The modeling and analysis of real-time applications focus on the worst-case scenario because of their strict timing requirements. However, many real-time embedded systems include critical applications requiring not only timing constraints but also other system limitations, such as energy consumption. In this paper, we study the energy-aware real-time scheduling of Directed Acyclic Graph (DAG) tasks. We integrate the Dynamic Power Management (DPM) policy to reduce the Worst-Case Energy Consumption (WCEC), which is an essential requirement for energy-constrained systems. Besides, we extend our analysis with tasks' probabilistic information to improve the Average-Case Energy Consumption (ACEC), which is, instead, a common non-functional requirement of embedded systems. To verify the benefits of our approach in terms of reduced energy consumption, we finally conduct an extensive simulation, followed by an experimental study on an Odroid-H2 board. Compared to the state-of-the-art solution, our approach is able to reduce the power consumption up to** 32.1%.

Index Terms—**Parallel Real-Time Tasks, Energy Minimization, Dynamic Power Management, Probabilistic Execution Time.**

I. Introduction

The current trend of embedded systems is to move towards high-performance multi-core architectures, including multiprocessor System-on-Chip. Many emerging computation-intensive real-time applications, such as self-driving cars, rely on parallel processing, i.e., they can simultaneously execute on multiple processors. The real-time community has studied the scheduling strategies for different parallel workload models, such as the *Directed Acyclic Graph (DAG)* task model [1]–[3], gang task model [4]–[6], and synchronous task model [7]. The parallel task model's fundamental characteristic, i.e., running simultaneously on multiple processors, is essential to exploit the computational power of modern multi-core platforms, including heterogeneous processors. The evolution and pervasiveness of Internet-of-Things (IoT) applications increase this need of powerful embedded systems, but often clash with energy consumption constraints.

The energy constraints. Many real-time embedded systems include critical applications requiring not only a predictable timing behavior, but also to satisfy other system constraints. Energy consumption is one of them, which might be a non-functional or functional requirement. In the former case, the design objective is to reduce as much as possible the energy consumption to avoid the cascade effect on the design of the overall system. Typical metrics used to evaluate such a requirement are the average power consumption and the *Average-Case Energy Consumption (ACEC)*. The second case, instead, is the case of *energy-constrained systems*, typically when the electrical source is an unstable energy harvester coupled with an energy-storage device, e.g., a battery. The possibility to perform battery re-charging during the operation is usually limited and dependent on external factors. Examples include satellites, devices located in remote regions, and medical equipment (e.g., pacemakers). Hence, along with the real-time performance requirements, such a platform must survive for a given time frame even in the absence of a stable power source. The energy-efficient design evolves from a nice-to-have feature to another critical requirement. In this case, the metric to be considered is the *Worst-Case Energy Consumption (WCEC)* [8]. The main component affecting ACEC and the WCEC is the processor's power consumption, which mainly comes from two sources, i.e., the switching activity and the leakage current. The former contributes to dynamic power consumption, and the latter is known as static power consumption. Depending on the platform, one source may dominate the other, as described in the next paragraph.

Energy management. There are two approaches for energy management, i.e., *Dynamic Voltage Frequency Scaling (DVFS)* and *Dynamic Power Management (DPM)*. They are commonly applied to reduce dynamic and static power consumption. DVFS adjusts the voltage and frequency of a processor during run-time in order to reduce the switching component of the power consumption. In the last decade, the dynamic power consumption was the dominant factor, and hence DVFS has been emphasized in research [9]. However, there is an exponential increase in the static power consumption with the transistor technology shift toward the sub-micron domains. In such a platform, static power becomes equal to dynamic power (e.g., in 90 nm high-end processor technology, leakage power consumes nearly half of the total power dissipation [10]). With the shift towards more dense manufacturing technologies, static power could be even more significant than dynamic power. Hence, the interest on DPM approaches is increasing, in order to exploit the idle interval and perform processor mode-

Corresponding author: Zhishan Guo, zsguo@ucf.edu.
¶ = Equal contribution.

978-1-6654-2802-6/21/$31.00 ©2021 IEEE
DOI 10.1109/RTSS52674.2021.00014

switching to reduce leakage power consumption.

Challenges. Applying DPM is a non-trivial problem, especially in hard real-time context: the transitioning between different power modes has additional overheads in terms of energy consumption and latency. Therefore, triggering the low power mode switch when the processor is idle is beneficial (for power saving) only if the idle slot is longer than a certain threshold, known as the break-even time [11]. Therefore, for DPM to be beneficial, it is critical to efficiently decide whether or not performing power mode transitioning. This challenge is amplified when dealing with heterogeneous processors and parallel DAG tasks. The power mode switch decision can be taken at run-time, depending on the actual system condition, while an offline analysis is required to verify the ACEC and/or WCEC requirements. To perform such analysis we have to rely on either the probabilistic information of the task execution time (for ACEC requirements) or the *Worst-Case Execution Time (WCET)* information (for WCEC requirements).

Contribution. Considering the parallel DAG task model, we study how to integrate the DPM approach to reduce CPU[1] energy consumption. We calculate the break-even points for heterogeneous processors and model the idle intervals to find the best possible task allocation. Our approach also guarantees hard real-time requirements by taking into account the DPM overheads, given a statically computed WCET of the tasks, even in the ACEC optimization case. Specifically, the key contributions are:

1) We propose a resource management strategy to allocate the DAG nodes to a set of heterogeneous processors equipped with DPM and, in particular, with the C-States mechanism (subsequently explained in Section III-B), that satisfies the timing requirements.
2) We perform an analysis to find the break-even points that maximize the benefit of the DPM technique. Such values are then exploited both online to optimize the energy at run-time, and offline to compute the WCEC of the tasks for verifying system requirements.
3) We present a probabilistic analysis to calculate and optimize the ACEC and verify the common-case energy requirements, while still guaranteeing the hard timing constraints.
4) Finally, the proposed techniques are applied in a simulation environment to test different platform/task set configurations, followed by a small-scale experimental evaluation on a real platform, which will strengthen the results' confidence.

As later presented in Section II, this work advances the state-of-the-art by providing comprehensive models and DPM policies able to exploit modern system properties (heterogeneous architectures, multi-level DPM) and different requirements (optimization of ACEC or WCEC), while guarantee the strict hard real-time requirements of parallel DAG tasks.

Paper organization. The rest of the paper is organized as follows. We present the related work in Section II. Section III describes all the considered models and the problem, and it provides to the reader the necessary background. Section IV presents the computation of the break-even points, which are used in the subsequent Section V to derive the best allocation and the WCEC value. This analysis is extended with probabilistic information to obtain the ACEC in Section VI, and the simulation and experimental results are respectively presented in Section VII and Section VIII. We conclude the paper in Section IX with some future research directions.

II. Related Work

To date, many previous works have studied the energy/power optimization for sequential (no intra-task parallelism) tasks in both single-core and multi-core platforms (refer to [12] for a comprehensive survey). However, the sequential task model does not allow a single task to execute at multiple cores simultaneously. Optimizing the energy in a parallel context differs significantly from the sequential tasks' approaches. Although there is much work proposed on the RT community that studied the real-time scheduling analysis of the parallel task model [13]–[19], none of them have considered the energy-awareness.

To our knowledge, a limited number of works studied the energy-aware scheduling strategy of the real-time parallel tasks, especially of the DAG model. Zhu et al. [20], [21] proposed an energy-aware scheduling policy that utilizes slacks between the inter-dependent sequential tasks. Both the work by Bhuiyan et al. [22] and Guo et al. [3] considered a simplified model (e.g., the number of cores are unlimited, the entire schedule until the hyper-period available a priori) to propose the energy-aware real-time scheduling of DAG tasks. Based on the DAG task model, some recent works have studied the energy-aware scheduling in a homogeneous and heterogeneous clustered platforms [23]–[25]. The work by Saifullah et al. [26] studied the CPU energy optimization of the DAG task considering the federated scheduling policy. Considering the gang task model in a homogeneous platform, Paolillo et al. [27] studied the energy-aware scheduling malleable gang jobs. All these work as above-mentioned restricted their attention to the DVFS policy to reduce energy consumption and do not take into account the DPM policy.

Gerards et al. [28] studied the energy-aware scheduling of the frame-based real-time tasks considering both the DPM and DVFS approach. However, they did not consider the time overhead of the DPM policy while switching processor execution mode. Besides, they have restricted their attention to a single-core platform. Esmaili et al. [29] proposed an approach for modeling idle intervals in MPSoC platforms. Huang et al. [30] proposed a DPM policy optimizing the energy consumption and by using time-triggered scheduling, but limited the discussion to ACEC and homogeneous processors. Compared to the previous works, this paper advances the

[1] There are other components (e.g., I/O devices, cache, system bus) that contribute to the overall energy consumption. However, in this work, we focus on CPU, because it is one of the major contributors to the overall system energy consumption and the more challenging with respect to timing requirements.

energy-aware intra-task scheduling by proposing a multi-level DPM policy in a heterogeneous context, analyzing both the ACEC and WCEC cases, while guaranteeing hard real-time requirements.

