Energy-Harvesting-Aware Federated Scheduling of Parallel Real-Time Tasks

Jamal Mohammadi[1], Mahmoud Shirazi[2†], Mehdi Kargahi[1,3*†]

[1*]School of Electrical and Computer Engineering, College of Engineering, University of Tehran, Tehran, Iran.
[2] Department of Computer Science and Information Technology, Institute for Advanced Studies in Basic Sciences (IASBS), Zanjan, Iran.
[3]School of Computer Science, Institute for Research in Fundamental Sciences (IPM), Tehran, Iran.

*Corresponding author(s). E-mail(s): kargahi@ut.ac.ir;
Contributing authors: jmohammadi@ut.ac.ir; m.shirazi@iasbs.ac.ir;
†These authors contributed equally to this work.

Abstract

This paper presents HEARTS, a multicore energy scheduling approach utilizing a federated strategy designed for parallel real-time tasks of significant computational demands in embedded systems deployed in environments of unreliable power sources; like surveillance and intelligent city infrastructures. HEARTS, specialized for high-utilization parallel tasks, divides the scheduling horizon into multiple windows, so that it dynamically allocates the cores to the tasks based on the energy availability. It introduces two schedulers based on the first-fit and last-fit approaches. We demonstrate the optimality of the last-fit-based scheduler when the battery capacity exceeds some specific threshold; further, we show scenarios where the first-fit-based scheduler performs better under lower capacities. Simulations using two setups—one with random harvested energy and task parameters, and the other with real solar energy and benchmark tasks—show a maximum deviation of 19.05% and 21.34% from two theoretical optimal solutions, respectively, and a substantial improvement of 28.24% over the energy partitioning approach.

Keywords: Real-time Scheduling, Energy-Aware Scheduling, Parallel Tasks, Energy Harvesting

1 Introduction

Real-time embedded systems are vital components of Cyber-Physical Systems (CPSs), providing essential real-time and reliable services [1][2]. These systems interact with various sensor data, which must be processed within strict time constraints. As CPSs continue to evolve, the computational demands of tasks such as computer vision, pattern recognition, and motion planning grow, especially with the use of more advanced algorithms and the need to handle finer details, like higher-resolution images in machine vision [3][4]. Over the past decade, there has been a paradigm shift in processor performance, driven by the increasing number of processor cores. The growing demand for parallel processing has led to the rapid development of multi-core embedded processors, which have now evolved into many-core processors [5]. Examples include Intel's 48-core SCC chip [6], TILERA's 100-core TILE-Gx100 [7], and picoChip's 248-core PC205 [8]. Consequently, the landscape has seen a surge in research directed towards parallel languages and runtime environments like Cilk [9], Intel Cilk Plus [10], and IBM X10 [11] designed to harness the processing power of multi-core architectures. These programming languages embody parallel algorithms through structures like `spawn`, `sync`, `fork`, `join`, and `parallel-for`.

The escalating demand for parallelism in multi-core systems intensifies the challenge of energy consumption, particularly for parallel tasks on multiprocessor systems. As the number of cores increases, concurrent tasks demand more energy, exacerbating the strain on power resources. Real-time applications, such as autonomous vehicles and robotics, exemplify this challenge. In these domains, tasks like simultaneous sensor data processing, environment perception, and decision-making mandate parallel execution on multi-core systems. Energy supply becomes a critical factor in these scenarios, as many of these applications operate in environments where energy resources are limited.

Energy consumption is a critical concern in CPSs, as they must function efficiently under strict energy constraints. These systems typically rely on energy storage solutions such as batteries or super-capacitors. However, replacing or upgrading these storage options can be costly or impractical [12]. Alternative environmental energy sources like solar, wind, or thermal energy offer promising possibilities [13], but their intermittent nature due to changing environmental conditions makes it challenging to ensure a steady energy supply [14][15]. Effectively managing energy in these dynamic and resource-limited environments requires advanced strategies, especially when handling parallel tasks on multi-core embedded systems in real-time applications.

The shift in real-time systems, transitioning from single-core to multi-core processors, has given rise to various scheduling algorithms tailored for parallel real-time tasks [16][17][18][19][20]. This study employs the federated scheduling approach [21][22][23][24], a generalized strategy derived from partitioned scheduling algorithms, designed to scheduling Directed Acyclic Graph (DAG) task model. Federated scheduling is renowned for its exceptional real-time performance [25], and efficient management of DAG tasks by assigning dedicated processors to each heavy task. Prior research underscores the advantages of federated scheduling, particularly for high-utilization tasks, including improved analytical schedulability and reduced scheduling overhead [26].

In this paper, we study a multiprocessor system equipped with an energy harvester. Practical instances of such systems include autonomous devices for environmental monitoring [27], surveillance systems used particularly in border or wildlife protection [28], space exploration rovers and probes [29], underwater systems for marine life studies [30][31], and smart city infrastructure facilitating intelligent lighting and traffic monitoring [32][33].

These systems rely on complex algorithms for critical functions such as analyzing environmental imagery or signals for decision-making. This involves real-time processing of large datasets, leading to significant computational demands. Consequently, these systems may need to analyze gigabytes of data daily in real-time [34]. The volume of data could overwhelm any network, making it impractical to send back to a data center for processing. Instead, it must be processed on-site using a compact, parallel supercomputer. Furthermore, energy autonomy is a crucial requirement for these systems, enabling them to operate solely on power harvested from environmental sources such as sunlight, wind, and kinetic energy. This capability is vital for functionality in isolated areas or situations where conventional power sources are unavailable.

The objective of this paper is to schedule tasks in such a system based on the energy's input limit and intermittency, as well as the number of available cores. The primary challenge is determining the number of cores assigned to each task and determining which tasks should be executed in each unit of time, considering the input energy, to ensure all tasks meet their deadlines. We introduce a runtime scheduling method, namely HEARTS (Harvesting-Energy Aware Runtime Task Scheduler), which decides on the number of assigned cores to each task in different time intervals and on scheduling details of each interval, using two distinct energy scheduling methods.

HEARTS is a dynamic runtime scheduler we designed for multi-core processors powered by harvested, intermittent energy sources like solar or wind. It addresses the critical challenge of executing parallel, real-time tasks—each modeled as a DAG—within strict energy constraints imposed by fluctuating energy inputs. By intelligently assigning cores to tasks and dynamically scheduling them based on current energy availability and task deadlines, HEARTS employs two innovative energy scheduling methods based on first-fit (PASAP) and last-fit (PALAP) approaches. These methods adaptively manage the allocation of harvested energy to tasks, ensuring all tasks are completed within their deadlines while minimizing the number of active cores. This approach is particularly useful for autonomous systems in remote or energy-scarce environments—such as environmental monitoring devices, surveillance systems, space exploration rovers, and smart city infrastructure—where maintaining operation solely on harvested energy is essential.

This paper investigates the scheduling of parallel tasks within multi-core systems powered by harvested energy, ensuring the fulfillment of task deadlines. Previous studies [35][36][37][38] have explored either energy-aware task scheduling on single-core systems or energy-efficient task scheduling on multi-core systems. Remarkably, no prior research has addressed the specific challenge of scheduling parallel tasks aware of harvested energy in multi-core environments.

This paper's contributions are summarized as below:

- Analyzing the energy consumption of parallel tasks, considering the number of dedicated cores assigned to each task and the task properties.
- Introducing two innovative energy scheduler based on the first-fit and last-fit approaches accommodating variable input energy rates.
- Proposing a dynamic runtime scheduler that leverages two energy schedulers, as introduced in this paper.
- Examine the performance of energy schedulers by proving the optimality of the last-fit approach above a certain battery capacity threshold and highlighting scenarios where the first-fit approach is superior below this threshold.

The remainder of this paper is structured as follows: Section 2 presents the system model and preliminary concepts. The system properties, the associated problem, and its computational complexity are analyzed in Section 3. Next, we describe our proposed method in Section 4, where we discuss how the runtime scheduler calls the energy scheduler during the mission horizon to determine the number of cores required for each task to meet their deadlines. Section 5 explores the power consumption of parallel tasks, while Section 6 introduces two energy scheduling algorithms to assess task schedulability and provide detailed information on task energy scheduling at each step. Section 7 provides the runtime scheduler's detail. In Section 8, we analyze the time complexity of the proposed method. In Section 9, we discuss the optimality of the proposed method. In Section 10, we evaluate the proposed algorithms by simulations. Section 11 provides literature review of real-time harvesting energy-aware and energy-efficient scheduling. Finally, Section 12 concludes the paper.

2 System model

This section defines the system model considered in this paper. The desired system in this paper is a multiprocessor system equipped with an energy harvester. All tasks in this system are independent and multi-threaded and can be executed across multiple cores. The system's mission is to execute the multi-threaded tasks within the interval from zero to *horizon*, namely the duration of the system's mission.

In our system, time is discretized into quanta referred to as *steps*. Each parameter associated with a task is quantified in terms of steps, providing a standardized unit of time for all task-related parameters. Our system considers a computing platform consisting of N identical computing cores of a similar and constant frequency. Details regarding the minimum required core count for each task are discussed in Section 3.2. All task parameters have been assessed using a single core.

In the remainder of this section, we introduce the parallel task model and energy models. Table 1 summarizes the notations used in this paper.

2.1 Parallel task model

Each parallel task τ_i can be represented as a directed acyclic graph (DAG), where vertices denote computational threads and edges indicate precedence relationships. In our work, we focus on two key parameters derived from the DAG representation: *the total execution time* (C_i) and *the critical path length* (L_i). The total execution time C_i

Table 1: Notation and nomenclature used in this paper.