III. System Model, Problem Statement and Background

A. Workload Model

In this work, we consider a set of sporadic parallel DAG tasks denoted by $\tau = \{\tau_1, \tau_2, \cdots, \tau_n\}$. For each task $\tau_i \in \tau$ $(1 \leq i \leq n)$, the minimum inter-arrival time, relative deadline, and WCET[2] are respectively denoted by $T_i, D_i,$ and C_i. We assume that each task is an implicit deadline task, i.e., $D_i = T_i$. A task τ_i consists of total of N_i nodes, each denoted by $\mathcal{N}_i^l (1 \leq l \leq N_i)$. Each node $\mathcal{N}_i^l$ has its worst-case execution requirement C_i^l. If the nodes are executed on a single unit-speed processor, then $\sum_{l=1}^{N_i} C_i^l = C_i$. There exist precedence constraints among the nodes in a DAG, which is represented by a directed edge among the nodes. For example, in Figure 1(a), $\mathcal{N}_i^2 \rightarrow \mathcal{N}_i^3$ implies that $\mathcal{N}_i^3$ can not start execution if $\mathcal{N}_i^2$ is still executing. In this case, $\mathcal{N}_i^2$ is called the parent of $\mathcal{N}_i^3$, while $\mathcal{N}_i^3$ is a child of $\mathcal{N}_i^2$. A node may have multiple parents or children, e.g., $\mathcal{N}_i^4$ has two parents ($\mathcal{N}_i^1, \mathcal{N}_i^2$), and $\mathcal{N}_i^1$ has two children ($\mathcal{N}_i^4, \mathcal{N}_i^5$). If a node has multiple parents, it can start the execution only when all of its parents finish their execution. The *maximum degree of parallelism* of task τ_i is denoted by M_i, which is the maximum number of nodes that execute in parallel, at any time.

A *critical path* in a DAG task is a directed path that has the maximum total execution requirements. The length of the critical path, L_i, is the sum of all the nodes' execution requirements on a critical path. For instance, in Figure 1(a), $\mathcal{N}_i^1 \rightarrow \mathcal{N}_i^4 \rightarrow \mathcal{N}_i^6$ and $\mathcal{N}_i^1 \rightarrow \mathcal{N}_i^5 \rightarrow \mathcal{N}_i^6$ are the critical paths, and the critical path length is 10. Note that, L_i is the minimum execution time of task τ_i even when the task gets exclusive access in an infinite number of cores. Hence, to ensure that τ_i is schedulable, the condition $T_i \geq L_i$ must hold.

B. Platform and Energy Model

We consider a set of heterogeneous processors $\{p_1, p_2, ..., p_m\}$ (including processors with an homogeneous architecture but configured with different frequency scaling). To each processor and workload we statically assign a coefficient called *execution speed* $S_{i,j}^l = \frac{C_i^l}{C_{i,j}^l}$, where $C_{i,j}^l$ represents the amount of time required to execute the node $\mathcal{N}_i^l$ on the processor p_j. The value $C_{i,j}^l$ depends on the processor's architecture and DVFS configuration. The nodes are assumed to be profiled on each processor and the $C_{i,j}^l$ computed offline. If, for any reason, a node of a task cannot be executed on a particular processor, then the time is set to $C_{i,y}^x = \infty$. The processors are capable of performing the DPM. Different DPM strategies have been implemented in the last decades, and the most common is to select between *running* and *sleeping* state. However, modern processors are capable to perform multi-level DPM, i.e., they can switch to different levels of power saving. We generalize them with the concept of *C-States*. Each processor, at a given instant, can be in one of the following power saving states: $\{\Theta_0, \Theta_1, ..., \Theta_r\}$. The state Θ_0 represents the condition when the processor is active and running the tasks. The states $\Theta_1, ..., \Theta_r$ represent, instead, the power-saving states, i.e., when the processor is (partially) shut down and not running the workload. The larger the index of the C-State, the more aggressive the power saving technique is and, consequently, the larger the overhead to switch back to Θ_0. An example for Intel processors is shown in Table I. To each processor p_k is mapped the power consumption $\mathcal{C}^{j,k}$ and the overhead $T_{sw}^{j,k}$ of the j-th C-State. The value of $\mathcal{C}^{j,k}$ depends on many external factors (such as temperature) but it is usually provided by the manufacturer (in worst- and/or common-case scenario) or experimentally measured under different execution conditions. The overhead $T_{sw}^{j,k}$ is the amount of time required to wake-up the p_k processor from the C-State Θ_j to the C-State Θ_0. During this wake-up process, the system consumes an energy overhead identified by $E_{sw}^{j,k}$. The values $T_{sw}^{0,k}$ and $E_{sw}^{0,k}$ have no meaning, but for the sake of the following notation we consider them to be $T_{sw}^{0,k} = E_{sw}^{0,k} = 0$. If such information are available, then we do not need to specify a power model to use the approach proposed in this article.

TABLE I
An example of C-States typical of the Intel processors from the 2010s onwards. [31].

C-State	Θ_0	Θ_1	Θ_2	Θ_3
Core voltage	ON	ON	ON	OFF
Core clock	ON	OFF	OFF	OFF
Core PLL	ON	ON	OFF	OFF
L1/L2 caches	Keep	Keep	Flush	Flush
Wake-up time	-	LOW	MEDIUM	HIGH
Wake-up energy	-	LOW	MEDIUM	HIGH
Idle power	MAX	MEDIUM	LOW	VERY LOW

Remark 1. *The heterogeneity is visible in the platform description: each processor has a different speed (for a particular workload), different power consumption in each C-State, and timing and energy overheads. We do not target a specific "heterogeneity", provided that the previous values are known or can be measured.*

C. Real-Time DAG Task Decomposition

Task decomposition is a well-known technique proposed by Saifullah et al. [13] that transforms the nodes of a parallel DAG task τ_i into a set of sequences of nodes, each one possible to run in parallel (refer to Figure 1). Initially assuming that at least M_i cores are available for each task, for every node $\mathcal{N}_i^l$, a vertical line is drawn at every time instant where $\mathcal{N}_i^l$ starts or ends. These vertical lines partition the DAG into several segments $t_i^1, t_i^2, \ldots$. In this manner, while respecting the node dependencies (i.e., edges in the DAG), task decomposition converts each node $\mathcal{N}_i^l \in \tau_i$ into an individual sub-task, and

[2] The WCET of the tasks is assumed as computed statically by considering a single unit-speed processor.

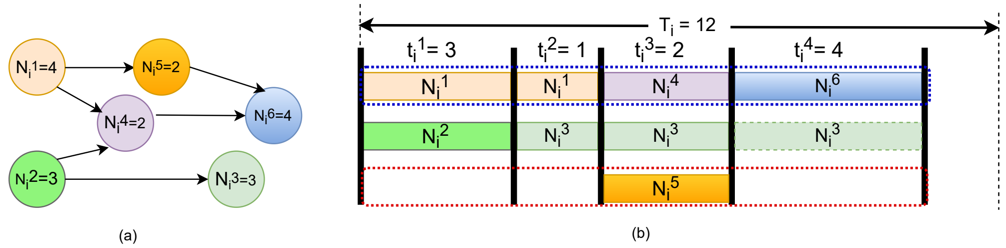

Fig. 1. A task represented with (a) the DAG model, and (b) an example of its decomposed structure (after applying task decomposition).

also defines a *scheduling window* (for each node $\mathcal{N}_i^l$), which denotes the time slot from the release offset of $\mathcal{N}_i^l$ to its deadline, measured in segments.

Example 1. *Consider the decomposed DAG task τ_i in Figure 1(b). Task decomposition converts each node $\mathcal{N}_i^l$ to an individual sub-task with a start time and a deadline. In this example, the scheduling windows for the nodes, i.e., $\mathcal{N}_i^1, \mathcal{N}_i^2, ...\mathcal{N}_i^6$, are [1, 2], [1, 1], [2, 4], [3, 3], [3, 3], [4, 4], respectively. This means that $\mathcal{N}_i^1$ executes from the beginning of the 1^{st} segment to the end of the 2^{nd} segment, while $\mathcal{N}_i^2$ executes throughout the 1^{st} segment. The DAG task τ_i of Figure 1 has a maximum degree of parallelism of $M_i = 3$ in segment t_i^3, where the nodes $\mathcal{N}_i^3, \mathcal{N}_i^4$, and $\mathcal{N}_i^5$ execute in parallel.*

Intra-Task Processor Merging. Once a DAG task τ_i is decomposed and allocated to multiple processors, likely, some of these processors are lightly loaded. This situation may not be optimal, from both the resource utilization and energy consumption standpoints. Even worse, the number of processors m may be lower than the number of levels, making the allocation unfeasible. To get rid of this problem, Guo et al. [3] proposed the intra-task merging technique that merges those lightly loaded processors, while guaranteeing the timing constraints' satisfaction. Intra-task processor merging reduces the number of required cores and, consequently, decreases the leakage power consumption, resulting in overall energy and resource efficiency.

The schedulability goal of this paper is to ensure that no sub-tasks exceed the scheduling window time frames output of the task decomposition. In fact, provided that the task decomposition is correctly performed, if all the sub-tasks meet their deadlines at the end of each scheduling window – maintaining constant in this way the task WCET –, the schedulability of the whole task set depends only on the scheduling algorithm applied at the task-level. Our approach focuses on analyzing the DAG of the single task without restricting the choice of the scheduling algorithm among the tasks. In particular, we ensure the intra-task timing constraints, i.e. the DAG nodes, while an external scheduling algorithm is in charge of the inter-task scheduling. We summarized the key notations presented in this section in Table II.

D. The Optimization Problem

This work focuses on minimizing the total energy consumption at the task-level, including the energy consumption when the processor is idle. The energy optimization strategy must also meet the strict timing requirements under all circumstances. The minimization operates on the processor allocation of each node and on the decision if switching or not the processor to a higher C-State during idle times. The choice of a DPM-only approach instead of DVFS is motivated by [3, Theorem 2], which asserted that selecting a fixed speed is beneficial with respect to energy assumptions, over a dynamic approach, for real-time DAG tasks.

TABLE II
SUMMARY OF KEY MODEL NOTATIONS.