Notation	Definition
τ	System task set.
τ_i	Task number i.
C_i	The total execution time of task τ_i
L_i	The critical path length of task τ_i.
π_i	Period of task τ_i.
D_i	Implicit relative deadline of task τ_i.
p_i	Power consumption of task τ_i on a single core.
u_i	Utilization of task τ_i.
$W_{i,m}^{max}$	Worst-case execution-time of task τ_i on m cores.
HP	The hyper-period of task set τ.
n_i^{min}	The minimum number of cores needed to meet the deadline of task τ_i.
$minCores$	The sum of the minimum number of cores required for all tasks, irrespective of the harvested energy rate.
P_{static}	Static power consumption of each core.
N	The number of cores in the system.
$P_R(s)$	Energy charge rate at Step s.
B	Nominal capacity of battery.
IE	The initial energy in the battery.
res_s	The portion of the harvested energy in Step s that is reserved in the battery for use in subsequent steps.
$used_s$	Total used energy in Step s.
$ure(s)$	Unallocated replenished energy at Step s.
$En(s)$	The available energy in Step s.
ca	Core assignment matrix.
ca_i	Row i of core assignment matrix.
win_{size}	Default scheduling window size.
α	Beginning of current scheduling window.
β	End of current scheduling window.
χ	The earliest deadline among the last jobs of all tasks within the current scheduling window.

is the sum of the execution times of all vertices in the DAG, which is also referred to as the *worst-case execution time (WCET)* of task τ_i on a single processor. The critical path length L_i represents the longest execution path in the DAG. These parameters are essential for parallel task scheduling. Further details on DAG-based task models and their measurements can be found in [39–43].

We consider a task set of $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$, consisting of n parallel periodic tasks with implicit deadline, where each task τ_i is defined by:

$$\tau_i = (C_i, L_i, \pi_i, \phi_i, p_i) \tag{1}$$

where

- C_i represents the total execution time of task τ_i.
- L_i indicates the critical path length of task τ_i.
- π_i represents the period of task τ_i with implicit relative deadline of D_i, i.e., $\pi_i = D_i$.
- Φ_i represents the release time of the first job of task τ_i.
- p_i gives the power consumption of task τ_i when executing on a single core.

If phase of a task is not specified, the release time of the first job is assumed to be zero. Each task has a fixed priority, and the ones with a smaller index are of higher priority. More formally, given two tasks τ_i and τ_j, τ_i has a higher priority if $i < j$.

Each task τ_i is assigned a particular number of dedicated cores, which remain exclusive to it, even during idle intervals. The method of core allocation is elaborated in Section 3.1. Moreover, for each task τ_i, a minimum number of cores is essential to meet its deadline; additionally, it's feasible to calculate the task-specific useful core count, indicating that allocating more cores to task τ_i becomes ineffective, leaving the execution time unchanged. In Section 3.2, we provide formulas for determining minimum and maximum core counts. Any number of cores within the minimum and maximum range can be used to meet the task deadline. A higher core count within the range provides more slack while achieving the specified deadline.

The *hyper-period* is defined as the least common multiple (lcm) of the periods of all tasks, i.e., $HP = lcm(\pi_1, ..., \pi_n)$.

The *utilization* of task τ_i, denoted by u_i, is defined as the ratio of its total execution time (C_i) to its period (π_i):

$$u_i = \frac{C_i}{\pi_i} \tag{2}$$

Utilization is a measure that represents the fraction of a single core's capacity required by task τ_i over its period. Specifically, it quantifies the workload intensity of the task relative to the available processing time on a single core. A utilization of $u_i = 1$ implies that task τ_i would fully occupy a single core during its entire period, leaving no idle time or capacity for other tasks. When $u_i > 1$, it indicates that the task demands more processing time than what is available on a single core within its period. Consequently, such a task cannot meet its deadline if executed on a single core and requires allocation to multiple cores. We assume that the tasks have utilizations greater than one, i.e., $\frac{C_i}{\pi_i} > 1$, for $1 \leq i \leq n$. Therefore, allocating one core would not be sufficient to meet the deadline.

Because of the uncertainty in inputs of the parallel program as well as specification of the exact execution paths of DAG in run time (due to the presence of conditions and loops in the program), given the number of cores allocated to a task, we may see different execution times per job for each task [44]. In other words, with the constant parameters of the C_i, L_i, and the number of assigned cores, the execution time of each job of task τ_i can be different. Nevertheless, some boundaries for best/worst execution time can be specified, which we will address in Section 3.2.

2.2 Energy models

Here, we introduce the energy models, including the processor energy model and the energy supplier model.

2.2.1 Processor energy model

Here we use a widely used model of processor energy consumption ([45][46][47][48][49][50]). The model divides power consumption into two parts: **static**

power and **dynamic power**. Each core consumes power at a constant rate of P_{static} while it is active. In addition, energy consumption is affected by the execution of tasks on cores. While running on **only one** core, each parallel task τ_i consumes energy at a rate of p_i in the worst-case scenario. We also introduce a function to calculate the energy consumption rate of each task on m cores, which will be described in Section 5.

In our analysis, we simplify the consideration of energy overhead associated with interconnecting the cores and the utilization of shared resources. More precisely, We assume these energy overheads are accounted for within the energy consumption rate p_i of the task and the static energy rate of each core, i.e., P_{static}.