	Symbol	Description
Workload	τ_i	The i-th task
	T_i, C_i, D_i	Period, WCET, and relative deadline of the task τ_i
	$\mathcal{N}_i^l, N_i$	The l-th node and the total number of nodes of τ_i
	M_i	the maximum level of parallelism for task τ_i
	c_i^l	The WCET of the node $\mathcal{N}_i^l$
	b_i^l, d_i^l	Start time and duration of node $\mathcal{N}_i^l$
	L_i	Length of the critical path for task τ_i
	t_i^z	Length of the z-th segment of task τ_i
Platform	p_i	The i-th processor
	$S_{i,j}^l$	The execution speed for processor p_j and workload $\mathcal{N}_i^l$
	Θ_i	The i-th C-state
	$\mathcal{C}^{j,k}$	Power consumption of the processor p_k in Θ_j
	$T_{sw}^{j,k}, E_{sw}^{j,k}$	Time and Energy overhead to switch from Θ_j to Θ_0 in p_k

The offline analysis allows us to perform the same optimization with two different goals: the WCEC or the ACEC minimization. The former involves the use of the WCET of the nodes to carry out the best optimization in the worst-case scenario, while the latter exploits probabilistic information on the node execution time to compute (and minimize) the average-case energy consumption. While the former targets the specific case of energy-constrained devices as described in Section I, the latter is oriented to a wider class of embedded systems in which the energy consumption reduction is a non-functional requirement but a desired feature.

IV. Modeling Idle Intervals and their Energy Consumption

To incorporate the DPM approach in DAG tasks we need to model the idle intervals at the end of each node. In Subsection IV-A, we propose an approach to compute the idle intervals. In Subsection IV-B, we provide a discussion regarding the energy consumption during each idle interval and the task period.

A. Modeling the Idle Interval

In the DPM approach, the processors enter a deep sleep state when idle and wake up when necessary. DPM is a useful tool in decreasing system energy consumption without degrading the performance. However, the beneficial employment of DPM is a non-trivial problem. The main reason is the non-negligible transition overhead (in terms of time and energy) between sleep and wakeup state [32], [33]. It is essential to know the processor idle interval, which leads to achieving the right strategy for the successful employment of the DPM approach. **Calculating the idle interval before each node**. Upon applying the existing task decomposition and intra-task processor merging techniques (refer to [3], [13] and Subsection III-C), some essential information (e.g., the maximum degree of parallelism of the task, the required number of cores) becomes available. Recall that task decomposition is a technique to convert each node, $\mathcal{N}_i^l \in \tau_i$, into an individual sub-task with its own release time, deadline, and execution requirement, without violating their precedence constraints. At each level, there might exist a single node or multiple nodes. For example, in Figure 1, $\mathcal{N}_i^1, \mathcal{N}_i^4$, and $\mathcal{N}_i^6$ execute at level 1 (bordered inside the dotted blue rectangle), and $\mathcal{N}_i^5$ executes at level 3 (bordered inside the dotted red rectangle).

For such a decomposed task τ_i, we are aware of the optimal execution speed for each node $\mathcal{N}_i^l \in \tau_i$, which also guarantee the real-time correctness, i.e., τ_i does not miss its deadline under any circumstances. In the assignment of a processor to each level, we satisfy the speed requirements of each node in this level, i.e., the assigned processor executes at speed greater or equal to each node's energy-aware execution speed. For details, refer to Section V and Eq. (7), where we propose a processor-task allocation approach that minimizes energy consumption. Hence, each node finishes its execution earlier and leave some *idle slot* within its scheduling window.

Before moving further into the details, we define some notations used several times throughout this paper. We denote the start time and the duration of a node $\mathcal{N}_i^l \in \tau_i$ at processor p_k as $b_i^{l,k}$ and $d_i^{l,k}$, respectively, where $d_i^{l,k}$ is defined as $C_i^l/S_{i,k}^l$. The speed of S_k of the processor p_k is given and constant. We define a decision variable called $\mathcal{O}_{k,\mathcal{N}_i^l,\mathcal{N}_i^m}$ that represents node execution order at p_k processor [29]. We define $\mathcal{O}_{k,\mathcal{N}_i^l,\mathcal{N}_i^m}$ as:

$$\mathcal{O}_{k,\mathcal{N}_i^l,\mathcal{N}_i^m} = \begin{cases} 1 & \text{if } \mathcal{N}_i^l \text{ is scheduled immediately before} \\ & \mathcal{N}_i^m \text{ at } p_k \text{ processor} \\ 0 & \text{otherwise} \end{cases}$$

We use the notation $\mathcal{O}_{k,0,\mathcal{N}_i^l}$ to denote whether $\mathcal{N}_i^l$ is the first node to be scheduled at p_k processor, at any inter-arrival period of a job of task τ_i. Similarly, $\mathcal{O}_{k,\mathcal{N}_i^m,\mathcal{J}}$ denotes that $\mathcal{N}_i^m$ is the last node (at any inter-arrival period of a job of task τ_i) to be scheduled at p_k processor. Now we can model idle intervals for all the nodes that are allocated to the p_k processor, thanks to the decision variable, $\mathcal{O}_{k,\mathcal{N}_i^l,\mathcal{N}_i^m}$. Note that the start time of node $\mathcal{N}_i^m$ depends on other nodes (if it has multiple parents) executing on a different processor. However, the task decomposition technique determines the starting time of each node respecting their precedence constraints, refer to Figure 1(b). Here we concern only about the parent node $\mathcal{N}_i^l$ that shares the same processor with $\mathcal{N}_i^m$ as the completion time of this parent node influences the idle time available before node $\mathcal{N}_i^m$ starts execution.

We consider the following two cases (to model the idle interval before executing node $\mathcal{N}_i^m$ at p_k processor) in each period:

Case-1: $\mathcal{N}_i^m$ *is not the first node to be served at processor* p_k (e.g., node $\mathcal{N}_i^6$ at the topmost level in Figure 1(b)). Let the total amount of idle time immediately before scheduling $\mathcal{N}_i^m$ (at processor p_k) be $\mathcal{I}_i^{m,k}$ and calculated as follows:

$$\mathcal{I}_i^{m,k} = b_i^{m,k} - \sum_{\forall l:\mathcal{N}_i^l \in \tau_i} (b_i^{l,k} + d_i^{l,k}) \cdot \mathcal{O}_{k,\mathcal{N}_i^l,\mathcal{N}_i^m} \qquad (1)$$

Case-2: $\mathcal{N}_i^m$ *is the first node to be served at processor* p_k (e.g., node $\mathcal{N}_i^5$ at the bottom-most level in Figure 1(b)). In this case $\mathcal{I}_i^{m,k}$ is:

$$\mathcal{I}_i^{m,k} = b_i^{m,k} - \sum_{\forall l:\mathcal{N}_i^l \in \tau_i} (b_i^{l,k} + d_i^{l,k}) \cdot \mathcal{O}_{k,\mathcal{N}_i^l,\mathcal{J}} \qquad (2)$$

Where the second term denotes the completion time of the last node of task τ_i (at p_k processor) in the previous period. By subtracting it from $b_i^{m,k}$, we calculate the idle time (before serving this node) in this period. Because the goal of our approach is to optimize the single-task energy consumption, we assumed that the previous job running on the processor is a job of the same task τ_i. In the other case, depending on the task scheduling algorithm, we have three options: 1) replace τ_i in Eq. (2) with the previous task, provided that the schedule is predictable; 2) assume the idle time only from the beginning of the task, i.e., $\mathcal{I}_i^{m,k} = b_i^{m,k} - \min_x b_i^{x,y}$; 3) use, even so, the single-task idle time of Eq. (2). The last two solutions are both sub-optimal but safe to use. Which one is preferable depends on the specific case and considered task scheduling algorithm.

B. Energy Consumption During an Idle Interval

Having incorporated the DPM in our model, we now study the energy consumption the system incurs switching to and from the sleep mode. The switching to the sleep mode is not beneficial (w.r.t energy consumption) if the idle interval is smaller than a threshold [33]. This idle interval is known as the *break-even time* [11], [34]. Esmaili et al. [29] calculated the break-even time as follows: $T_{BE} = \max(T_{sw}, \frac{E_{sw}}{\mathcal{C}})$. We extend this notation by introducing a set of break-even times for each C-State Θ_j $(1 \le j \le r)$ and for each processor p_k:

$$T_{BE}^{j,k} = \max\left(T_{sw}^{j,k}, T_E^{j,k}\right) \qquad (3)$$

where:

$$T_E^{j,k} = \frac{E_{sw}^{j,k} - E_{sw}^{j-1,k} - \mathcal{C}^{j,k}T_{sw}^{j,k} + \mathcal{C}^{j-1,k}T_{sw}^{j-1,k}}{\mathcal{C}^{j-1,k} - \mathcal{C}^{j,k}} \qquad (4)$$

The break-even point formula has been derived as follows:

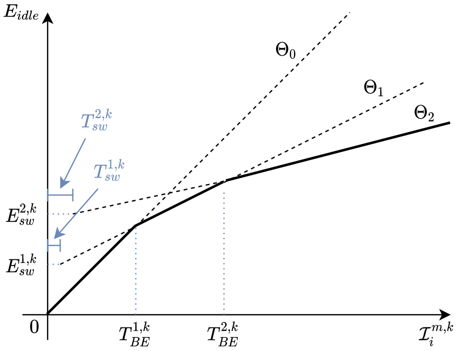

Fig. 2. The graphical interpretation of the break-even points of an example with three C-states and $\mathcal{C}^{0,k} = 1$, $\mathcal{C}^{1,k} = 0.5$, $\mathcal{C}^{2,k} = 0.25$, $\mathcal{E}_{sw}^{1,k} = 5$, $\mathcal{E}_{sw}^{2,k} = 10$, $\mathcal{T}_{sw}^{1,k} = 1$, $\mathcal{T}_{sw}^{1,k} = 3$.