2.2.2 Energy supplier model

The energy supplier consists of an energy harvester and an energy storage unit, with the energy harvester replenishing energy from a renewable source. At each time step s, an energy predictor provides a pessimistic estimate of the energy charging rate denoted as $P_R(s)$ for step $0 \leq s \leq H$, where H is the prediction time horizon. The replenished energy is intermittent, and the energy harvested across different steps may not necessarily be constant.

Additionally, an energy storage device in the form of a battery with a nominal capacity of B is considered. The battery's initial charge is indicated as IE. Similar to the approach in [36], the model assumes the possibility of overlapping charge and discharge intervals.

The size of the battery capacity B and the nature of the energy source play pivotal roles in system performance. Large batteries, which can act as substantial energy buffers, offer a degree of isolation from fluctuations in harvested energy. However, energy-aware scheduling remains essential even in such scenarios due to unpredictable environmental conditions and battery degradation over time. Conversely, small battery capacities, typical in compact and resource-constrained devices like CubeSats[51] or unmanned aerial vehicles, introduce more immediate challenges. In these cases, dynamic energy scheduling is necessary to ensure that task deadlines are met based on real-time energy availability.

3 Background and problem definition

This section discusses some necessary backgrounds for the system under study and the problem definition. We commence with a brief overview of the federated scheduling algorithm in multi-core systems. Subsequently, we discuss time-related boundaries in executing parallel tasks. We then examine the number of cores needed for each parallel task, followed by introducing a data structure for representing the number of cores allocated to each parallel task. Finally, we define the problem.

3.1 Federated scheduling strategy

The federated scheduling strategy was introduced in [21] as a generalized approach to partitioned scheduling for parallel tasks and has since become widely used in real-time parallel scheduling research. This strategy categorizes tasks as **high-utilization**

if their utilization is greater than 1, or **low-utilization** otherwise. High-utilization tasks are assigned a dedicated cluster of cores. At the same time, a single processor scheduling algorithm schedules the low-utilization tasks, which are executed sequentially.

In our proposed system, each task has a dedicated set of cores, upon which it is scheduled by a greedy (or work conserving) scheduler. It never keeps a core idle if there is ready work available. A greedy scheduler executes each task τ_i on its dedicated cores. Such a scheduler is introduced in [52]. The details of the greedy scheduler's implementation are beyond the scope of this paper. Throughout this paper, when we refer to the execution of a parallel task, we mean the execution of the task's DAG by the greedy scheduler on the task's dedicated cores. Conversely, when we mention suspending a parallel task, we are indicating the suspension of the greedy scheduler associated with the respective task, leading to the interruption of the task's execution.

During each step, i.e., s, a core can either be idle or perform one unit of a job. A step is *complete* if none of the cores assigned to task τ_i are idle during that step; otherwise, it is an *incomplete* step [53]. Figure 1 shows an example of a task's complete and incomplete steps. In this figure, task τ_i is assigned four cores. In the first step, all four cores are utilized. In the second step, one core is used, followed by two cores in the third step, and so on. Lemma 1 explains the possible circumstances for running the task τ_i.

Lemma 1. *When a task τ_i is executed by a greedy scheduler on its dedicated cores, the maximum number of incomplete steps is equal to the critical path length of the task, i.e., L_i. [53][54][55].*

According to Lemma 1, during the execution of the task τ_i, there will be a maximum of τ_i incomplete steps, and the rest of the steps are necessarily complete. Therefore, despite the uncertainty of how each task runs on its cores, possible and impossible cases can still be considered. For example, if in Figure 1 the size of the critical path is 4, the method of execution shown in this figure would be a possible execution of task τ_i on its cores. However, if the size of the critical path is equal to 3, this schedule will not be possible. Because, according to Lemma 1, the number of incomplete steps cannot be greater than the length of the critical path. Next, in Section 5, we will use the concept of complete and incomplete steps to analyze the worst-case energy consumption of tasks.

3.2 Tasks' execution time boundaries

As mentioned in the previous section, given the number of assigned cores for a task, we may see different execution times per job because of the uncertainty in DAG execution. We use (3) as the upper bound of execution time or worst case execution time (WCET) of task τ_i on m cores which was introduced in [56] for computing the worst-case response time of tasks scheduled with any greedy schedulers, and then formalized in [57], [58] for DAG tasks.

$$W_{i,m}^{max} = \lfloor \frac{C_i - L_i}{m} \rfloor + L_i \tag{3}$$

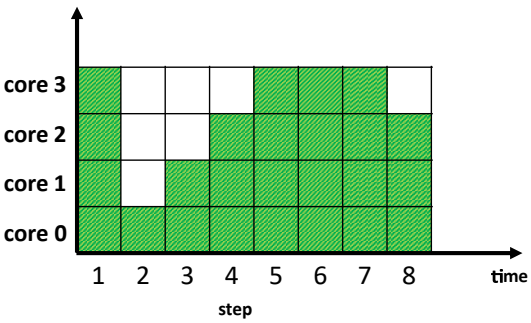


Fig. 1: An example of running a task on 4 cores. The solid squares indicate that the core is busy, and the empty squares indicate that the core is idle. Steps 1,5,6,7 are complete and steps 2,3,4,8 are incomplete.

Note that in this equation we round the fraction $\frac{C_i-L_i}{m}$ down because if we round it up, the equation $\lceil\frac{C_i-L_i}{m}\rceil + L_i$ can generate $L_i + 1$ incomplete step (one possible incomplete step in $\frac{C_i-L_i}{m}$ and L_i incomplete steps in the following) which is in contradiction with Lemma 1. Moreover in (3), if $m = \infty$, the execution time will be equal to L_i and if $m = 1$, the execution time will be equal to C_i.

The lower bound of the execution time of task τ_i is always L_i and execution time will never be smaller than L_i, even with infinite cores. If the task τ_i contains m independent and parallel threads, the execution time can be equal to $\lceil\frac{C_i}{m}\rceil$ as long as the value of $\lceil\frac{C_i}{m}\rceil$ is not smaller than L_i. Therefore the lower bound of execution time of task τ_i can be calculated according to the following equation:

$$W_{i,m}^{min} = max(\lceil\frac{C_i}{m}\rceil, L_i). \tag{4}$$

Li et al. [21] have shown that if the parallel task τ_i with the implicit relative deadline of D_i is assigned the number of n_i^{min} specific cores where:

$$n_i^{min} = \lceil\frac{C_i - L_i}{D_i - L_i}\rceil \tag{5}$$

then, by using a greedy scheduler, all the jobs of task τ_i meet their respective deadlines. In other words, the minimum number of cores required to meet the deadline of task τ_i is equal to n_i^{min} and increasing the number of cores, makes more slack to execute the task. We define the sum of the minimum number of cores required for all tasks as follows:

$$minCores = \sum_{\tau_i \in \tau} n_i^{min} \tag{6}$$

we can determine the smallest m such that any additional cores do not significantly reduce execution time. Solving $\frac{C_i-L_i}{m} \leq 1$ leads to $m \geq C_i - L_i$. Adding 1 to

account for any remainder ensures that the critical section is properly managed, giving $n_i^{\max} = C_i - L_i + 1$. Hence, once the number of cores exceeds $C_i - L_i$, the execution time becomes constant at L_i, and additional cores will not reduce the execution time further. Therefore, the upper limit of the number of cores will be:

$$n_i^{max} = C_i - L_i + 1 \tag{7}$$

3.3 Effective number of cores

According to (5), each task requires a minimum number of cores to meet its deadline. The maximum number of cores is also calculated by (7). Any number of cores between the minimum and the maximum can be used to meet the deadline. However, we are only interested in the number of cores that cause a change in the execution time of parallel task τ_i in (3), and among all the number of cores that have the same execution time, we choose the smallest one. Thus, we define the vector of *effective number of cores* of task τ_i as $\eta^i = (\eta_1^i, \eta_2^i, ..., \eta_{|\eta^i|}^i)$ where η_1^i equals to n_i^{min}, η_j^i is the minimum number of cores that $W_{i,\eta_{j-1}^i}^{max} > W_{i,\eta_j^i}^{max}$ and $|\eta^i|$ is the size of vector η^i. More formally η^i defined as:

$$\eta^i = \begin{cases} \eta_j^i = min\{\eta_{j-1}^i < k \leq n_i^{max} | \lfloor \frac{C_i - L_i}{\eta_{j-1}} \rfloor \neq \lfloor \frac{C_i - L_i}{k} \rfloor\} \\ \eta_1^i = n_i^{min} \end{cases} \tag{8}$$

3.4 Core assignments matrix

In Section 3.3, we introduced the vector η^i to represent the effective number of cores for task τ_i . Each task τ_i can meet its deadline when executed with any value from its corresponding vector η^i. Furthermore, a higher number of cores allows for more slack in its execution. To determine the appropriate number of cores for each task in the system, we consider the possibility of executing each task with different values from its η^i vector, resulting in varying levels of slack. To accommodate this flexibility, we construct the matrix ca, as shown in (9), which integrates these different combinations of core numbers. Each column in ca corresponds to a task, and each row ca_j in this matrix features distinct values, such as column i, representing an alternative value from the corresponding η^i of task τ_i. Therefore, $ca_{j,i}, 1 < i \leq n$, represents the number of cores assigned to task τ_i in core assignment j. The rows in the matrix ca are arranged in ascending order based on the total number of cores allocated to the tasks in each row. Rows that exceed the system core or N are eliminated.

$$ca = \begin{bmatrix} \eta_1^1 & \eta_1^2 & \cdots & \eta_1^n \\ \eta_1^1 & \eta_1^2 & \cdots & \eta_2^n \\ \vdots & \vdots & \ddots & \vdots \\ \eta_{|\eta^1|}^1 & \eta_{|\eta^2|}^2 & \cdots & \eta_{|\eta^n|}^n \end{bmatrix} \tag{9}$$

The matrix ca is constructed by systematically combining core assignments from the different η^i vectors. While the lengths of the η^i vectors may differ, the matrix ca integrates all possible combinations of core assignments from each η^i.