- The break-even time cannot be lower than the time required for switching, otherwise we can possibly invalidate the timing requirements: if the maximum idle time is smaller than the time required for switching, the overhead of the C-State switch delays the activation of the following node.
- Since the C-State switch has an overhead also in terms of energy ($E_{sw}^{j,k}$), the switch is not advantageous for smaller values of idle time. The graphical representation has been depicted in Figure 2. The energy consumption as a linear function of time for a C-State Θ_j is $\mathcal{C}^{j,k} \cdot t$, then the intersection between the energy consumption line of Θ_{j-1} and Θ_j gives us the break-even point for the switching from Θ_{j-1} to Θ_j. The break-even point is then the solution x of the equation: $E_{sw}^{j-1,k} + \mathcal{C}^{j-1,k}(x - T_{sw}^{j-1,k}) = E_{sw}^{j,k} + \mathcal{C}^{j,k}(x - T_{sw}^{j,k})$.

To model the CPU energy consumption during an idle interval, we proceed by extending the original function for E_{idle} [29], obtaining:

$$E_{idle}(\mathcal{I}_i^{m,k}) = \begin{cases} \mathcal{C}^{0,k}\mathcal{I}_i^{m,k} & \text{if } 0 \leq \mathcal{I}_i^{m,k} < T_{BE}^{1,k} \\ E_{sw}^{1,k} + \mathcal{C}^{1,k}(\mathcal{I}_i^{m,k} - T_{sw}^{1,k}) & \text{if } T_{BE}^{1,k} \leq \mathcal{I}_i^{m,k} < T_{BE}^{2,k} \\ ... & \\ E_{sw}^{k,r} + \mathcal{C}^{r,k}(\mathcal{I}_i^{m,k} - T_{sw}^{r,k}) & \text{if } T_{BE}^{r,k} \leq \mathcal{I}_i^{m,k} \end{cases} \tag{5}$$

In this formula, $T_{BE}^{i,k}$ denotes the break-even times previously computed by Eq. (3), and $\mathcal{I}_i^{m,k}$ denotes the total idle time before executing $\mathcal{N}_i^m \in \tau_i$, and $\mathcal{I}_i^{m,k}$ is derived from Eq. (1) and Eq. (2). Finally, total energy consumption during all the idle intervals in a task-period, T_i, is calculated as follows:

$$E_{idle}(T_i) = \sum_{\forall l : \mathcal{N}_i^l \in \tau_i} E_{idle}(\mathcal{I}_i^{l,k}) \tag{6}$$

In Section V we exploit this information of $E_{idle}(\mathcal{I}_i^{m,k})$ to derive an offline minimization strategy for the WCEC. While, in Section VI, we introduce probabilistic information to use the same strategy for computing, and minimizing the ACEC. However, it should be noted that the decision taken with Eq. (5) can be also applied online for a further optimization: The idle time can be measured at the end of each node and compared with the offline-computed break-even points. The decision can, in this way, change online and be different with respect to the decision used for WCET and ACEC analyses, further improving the energy consumption for each specific case of execution. However, the following offline analyses are still essential to verify that the system adhere with the WCEC and ACEC design requirements.

TABLE III
THE ENERGY CONSUMPTION TABLE, SHOWING THE RELATION BETWEEN THE PROCESSOR $p_k \in p$ WHEN p_k IS ALLOCATED TO A LEVEL $\mathcal{L}_j^i \in \mathcal{L}^i$.

Processors→ Levels↓	p_1	p_2	$\cdots$	p_k
$\mathcal{L}_1^i$	E_1^1	E_1^2	$\cdots$	E_1^k
$\mathcal{L}_2^i$	E_2^1	E_2^2	$\cdots$	E_2^k
$\vdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$
$\mathcal{L}_j^i$	E_j^1	E_j^2	$\cdots$	E_j^k

V. Efficient Processor Allocation Algorithm

Our energy management strategy is composed of two processes. *First,* identify the idle slot while executing a task τ_i and employ the DPM technique, i.e., decide whether to switch to a higher C-State during the idle slots to minimize energy consumption. *Second,* allocate a set of cores to the DAG task τ_i to minimize overall energy consumption while satisfying the strict schedulability requirements. In Section IV, we have already discussed how to identify the idle interval (while executing a task) interval and calculate the energy consumption during this idle interval. In this section, we discuss our second goal, i.e., allocating a set of cores to the nodes of the DAG task τ_i to reduce energy consumption. We apply the Hungarian algorithm [35] to the decomposed DAG to find an energy-conserving task-processor allocation. Although the DAG decomposition technique or the Hungarian algorithm itself is not new, analyzing the energy optimization by exploiting the idle slots is sufficiently novel. Finally, we discuss how to calculate the overall energy consumption throughout the task period of T_i (of task τ_i), combining both the analyses presented in Section IV and in Section V.

Processor to task allocation. Let, $\mathcal{L}^i$ denotes the set of levels of task τ_i when decomposed. In our task allocation approach, we allocate one processor p_k exclusively at each level $\mathcal{L}_j^i \in \mathcal{L}^i$; hence, all the nodes at the same level execute at the same speed throughout their release to their deadline. If the processor speed is known, we can easily calculate the energy consumption, as we know the information regarding the release time, deadline, and execution requirement of each node executing at each level. Note that we can allocate a processor to any level, provided that the speed of the processor is sufficient to finish all the nodes within its scheduling window. Hence, at any level $\mathcal{L}_j^i \in \mathcal{L}^i$, we can allocate a processor $p_k \in p$, if it satisfies the following constraints:

Algorithm 1 Building the energy consumption table.

```
1: Input: Set of processors $p = \{p_1, \dots, p_k\}$ and the number of levels $\mathcal{L}^i = \{\mathcal{L}^i_1, \dots, \mathcal{L}^i_j\}$ in a DAG task $\tau_i$.
2: Output: A table that store the energy consumption value.

3: $\mathcal{E}[j][k]$;    /* A table to store energy consumption */
4: Set $\infty$ to $10^6$    /* An arbitrary large value */
5: for $x = 1$ to $j$ do
6:    for $y = 1$ to $k$ do
7:       /* Verify speed-constraints in Eq. (7) */
8:       if $\forall \mathcal{N}^l_i \in \mathcal{L}^i_x : \frac{C^l_i}{\sum_{z=b^l_i}^{f^l_i} t^z_i} \leq S_y$ then
9:          Calculate $E^k_j$ according to Eq. (8)
10:         $\mathcal{E}[x][y] = E^k_j$;
11:      else
12:         $\mathcal{E}[x][y] = \infty$;
13:      end if
14:   end for
15: end for
16: return $\mathcal{E}$.
```

Algorithm 2 Processor to task allocation.

```
1: Input: Energy consumption table $\mathcal{E}$.
2: Output: Processor to level allocation that results minimum energy consumption.
3: Set $\infty$ to $10^6$    /* An arbitrary large value */
4: /* If $\mathcal{E}$ is not a square table, add dummy row to make it square size */
5: if $j < k$ then
6:    for $x = j + 1$ to $k$ do
7:       for $y = 1$ to $k$ do
8:          $\mathcal{E}[x][y] = \infty$;    /* Dummy row */
9:       end for
10:   end for
11: end if
12: Solve $\mathcal{E}$ using the Hungarian algorithm [35].
13: return the optimal processor to level allocation.
```

$$\forall \mathcal{N}^l_i \in \mathcal{L}^i_j : \frac{C^l_i}{\sum_{z=b^l_i}^{f^l_i} t^z_i} \leq S^l_{i,k} \tag{7}$$

Here C^l_i, b^l_i, and f^l_i respectively denotes the execution requirement, release offset and deadline of node $\mathcal{N}^l_i$ (see Example 1), t^z_i denotes the length of segment z, and $\sum_{z=b^l_i}^{f^l_i} t^z_i$ denotes the available execution slot for a specific node. Refer to Example 1 and Figure 1(b) for details. Let us assume that we allocate a processor p_k in level $\mathcal{L}^i_j$, and denote the energy consumption (by processor p_k) as E^k_j, where we calculate E^k_j as follows:

$$E^k_j = E_{idle}(T(\mathcal{L}^i_j)) + E_{active}(T(\mathcal{L}^i_j)) \tag{8}$$

where

$$E_{idle}(T(\mathcal{L}^i_j)) = \sum_{\forall \mathcal{N}^l_i \in \mathcal{L}^i_j} E_{idle}(\mathcal{I}^{l,k}_i), \text{ and}$$

$$E_{active}(T(\mathcal{L}^i_j)) = \sum_{\forall \mathcal{N}^l_i \in \mathcal{L}^i_j} d^{l,k}_i \mathcal{C}^{0,k}$$

Here, $T(\mathcal{L}^i_j)$ denotes the duration of level $\mathcal{L}^i_j$. Note that, there may be some idle time available at the end of each level (since the critical path length L_i is less than or equals to task period T_i), task decomposition distributes such idle slot, i.e., $T_i - L_i$, uniformly by multiplying each segment by a common factor T_i/L_i (refer to Section III in [3]). Hence, we can conclude that $\forall j : T(\mathcal{L}^i_j) = T_i$.