Section 7 will employ the *ca* matrix to investigate core assignments that enable the attainment of a feasible schedule for the system tasks

3.5 Problem statement

In this paper, we focus on high-utilization tasks and assume the absence of low-utilization tasks (because the scheduling of low-utilization tasks is studied sufficiently in the literature (see [59][36] [37][38])). According to the federated scheduling strategy, in the presence of low-utilization tasks, they are aggregated onto a single core for management via single-threaded scheduling algorithms. Our emphasis on high-utilization tasks does not limit the system's ability to handle low-utilization tasks; instead, it highlights the prioritization of tasks with significant energy demands, which are crucial for the system's missions. Should the inclusion of low-utilization tasks become necessary, we propose two seamless integration strategies: (1) developing a unified energy consumption model for aggregated tasks on a single core, enabling efficient energy scheduling alongside high-utilization tasks; and (2) categorizing tasks based on their energy demands and core requirements into low- and high-utilization groups. Low-utilization tasks are aggregated into groups where the total utilization of each group does not exceed the capacity of a single core, ensuring efficient core usage. These groups are then assigned to individual cores, while high-utilization tasks are allocated to the remaining cores. For scheduling, low-utilization groups are managed using energy-aware algorithms such as PFP_{ASAP} [38], whereas high-utilization tasks continue to be scheduled using the proposed method in this paper.

Therefore, the objective is to schedule the high-utilization tasks to ensure their deadlines are met while accounting for the harvested energy constraints. A task set is considered *schedulable* when all tasks within the set meet their respective deadlines. In the following, we define the problem, then we examine the computational complexity of it.

Problem 1. *Given task set τ, input energy rate P_R, battery capacity B, total number of system's core N, the problem is to find a core assignment to each task τ_i and performing the corresponding scheduling such that the task set be schedulable.*

Since each task in our system operates on its dedicated cores, there is no time contention between tasks. However, due to constraints on input energy, tasks contend for power. The power request of each task, along with the harvested energy rate, determines whether a task needs to wait for execution of higher-priority tasks, executing in series with them, or if it can run in parallel without waiting. Therefore, following the determination of the number of cores for each task, the power consumption pattern of each task may change, discussed in Section 5. Considering the limited harvested energy rate, this change can delay the execution of lower priority tasks. Therefore, meeting the deadline for each task is contingent on the harvested energy rate, the task's core count, and the core count of tasks with higher priority.

The following theorem shows the computational complexity of Problem 1.

Theorem 2. *The problem of determining a feasible schedule for the system tasks while considering input energy constraints, i.e., Problem 1, is NP-hard.*

Proof. See A. □

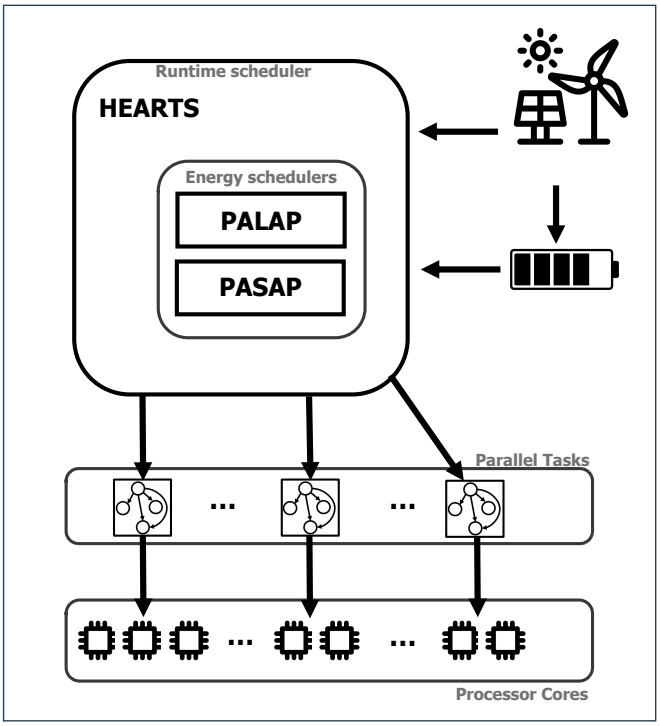


Fig. 2: An abstract representation of the proposed scheduling model. The HEARTS runtime scheduler dynamically manages harvested energy from an environmental source and assigns it to parallel tasks via PASAP and PALAP energy schedulers.

In Section 9, we further discuss how the battery capacity affects the complexity of the problem. We show that this problem can be solved in polynomial time with an infinite-capacity battery.

4 Overview of the proposed method

As explained in Sections 2 and 3, our system consists of high-utilization parallel tasks, each of which can be assigned a specific number of cores. The energy required to power the system is harvested from the environment, which is subject to fluctuations. We aim to ensure 1) all tasks are completed within their deadlines and 2) minimize the number of cores, while considering the input energy rate and battery capacity. If energy fluctuations or high-priority tasks cause some tasks to miss their deadlines, we may adjust the execution time and energy consumption of some tasks by changing the number of cores assigned to them to reach a feasible schedule.