We calculate E^k_j ($\forall \mathcal{L}^i_j \in \mathcal{L}^i, p_k \in p$) according to Eq. (8) and Algorithm 1, storing these information in a two dimensional array $\mathcal{E}$ represented in Table III. Algorithm 1 starts by creating $\mathcal{E}$ of size $j \times k$, where $j \leq k$ (Line 3), and traverses each level $\mathcal{L}^i_j \in \mathcal{L}^i$ and each processor $p_k \in p$ (Lines 5-6). Then it checks whether a processor can be assigned to a level (Line 8), i.e., speed of this processor satisfies the speed constraint presented in Eq. (7). If it satisfies the speed constraints, we store the energy consumed by this processor (when allocated to this level) at the corresponding cell (Line 9). Otherwise, we put an arbitrarily large value to this cell (Line 11). Algorithm 1 concludes by returning $\mathcal{E}$ (Line 15). Here, each entry $E^k_j \in \mathcal{E}$ denotes the energy consumed by processor p_k, when p_k is allocated to level $\mathcal{L}^i_j$. For any level $\mathcal{L}^i_j \in \mathcal{L}^i$, if any processor $p_k \in p$, fails to satisfy the constraints in Eq. (7), we put an arbitrarily large value in the corresponding cell. We do this to ensure that the scheduler does not assign p_k to $\mathcal{L}^i_j$, i.e. to a level which does not satisfy the timing requirements.

Now we know the energy consumption at all possible processor to level mapping combinations. We use this information to determine the processor allocation with minimum energy consumption. At each level, we assign a processor that is not allocated to any other level previously – we pick a single entry from each row and column in Table III. The pseudocode is presented in Algorithm 2. We determine the optimum assignment that minimizes the total energy consumption using the *Hungarian algorithm* [35], [36] (Line 12), which has polynomial complexity. Note that the Hungarian algorithm works only when the input is an $N \times N$ square matrix with non-negative elements. In our case, Table III may not be square in size as $j \leq k$. Hence, we add $(k - j)$ extra dummy rows in Table III (to make it square in size), and fill them with arbitrary large values (Lines 5–11). Finally, Algorithm 2 concludes by returning the optimal processor to task allocation that results in minimum energy consumption.

Total Energy Consumption. Having computed the energy consumption at each level and the optimal processor allocation, let the energy consumption at each level $\mathcal{L}^i_j$ (after assigned with the processor determined by Algorithm 2) be denoted by E^i_j. The total energy consumption E^i_{tot} of the task τ_i is:

$$E^i_{tot} = \sum_{\forall \mathcal{L}^i_j \in \mathcal{L}^i} E^i_j \tag{9}$$

Optimality. The use of the Hungarian algorithm guarantees the energy consumption optimality of the processor to level allocation. The optimality of the overall solution depends then on the selection of DPM states, which is optimal by construction for a given DAG decomposition. In fact, Eq. (5) minimizes the energy consumption according to the idle times $\mathcal{I}_i^{m,k}$ which in turns depends on the DAG decomposition and on how intra-task processor merging is performed. Therefore, we can claim the optimality of the offline algorithm for a given DAG decomposition, while the overall optimality depends on the algorithms used to build the DAG decomposition.

VI. Exploiting Probabilistic Information to Model the Average-Case

The analysis presented in the previous two sections was performed by considering the WCET: the definition of duration of a node ($d_i^{l,k}$) was defined as C_i^l/S_k. Instead, in this section, we consider the duration of a node as a function of the random variable[3] of the execution time $\tilde{X}_i$, i.e., $\tilde{d}_i^{l,k} = \tilde{X}_i^l/S_k$. This random variable is used, as subsequently detailed, to optimize the ACEC instead of the WCEC.

Remark 2. *The probabilistic information is used for energy optimization only, and the hard real-time guarantees are not affected by such information. In this way, any inaccuracy in the probabilistic characterization of the execution time would impact the energy optimization problem only, but not the schedulability analysis, that still relies on the static WCET.*

Alternatively, the statically-computed WCET can be replaced by the probabilistic-WCET [37]. In a such a case, the probabilistic-WCET can be used to compute the probabilistic-WCEC and, in turn, the WCEC and apply the same procedure of the previous section. The probabilistic analysis of this section focus, instead, on the average-case and not the worst-case.

A. Probabilistic Execution Time Model

A random variable is the statistical representation of the output of a phenomenon: in our case, it is the node execution time. In common with previous works [38], [39], the random variable of the execution time is a discrete quantity, and it can be expressed with a *probabilistic profile matrix*. This matrix is identified by the symbol $pET_i^{l,k}$, has size $3 \times w$, and is defined as follows:

$$pET_i^{l,k} = \begin{pmatrix} e_1 & e_2 & \cdots & e_w \\ f_i(e_1) & f_i(e_2) & \cdots & f_i(e_w) \\ F_i(e_1) & F_i(e_2) & \cdots & F_i(e_w) \end{pmatrix} \tag{10}$$

where $e_1, e_2, ..., e_w$ are execution time values, $f(\cdot)$ is the probabilistic mass function (PMF), and $F(\cdot)$ the cumulative distribution function (CDF). Even if the last two rows are redundant (PMF is computable from the CDF and vice versa), this simplifies the notation in the subsequent sections.

Example 2. *Let us consider the following execution profile:*

$$pET_1^{2,3} = \begin{pmatrix} 5 & 7 & 12 & 19 & 20 \\ 0.10 & 0.60 & 0.25 & 0.04 & 0.01 \\ 0.10 & 0.70 & 0.95 & 0.99 & 1 \end{pmatrix}$$

It represents the statistical distribution of the execution time of the node $\mathcal{N}_1^2 \in \tau_1$ running in the processor p_3. The probability that the node requires 5 units of execution time is 0.1, for 7 unit of execution time case the probability is 0.6 and so on. The last row represents the CDF. For instance, in this profile, the probability is 0.95 for execution time less or equal to 12.

1) Estimating the Distribution: The distribution of execution time is rarely known at design-time, and it is, in general, computationally expensive. A more practical approach is to measure the execution time directly on the real system and estimate the probabilistic execution time distribution. The estimation is possible by exploiting different statistical methods. If we are interested in the probabilistic-WCET (pWCET), we have to choose the Extreme Value Theory approach, while an estimation of the probabilistic-ET can be performed by estimating the empirical CDF. We assumed the WCET as given and statically computed in this work, so the pWCET is not useful in such a scenario. Instead, to improve the average-case energy consumption, the pET can be estimated by directly measuring n-samples of the execution time:

$$F_i(x) = \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}_{x_i<x}$$

where $\mathbf{1}_{x_i<x}$ is the indicator function[4] and $x_1, x_2, ..., x_n$ is the set of measured execution time samples. From the empirical-CDF $F_i(x)$ it is possible to compute the PMF by subsequent differences of the CDF: $f_i(x) = F(x) - F(x-1)$. In this way, the pET matrices can be estimated.

Uncertainty quantification. Provided that we observed all input and states of our systems, the Dvoretzky-Kiefer-Wolfowitz inequality [40] provides us the measurement error:

$$\epsilon = \sqrt{\left(\log\left(\frac{2}{c}\right)\right)/(2n)} \tag{11}$$

where c is an arbitrary low confidence. For example, if the number of samples is $n = 10\,000$, the maximum absolute error on the CDF is lower than $\epsilon \leq 0.02$ with a confidence of $c = 10^{-6}$. More sophisticated statistical techniques can obtain better results with a smaller number of samples, but this analysis would be out of scope for this paper's goals. The error in the estimated distribution would increase (or decrease) the expected energy consumption, but it would not affect the scheduling analysis. Having this value, we can determine, by adding it to the subsequent formulas, the final expected error (at a given confidence) on the overall energy consumption of the system.

[3] In this paper, the random variables are identified by the mark ˜

[4] The indicator function $\mathbf{1}_A$ has value 1 if the condition A is respected, otherwise 0.

2) Common Operations: To deal with the algebra of random variables, we report in this paragraph the operations necessary for the subsequent analysis.

Convolution. The sum of random variables is performed thanks to an operator called *convolution*. This operator is identified by the symbol $\circledast$. In particular, the convolution of two random variables $X_{sum} = X_1 \circledast X_2$ is defined as the convolution of their PMFs $f_{X_{sum}}(x) = f_{X_1}(x) \circledast f_{X_2}(x)$. The general formula of $X = Y \circledast Z$ for discrete random variables is:

$$p_X(x) = p_Y(x) \circledast p_Z(x) = \sum_{n=-\infty}^{+\infty} p_Y(n) p_Z(x-n)$$

Sum and product with a constant. The sum and product of a random variable with a constant are a shift of PMF function along the random variable axis: the expression of the PMF for Y, where $Y = X + k$, is $p_Y(x) = p_X(x-k)$; the expression of the PMF for Z, where $Z = kX$, is $p_Z(x) = p_X(x/k)$. The pET matrix can be accordingly recomputed with the newly obtained PMF.

Expected value. Also called mean, average or first moment, the *expected value* of a discrete random variable is defined as $E[X] = \sum_{i=1}^{k} x_i p_i$, where n is the number of terms composing the PMF, i.e., in our pET representation, the number of columns of the pET matrix.

B. Computing the Probabilistic Idle Time and the Average Energy

From the probabilistic execution time profile of each node, we compute the probabilistic version of Eq. (1) and Eq. (2), by exploiting the previously defined operator:

Case-1:

$$\tilde{\mathcal{I}}_i^{m,k} = b_i^{m,k} - \bigcircledast_{\forall l:\mathcal{N}_i^l \in \tau_i} (b_i^{l,k} + \tilde{d}_i^{l,k}) \cdot \mathcal{O}_{k,\mathcal{N}_i^l,\mathcal{N}_i^m} \tag{12}$$

Case-2:

$$\tilde{\mathcal{I}}_i^{m,k} = b_i^{m,k} - \bigcircledast_{\forall l:\mathcal{N}_i^l \in \tau_i} (b_i^{l,k} + \tilde{d}_i^{l,k}) \cdot \mathcal{O}_{k,\mathcal{N}_i^l,\mathcal{J}} \tag{13}$$

The start time $b_i^{l,k}$, the decision variables $\mathcal{O}_{k,\mathcal{N}_i^l,\mathcal{N}_i^m}$ and $\mathcal{O}_{k,\mathcal{N}_i^l,\mathcal{J}}$ are deterministic and non-random variables. The break-even points $T_{BE}^{j,k}$ of Eq. (3) are not affected by the probabilistic analysis because they are computed from the processor's C-States characteristics and not dependent on the task. Instead, Eq. (5) is transformed by replacing the condition with the probability values extracted from the CDF of the idle time as follows:

$$\tilde{E}_{idle}(\tilde{\mathcal{I}}_i^{m,k}) = \begin{cases} \mathcal{C}^{0,k} \cdot \tilde{\mathcal{I}}_i^{m,k} & P = F_{\tilde{\mathcal{I}}_i^{m,k}}(T_{BE}^{1,k}) \\ E_{sw}^{1,k} + \mathcal{C}^{1,k} \cdot (\tilde{\mathcal{I}}_i^{m,k} - T_{sw}^{1,k}) & P = F_{\tilde{\mathcal{I}}_i^{m,k}}(T_{BE}^{2,k}) - F_{\tilde{\mathcal{I}}_i^{m,k}}(T_{BE}^{1,k}) \\ ... & \\ E_{sw}^{r,k} + \mathcal{C}^{r,k} \cdot (\tilde{\mathcal{I}}_i^{m,k} - T_{sw}^{r,k}) & P = 1 - F_{\tilde{\mathcal{I}}_i^{m,k}}(T_{BE}^{r,k}) \end{cases} \tag{14}$$

The graphical interpretations of the probabilistic version for break-even points and the $\tilde{E}_{idle}$ are depicted in Figure 3.

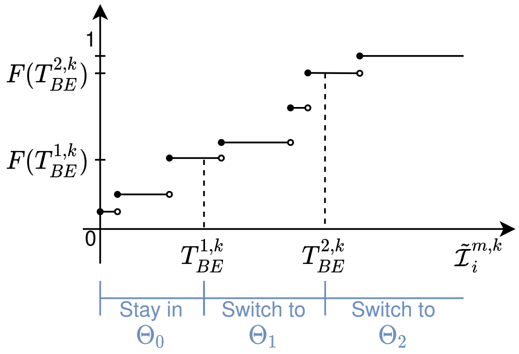

Fig. 3. Graphical interpretation, given the probabilistic information, of the break-even points. The figure shows the CDF of the idle time for the m node of the i-th task running on processor p_k.

From this equation it is possible to rebuild a probabilistic profile as defined in Section VI-A. Then, we can recompute the Eq. (8) by using a characteristic function of the random variable to compute a scalar value for $E_{idle}(\tilde{\mathcal{I}}_i^{m,k})$. We are interested in the ACEC, therefore we compute the expected value of $E_{idle}(\tilde{\mathcal{I}}_i^{m,k}) = \mathbb{E}[\tilde{E}_{idle}(\tilde{\mathcal{I}}_i^{m,k})]$ with the previous formula defined in Section VI-A2. In this work we limit our focus to the average value, but other approaches are possible, including optimizing the 95% percentile, the median value, the pWCET, or the pWCEC [8]. Once the scalar value for $E_{idle}(\tilde{\mathcal{I}}_i^{m,k})$ is computed, the energy analysis proceed as explained in the previous section – in particular, via Algorithm 1 and Algorithm 2 – but, instead of optimizing the WCEC, it minimizes the ACEC.

C. A Complete Example

To show a complete example with the probabilistic information, we consider the DAG task of Figure 1, and, in particular, we focus on the decision during the idle time between the node N_i^2 and the node N_i^3 execution on p_2 with speed $S_{i,2}^3 = 1$. Let us assume that the node N_i^2 has a begin time $b_i^{2,2} = 0$ and is characterized by the following probabilistic-ET:

$$pET_i^{2,2} = \begin{pmatrix} 1 & 2 & 3 \\ 0.20 & 0.75 & 0.05 \\ 0.20 & 0.95 & 1 \end{pmatrix}$$

By using the WCET to compute $d_i^{2,2}$, the idle time $\mathcal{I}_i^{3,2}$ is zero, as calculated by Eq. (5) because the next node N_i^3 has $b_i^{3,2} = 3$ and $S_{i,2}^3 = 1$. Instead, by replacing the duration with its probabilistic version $\tilde{d}_i^{2,2}$, we obtain the following probabilistic profile for $\tilde{\mathcal{I}}_i^{3,2}$:

$$p\tilde{\mathcal{I}}_i^{3,2} = \begin{pmatrix} 0 & 1 & 2 \\ 0.05 & 0.75 & 0.20 \\ 0.05 & 0.80 & 1 \end{pmatrix}$$

Let us assume that the computing platform has the following parameters:

- Three C-states: $\Theta_0, \Theta_1, \Theta_2$
- Power consumption of p_2 for each C-state: $\mathcal{C}^{0,2} = 15, \mathcal{C}^{1,2} = 5, \mathcal{C}^{2,2} = 1$

- Energy overheads: $E_{sw}^{1,2} = 7$, $E_{sw}^{2,2} = 12$
- Time overheads: $T_{sw}^{1,2} = 0.2$, $T_{sw}^{2,2} = 0.5$

So, we can compute the break-even times according to Eq. (3) (please note that the computation of break-even time is not dependent on probabilistic information): $T_{BE}^{1,2} = \frac{3}{5}$, $T_{BE}^{2,2} = \frac{11}{8}$. We then build the energy idle as follows:

$$\tilde{E}_{idle}(\tilde{\mathcal{I}}_i^{3,2}) = \begin{cases} 15 \cdot \tilde{\mathcal{I}}_i^{3,2} & P = F_{\tilde{\mathcal{I}}_i^{3,2}}(3/5) = 0.05 \\ 7 + 5 \cdot (\tilde{\mathcal{I}}_i^{3,2} - 0.2) & P = F_{\tilde{\mathcal{I}}_i^{3,2}}(11/8) - F_{\tilde{\mathcal{I}}_i^{3,2}}(3/5) = 0.75 \\ 12 + 1 \cdot (\tilde{\mathcal{I}}_i^{3,2} - 0.5) & P = 1 - F_{\tilde{\mathcal{I}}_i^{3,2}}(11/8) = 0.2 \end{cases}$$

Finally, we compute the scalar expected value of $E_{idle}(\tilde{\mathcal{I}}_i^{3,2}) = \mathbb{E}[\tilde{E}_{idle}(\tilde{\mathcal{I}}_i^{3,2})]$:

$$\begin{aligned} E_{idle}(\tilde{\mathcal{I}}_i^{m,k}) = {} & [15 \cdot 0] \cdot 0.05 + [7 + 5(1 - 0.2)] \cdot 0.75 \\ & + [12 + 1(2 - 0.5)] \cdot 0.2 = 10.95 \end{aligned}$$

This value represents the average energy consumption of the idle time when the break-even strategy is applied, and it can then be used to optimize the ACEC. For the sake of the example, we also computed the idle energy in case a fixed selection C-States policy is applied:

- the processor remains in Θ_0, obtaining average $E_{idle}(\tilde{\mathcal{I}}_i^{3,2}) = 17.25$
- the processor always switches (and remains) to Θ_1 , obtaining average $E_{idle}(\tilde{\mathcal{I}}_i^{3,2}) = 12.75$
- the processor always switches (and remains) to Θ_2 , obtaining average $E_{idle}(\tilde{\mathcal{I}}_i^{3,2}) = 13.15$

In this example, it is possible to notice how the optimized strategy is better than any fixed selection of the C-States. The next section performs an extensive simulation to show how this strategy is beneficial in different scenarios.

VII. Simulation Results

We performed a large-scale simulation and an experimental study on a real hardware platform to evaluate the proposed policies and their energy efficiency. The latter is presented in Section VIII, while, in this section, we focus on the simulation. Subsection VII-A compares our approach with a previous work by focusing on the WCEC optimization, while Section VII-B improves the ACEC by exploiting the probabilistic policy. We compared our policies with the recent work by Guo et al. [3] because it has a similar DAG task model and problem assumptions, while other works would make a correct and fair comparison difficult. We denoted this baseline in this article with *Fed_Guo*.

A. Energy Results of the WCEC Approach

To verify the performance of our WCEC policy, we conducted the simulation on randomly generated task sets by varying the following input parameters:

- $2 \leq M_i \leq 4$: Degree of parallelism of DAG tasks (with the number of nodes in the interval $[4; 16]$
- p: Set of processors
- $S \in [1; 1.5]$: Execution speeds

A total number of 10 000 random generated task sets have been explored, comparing the energy consumption of both our approach and the baseline approach *Fed_Guo*. The code and the full dataset is available online to enhance the reproducibility[5]. We depicted the results of the first 20 runs in Figure 4, while the aggregate data of the whole experiments are reported in Table IV. Our approach exhibits an average of 22% energy savings compared to *Fed_Guo*, with a peak maximum of 32.1% and a minimum 9.6%. Our approach outperforms the baseline in all the generated task sets. The application of the C-States method is more effective than the static policy applied to idle slots like in *Fed_Guo*, considerably reducing the energy consumption when the policy is optimized for the WCEC case.