We propose a runtime scheduler, namely **HEARTS** (Harvesting Energy Aware Runtime Task Scheduler), that divides the scheduling horizon into multiple windows. Each window represents a specific time interval. HEARTS selects a window and calls the **energy scheduler** to schedule the tasks based on the input energy within that

Algorithm 1: Overview of HEARTS (Harvesting-Energy Aware Runtime Task Scheduler)

```
1  i ← 0;
2  while not reached the end of scheduling horizon do
3      Choose i^th scheduling window;
4      Set the minimum core for each task, corresponding to the first row of the
         core assignment matrix ( section 3.4);
5      Calculate the energy consumption of each task, i.e., φ(s) according to (10)
         for assigned number of cores;
6      Call the PALAP (The Last-Fit based energy scheduler, Algorithm 5);
7      if failed to schedule using the Last-Fit algorithm then
8          Call the PASAP (The First-Fit based energy scheduler, Algorithm 6);
9      if not schedulable with minimum cores using both algorithms then
10         while core search not exhausted do
11             Assign the next row of the core assignment matrix to each task;
12             Calculate the energy consumption of each task for the new number
                 of cores using (10);
13             Call the PALAP ;
14             if failed to schedule using the PALAP algorithm then
15                 Call the PASAP ;
16             if feasible schedule found then
17                 break;
18     if no feasible schedule found then
19         Fail;
20     Run tasks on assigned cores;
21     Update battery energy level and reserved energy state for the next
         scheduling window;
22     i ← i + 1;
```

window and the battery capacity. Initially, HEARTS assigns the minimum number of cores to each task and calculates their energy consumption based on this allocation. Figure 2 shows an abstract representation of our proposed method.

Algorithm 1 shows an abstract view of HEARTS. At the start of each window, HEARTS initially assigns the minimum number of cores, calculated by (5), to each task. Then, it calculates the energy consumption for each task with the assigned cores. In Section 5, we provide an in-depth analysis of the worst-case energy consumption for parallel tasks with a specific number of dedicated cores.

HEARTS then calls the energy scheduler. We propose two energy schedulers in this paper. The first algorithm, employed by HEARTS, delays the allocation of requested energy as much as possible using the last-fit method. In contrast, the second algorithm uses the first-fit method, which allocates energy as soon as it becomes available. In Section 9, we will prove that the former algorithm is optimal where the battery capacity

is greater than a specific threshold. Therefore, HEARTS first checks whether the task set is schedulable by the first algorithm; if not, it tries the second one. Furthermore, in Section 9, we will show a case where the first-fit algorithm outperforms the last-fit algorithm when the battery capacity is smaller than the threshold value.

If the minimum number of cores assigned to each task fails to meet their respective deadlines, HEARTS employs an alternative core assignment method using the core assignment matrix introduced in Section 3.4. As previously discussed, each row in this matrix represents a potential core assignment for tasks. These rows are organized in ascending order based on the total number of cores, and any rows exceeding the system core count or N are excluded. This ensures that every potential configuration is considered during the core assignment process, providing a comprehensive exploration of the design space. Even if multiple tasks' core assignments are modified simultaneously in some iterations, the structured construction of core assignment matrix guarantees that all valid combinations are thoroughly explored.

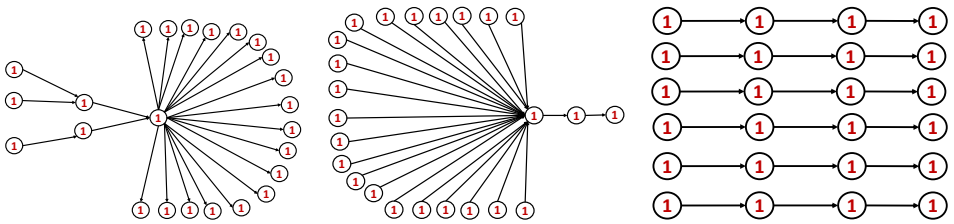
The energy scheduling process in HEARTS is repeated using both algorithms until a feasible schedule is obtained. If a feasible schedule cannot be achieved, the scheduling process fails.

In Section 5, we introduce a function that determines the energy consumption of each parallel task at any given Step s during its execution. Then, we will elaborate on the implementation details of energy scheduler in Section 6, which are our proposed method's most critical scheduling component. In Section 7, we provide a detailed description of the HEARTS, including how it schedules the static energy of cores, determines on/off times for extra cores, and computes the remaining energy in the battery for the next scheduling window.