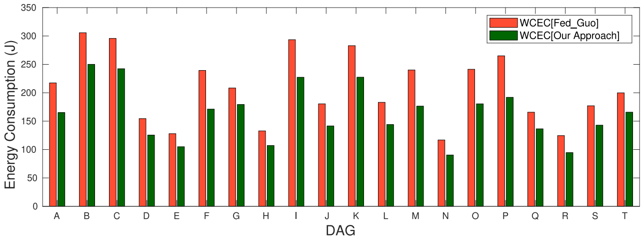

Fig. 4. Energy consumption comparison between our approach and *Fed_Guo*. The figure shows only the first 20 scenarios of the overall 10 000 randomly generated DAGs.

TABLE IV
Aggregate results of the improvement of our approach compared to the baseline *Fed_Guo* on 10 000 generated task sets.

Energy Saving	**Absolute value**	**Percentage**
Average	46 J	22.2%
Variance	264 J^2	-
Maximum	94 J	32.1%
Minimum	12 J	9.6%

B. Energy Results of the ACEC Approach

To check the improvement of the ACEC optimization, we considered the DAG A of Figure 4 and generated random CDFs for the task execution time. We performed the energy optimization, this time according to the probabilistic analysis of Section VI. Then, we run the task set 10 000 times by sampling the task execution time from the random CDFs previously generated with the inverse transform sampling method. The first 50 runs are illustrated in Figure 5. The theoretical WCEC (estimated by the analysis of the previous subsection) is 165. On the overall 10 000 samples data, the two optimizations performed as follows:

- WCEC-policy (Section V): the minimum and maximum observed energy consumption are respectively 28.8 J and 94.3 J, while the average value is 53.6 J.
- ACEC-policy (Section VI): the minimum and maximum observed energy consumption are respectively 27.8 J and 89.9 J, while the average value is 46.7 J.

Consequently, the ACEC-policy reduces the average energy consumption on average of about 12.8%, compared to our

[5] https://doi.org/10.5281/zenodo.5528052

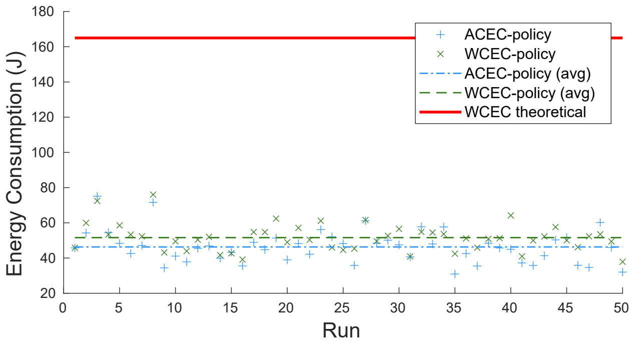

Fig. 5. The first 50 runs of the ACEC experiment. The figure shows the pessimism of the theoretical WCEC and how the allocation of the ACEC-optimized policy presents a lower average-case energy consumption compared to the WCEC-optimized policy.

TABLE V
CHARACTERIZATION OF THE ODROID-H2 BOARD. THE VALUES OF T_{sw} ARE PROVIDED BY THE MANUFACTURER, WHILE WE MEASURED THE OTHER PARAMETERS. EACH REPORTED VALUE IS THE AVERAGE OF 1 000 SAMPLES, AND THE OEFFICIENT OF VARIATION WAS LESS THAN 1 FOR ALL THE VALUES.

Θ_i	$T_{sw} [\mu s]$	$E_{sw} [mJ]$	$C_{l,1} [mW]$	$C_{l,2} [mW]$	$C_{l,3} [mW]$
C0	0	0	656.3	507.7	310.0
C1E	10	0.23	41.3		
C6	150	0.32	29.6		
C8	5963	1.31	27.4		

WCEC-policy, and 28.6%, compared to the *Fed_Guo* WCEC-policy.

VIII. EXPERIMENTAL RESULTS

This section reports the experimental evaluation to estimate the improvement in terms of both ACEC and WCEC compared to the *Fed_Guo* baseline (described in Section VII), when our analyses are employed on a real platform.

A. Experimental Setup

The chosen platform is the Odroid-H2 board equipped with a quad-core J4105 processor and implementing the Intel x86-64 C-States. In particular, this board is capable of applying DVFS for each core. We used this feature not for energy consumption but to simulate the processor heterogeneity setting a different fixed frequency to each core. The board has the 4 C-States (including the running C-State Θ_0) reported in Table V. The board runs the Linux kernel, which has been patched with PREEMPT_RT and configured for the real-time workload. Even if Linux is not a hard real-time kernel in the safety-critical sense, the PREEMPT_RT patch makes it a good test bench for scientific purposes [41]. To improve the reproducibility and reduce the interference, we dedicated the *core 0* to run the Linux kernel, service applications, interrupts, and the measurement scripts, while *core 1*, *core 2*, *core 3* are the actual processors considered in the analysis and configured to run real-time workload by following the relative guidelines [41]. The *core 1* was configured at full speed $S_1 = 1$, *core 2* at speed $S_2 = 1.25$, and *core 3* at speed $S_3 = 1.5$. We connected the board to an industrial-graded power measurement device with a resolution of $2\,\mu W$ and a sampling frequency of $320\,S/s$ controlled by a third machine not affecting the timing property of the system under analysis. To characterize each core, we computed the difference between the baseline (*core 0*, with all the other core offline) and each configuration (e.g., *core 0* and *core 2* online, with all the other core offline). The results of the relevant metrics are reported in Table V.

B. Results

We compare our approach to the same baseline used in Section VII. We built a DAG and its decomposed structure similar to the example of Figure 1. The workload of each node has been selected by taking 5 benchmarks from the well-known Mälardalen WCET suite [42]: $N_i^1 = \texttt{cnt}$, $N_i^2 = \texttt{insert}$, $N_i^3 = \texttt{ns}$, $N_i^4 = N_i^5 = \texttt{prime}$, and $N_i^6 = \texttt{sqrt}$. Each benchmark was modified to allow us to inject a random (but predictable, for reproducibility) input and to increase the total execution time. The original Mälardalen WCET benchmarks are too small and fast to efficaciously measure the power/energy consumption. Therefore, depending on the benchmark, we implemented each execution so that it is composed of repetitive executions for a variable number of times. The benchmark choices have been driven by their WCETs in order to match the decomposition of Figure 1. Due to the complexity of the test-bench hardware and software, a measurement-based approach was used to obtain the WCET estimation. Since a WCET underestimation would have invalidated the results, a proper run-time mechanism was put in place to detect any violation (which never occurred in any the experiment case) of the WCET thresholds of the nodes.

We run the experiments and compared the baseline *Fed_Guo* with our policy. Both WCEC-optimization and ACEC-optimization policies provided the same processor allocation. Hence the optimal WCEC solution is also the optimal ACEC solution in this case. We gathered 1 000 samples data resulting in the following aggregate data:

- *Fed_Guo*: the minimum and maximum observed energy consumption are respectively $1.78\,J$ and $4.80\,J$, while the average value is $2.00\,J$.
- Our policy: the minimum and maximum observed energy consumption are respectively $1.53\,J$ and $4.50\,J$, while the average value is $1.75\,J$.

Our policy improved the (observed) WCEC of 12.32% and of the ACEC of 6.29%. These results are slightly worse than the theoretical results of the previous section. This can be explained by the fact that our approach relies on a perfect characterization of the hardware and the benchmarks (like most of all hard real-time approaches), which is difficult in the considered complex computing platform. The operating system and the software layers are another source of overhead not considered in the model but present in the experimental data. Nevertheless, even with such an imperfect model, our approach still outperforms the baseline, saving a significant amount of energy.

IX. Conclusion

We proposed an energy-aware task allocation strategy that can optimize the WCEC in the context of hard real-time tasks described with a DAG and running on heterogeneous processors. The offline analysis allows us to verify worst-case requirements in an energy-constrained system, while the online actuation ensures the best possible choice depending on the actual values of tasks' execution time. Then, this allocation policy is extended thanks to probabilistic execution time information to optimize the average-case – i.e., minimize the ACEC. Both policies guarantee the schedulability thanks to the statically computed node WCETs, taking into account also the DPM overheads. To verify the effectiveness of our approach, we run an extensive simulation, which resulted in an energy saving of up to 32.1% of the WCEC compared to the baseline (22% on average), with a further improvement of 12.8% when the ACEC-policy is employed. Then, we run a small-scale experimental test on a real board to measure the real energy consumption, resulting in an improvement of 12.32% the WCEC and 6.29% the ACEC.

Future Research. Although representing parallelizable code using a DAG model is very popular in the real-time community, such a model suffers from several shortcomings [43], e.g., the complex internal structure of the code. Recently, an alternative model is proposed that characterizes a DAG by just two parameters [43] – *work* and *span*. This model does not require details knowledge regarding each node and its dependencies. In the future, we plan to extend our analysis for the model mentioned above. In addition, future works dealing with scheduling and resource management may consider other components, e.g., resource contention (cache misses, shared data, etc.), software overheads (context switches, scheduler, etc.), and input/output devices that affect the overall power and energy consumption.

Acknowledgments

This work is supported by NSF grants CNS 1850851, PPoSS 2028481, and the European Union's Horizon 2020 Research and Innovation programme (No. 801137) [44].

References

[1] V. Bonifaci, A. Marchetti-Spaccamela, S. Stiller, and A. Wiese, "Feasibility analysis in the sporadic DAG task model," in *ECRTS*, 2013.

[2] J. Li, J. J. Chen, K. Agrawal, C. Lu, C. Gill, and A. Saifullah, "Analysis of federated and global scheduling for parallel real-time tasks," in *ECRTS*. IEEE, 2014.

[3] Z. Guo, A. Bhuiyan, A. Saifullah, N. Guan, and H. Xiong, "Energy-efficient multi-core scheduling for real-time dag tasks," in *LIPIcs-Leibniz International Proceedings in Informatics*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[4] J. Goossens and P. Richard, "Optimal scheduling of periodic gang tasks," *Leibniz transactions on embedded systems*, vol. 3, no. 1, pp. 04–1, 2016.

[5] Z. Dong and C. Liu, "Analysis techniques for supporting hard real-time sporadic gang task systems," in *Real-Time Systems Symposium (RTSS), 2017 IEEE*. IEEE, 2017, pp. 128–138.

[6] A. ahmed Bhuiyan, K. Yang, S. Arefin, A. Saifullah, N. Guan, and Z. Guo, "Mixed-criticality multicore scheduling of real-time gang task systems," in *2019 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2019, pp. 469–480.

[7] B. Andersson and D. de Niz, "Analyzing global-edf for multiprocessor scheduling of parallel tasks," in *International Conference On Principles Of Distributed Systems*. Springer, 2012, pp. 16–30.

[8] F. Reghenzani, G. Massari, and W. Fornaciari, "A probabilistic approach to energy-constrained mixed-criticality systems," in *2019 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, 2019, pp. 1–6.

[9] N. K. Jha, "Low power system scheduling and synthesis," in *IEEE/ACM International Conference on Computer Aided Design. ICCAD 2001. IEEE/ACM Digest of Technical Papers (Cat. No. 01CH37281)*. IEEE, 2001, pp. 259–263.

[10] K. Huang, L. Santinelli, J.-J. Chen, L. Thiele, and G. C. Buttazzo, "Applying real-time interface and calculus for dynamic power management in hard real-time systems," *Real-Time Systems*, vol. 47, no. 2, pp. 163–193, 2011.

[11] H. Cheng and S. Goddard, "Online energy-aware I/O device scheduling for hard real-time systems," in *Proceedings of the conference on Design, automation and test in Europe:*. European Design and Automation Association, 2006.

[12] M. Bambagini, M. Marinoni, H. Aydin, and G. Buttazzo, "Energy-aware scheduling for real-time systems: A survey," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 15, no. 1, p. 7, 2016.

[13] A. Saifullah, D. Ferry, J. Li, K. Agrawal, C. Lu, and C. D. Gill, "Parallel real-time scheduling of dags," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 12, pp. 3242–3252, 2014.

[14] K. Agrawal, S. Baruah, P. Ekberg, and J. Li, "Optimal scheduling of measurement-based parallel real-time tasks," *Real-Time Systems*, pp. 1–7, 2020.

[15] D. Ferry, J. Li, M. Mahadevan, K. Agrawal, C. Gill, and C. Lu, "A real-time scheduling service for parallel tasks," in *2013 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2013, pp. 261–272.

[16] A. Saifullah, J. Li, K. Agrawal, C. Lu, and C. Gill, "Multi-core real-time scheduling for generalized parallel task models," *Real-Time Systems*, 2013.

[17] S. Baruah, V. Bonifaci, and A. Marchetti-Spaccamela, "The global EDF scheduling of systems of conditional sporadic DAG tasks," in *ECRTS*, 2015.

[18] S. Baruah, V. Bonifaci, A. Marchetti-Spaccamela, L. Stougie, and A. Wiese, "A generalized parallel task model for recurrent real-time processes," in *RTSS*. IEEE, 2012.

[19] S. Baruah, "The federated scheduling of systems of mixed-criticality sporadic DAG tasks," in *2016 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2016, pp. 227–236.

[20] D. Zhu, N. AbouGhazaleh, D. Mossé, and R. Melhem, "Power aware scheduling for and/or graphs in multiprocessor real-time systems," in *ICPP*. IEEE, 2002.

[21] D. Zhu, D. Mosse, and R. Melhem, "Power-aware scheduling for and/or graphs in real-time systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 15, no. 9, pp. 849–864, 2004.

[22] A. Bhuiyan, Z. Guo, A. Saifullah, N. Guan, and H. Xiong, "Energy-efficient real-time scheduling of dag tasks," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 17, no. 5, p. 84, 2018.

[23] A. Bhuiyan, D. Liu, A. Khan, A. Saifullah, N. Guan, and Z. Guo, "Energy-efficient parallel real-time scheduling on clustered multi-core," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 9, pp. 2097–2111, 2020.

[24] Z. Guo, A. Bhuiyan, D. Liu, A. Khan, A. Saifullah, and N. Guan, "Energy-efficient real-time scheduling of DAGs on clustered multi-core platforms," in *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2019, pp. 156–168.

[25] L. Premi, F. Reghenzani, G. Massari, and W. Fornaciari, "A game theory approach to heterogeneous resource management: Work-in-progress," in *2020 International Conference on Embedded Software (EMSOFT)*, 2020, pp. 25–27.

[26] A. Saifullah, S. Fahmida, V. P. Modekurthy, N. Fisher, and Z. Guo, "CPU energy-aware parallel real-time scheduling," in *32nd Euromicro Conference on Real-Time Systems (ECRTS 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

[27] A. Paolillo, J. Goossens, P. M. Hettiarachchi, and N. Fisher, "Power minimization for parallel real-time systems with malleable jobs and homogeneous frequencies," in *RTCSA*. IEEE, 2014.

[28] M. E. Gerards and J. Kuper, "Optimal dpm and dvfs for frame-based real-time systems," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 9, no. 4, pp. 1–23, 2013.

[29] A. Esmaili, M. Nazemi, and M. Pedram, "Modeling processor idle times in mpsoc platforms to enable integrated DPM, DVFS, and task scheduling subject to a hard deadline," in *Proceedings of the 24th Asia and South Pacific Design Automation Conference*, 2019, pp. 532–537.

[30] K. Huang, K. Wang, D. Zheng, X. Jiang, X. Zhang, R. Yan, and X. Yan, "Expected energy optimization for real-time multiprocessor socs running periodic tasks with uncertain execution time," *IEEE Transactions on Sustainable Computing*, pp. 1–1, 2018.

[31] I. Corporation, "Energy-efficient platforms – considerations for application software and services," Intel Corporation, White paper 325085-001, 2011. [Online]. Available: https://www.intel.ru/content/dam/doc/white-paper/energy-efficient-platforms-2011-white-paper.pdf

[32] A. Dunkels, F. Osterlind, N. Tsiftes, and Z. He, "Software-based on-line energy estimation for sensor nodes," in *Proceedings of the 4th workshop on Embedded networked sensors*. ACM, 2007, pp. 28–32.

[33] A. Sinha and A. Chandrakasan, "Dynamic power management in wireless sensor networks," *IEEE Design & Test of Computers*, vol. 18, no. 2, pp. 62–74, 2001.

[34] G. Chen, K. Huang, and A. Knoll, "Energy optimization for real-time multiprocessor system-on-chip with optimal DVFS and DPM combination," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 13, no. 3s, p. 111, 2014.

[35] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval research logistics quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.

[36] Http://www.hungarianalgorithm.com/hungarianalgorithm.php.

[37] R. Davis and L. Cucu-Grosjean, "A survey of probabilistic timing analysis techniques for real-time systems," *Leibniz Transactions on Embedded Systems*, vol. 6, no. 1, pp. 03–1, 2019.

[38] D. Maxim and L. Cucu-Grosjean, "Response time analysis for fixed-priority tasks with multiple probabilistic parameters," in *2013 IEEE 34th Real-Time Systems Symposium*, 2013, pp. 224–235.

[39] A. Bhuiyan, F. Reghenzani, W. Fornaciari, and Z. Guo, "Optimizing energy in non-preemptive mixed-criticality scheduling by exploiting probabilistic information," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 11, pp. 3906–3917, 2020.

[40] A. Dvoretzky, J. Kiefer, and J. Wolfowitz, "Asymptotic minimax character of the sample distribution function and of the classical multinomial estimator," *Ann. Math. Statist.*, vol. 27, no. 3, pp. 642–669, 09 1956.

[41] F. Reghenzani, G. Massari, and W. Fornaciari, "The real-time linux kernel: A survey on PREEMPT_RT," *ACM Comput. Surv.*, vol. 52, no. 1, Feb. 2019.

[42] J. Gustafsson, A. Betts, A. Ermedahl, and B. Lisper, "The Mälardalen WCET Benchmarks: Past, Present And Future," in *10th International Workshop on Worst-Case Execution Time Analysis (WCET 2010)*, ser. OpenAccess Series in Informatics (OASIcs), B. Lisper, Ed., vol. 15. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2010, pp. 136–146, the printed version of the WCET'10 proceedings are published by OCG (www.ocg.at) - ISBN 978-3-85403-268-7.

[43] K. Agrawal and S. Baruah, "A measurement-based model for parallel real-time tasks," in *30th Euromicro Conference on Real-Time Systems*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[44] W. Fornaciari, G. Agosta, D. Atienza, C. Brandolese, L. Cammoun, L. Cremona, A. Cilardo, A. Farres, J. Flich, C. Hernandez, M. Kulchewski, S. Libutti, J. M. Martínez, G. Massari, A. Oleksiak, A. Pupykina, F. Reghenzani, R. Tornero, M. Zanella, M. Zapater, and D. Zoni, "Reliable power and time-constraints-aware predictive management of heterogeneous exascale systems," in *Proceedings of the 18th International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation*, ser. SAMOS '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 187–194.