5 An analysis of power consumption of parallel tasks

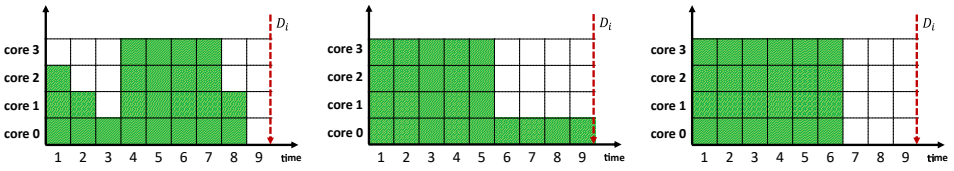
An important question about parallel tasks is how they consume energy during their runtime. For instance, suppose m cores are assigned to the parallel task τ_i. In this case, what is the amount of energy required for τ_i in each step? Single-threaded tasks use only one core at each step of execution, and the energy consumption rate is considered a fixed value p_i, measured as the worst-case amount of energy consumption. The assumption of energy consumption being proportional to WCET and single-core power provides a conservative estimate for ensuring task deadlines under worst-case conditions, even though empirical studies, such as those by [60] and [61], indicate that real-world energy profiles may differ. In parallel tasks, however, the number of cores used in each step of execution is not certain, and as a result, the amount of energy required in each step is unclear. To clarify this issue, we will first examine an example.

Example 1. *Consider the parallel task τ_i with the following parameters: total execution time $C_i = 24$, critical path length $L_i = 4$ and implicit deadline $D_i = 9$. Figure 3 shows three possible DAGs for this task. At runtime, each job of this task can utilize the cores assigned to it in different ways. We examine the different modes of execution for this task by considering four cores. Figure 4 shows some of the possible execution modes for this task.*



(a) (b) (c)

Fig. 3: Three different DAGs with $C_i = 24$ and $L_i = 4$ for task τ_i of Example 1.



(a) $W_{i,4}^{max} = 8$ (b) $W_{i,4}^{max} = 8$ (c) $W_{i,4}^{max} = 6$

Fig. 4: Three possible scenario of utilizing cores by task τ_i of Example 1. The solid squares indicate that the core is busy, and the empty squares indicate that the core is idle.

As shown in Figure 4, if we assign four cores to the task τ_i, in each step, a different number of cores may be utilized by it, and the exact number of them is not known in advance. This uncertainty can lead to different execution times and, hence, different dynamic energy consumption, even for different jobs of a task. For example, the task τ_i in the first step may utilize three core (Figure 4a) or all four cores (Figure 4b and (Figure 4c)). In the following, we present a method for calculating the worst-case energy demand of a parallel task.

Theorem 3. *The worst-case energy demand of parallel task τ_i with m dedicated cores occurs when this task has its shortest possible execution time, i.e., the task's execution time is $\lceil \frac{C_i}{m} \rceil$.*

Proof. At each step, the maximum energy demand occurs when all of the m allocated cores are used. The worst-case energy demand of task τ_i also occurs when the maximum amount of energy is requested in all steps, in which case the execution time will be equal to $\lceil \frac{C_i}{m} \rceil$. □

According to Theorem 3, for the successful execution of the task τ_i with m dedicated cores, in the first $\lceil \frac{C_i}{m} \rceil$ steps, $m * p_i$ units of energy must be provided in each step. In this case, the execution time will be considered to be $\lceil \frac{C_i}{m} \rceil$, and also considering the worst execution time of the tasks, which introduced in (3), we conclude that this method of energy allocation is very pessimistic, and it can be adjusted according to the upper bound of execution time, i.e., $W_{i,m}^{max}$. The following theorem represents the minimum energy allocation of a parallel task in each step.

Theorem 4. *The minimum energy allocation in Step s to the task τ_i with m dedicated cores, where $1 \leq s \leq W_{i,m}^{max}$, is equal to $\varphi_i^m(s)$ where the function φ is as follows:*

$$\varphi_i^m(s) = \begin{cases} m * p_i & s \leq \lceil \frac{(C_i - L_i)}{m} \rceil \\ p_i & \lceil \frac{(C_i - L_i)}{m} \rceil < s \leq C_i - (m-1) * \lceil \frac{C_i - L_i}{m} \rceil \\ 0 & o.w. \end{cases} \tag{10}$$

Proof. See B. □

As mentioned earlier, before the execution of the tasks, there is no precise information about the number of cores used in each step. Hence, there is no knowledge about the amount of energy consumed in each step. Therefore, as shown in Theorem 4, energy is allocated to the tasks based on the worst-case scenario.

The question arises that if the worst-case scenario of energy demand does not occur (i.e., in one step, fewer cores than the amount proposed in Theorem 4 is needed), what should be done with the surplus energy supplied to the job? In this case, the surplus energy of one step must be stored and used for the next steps of the same job. To store the excess energy of one step, which is caused by not following the worst-case scenario of energy consumption, dedicated energy storage is used. It means that the energy stored in this storage is only for the current job, and until the completion of the current job, it cannot be used in other tasks.

Note that there is no need for a separate battery for this purpose. Instead, part of the system's battery capacity can be reserved for this goal. In the following, we prove the expected capacity of this dedicated energy storage for each task.

Theorem 5. *For each parallel task τ_i, we need to reserve a part of the system's energy storage capacity with the following minimum amount:*

$$B_{res}(i) = \begin{cases} L_i * (m-1) & L_i \leq \lceil \frac{C_i - L_i}{m} \rceil \\ \lceil \frac{C_i - L_i}{m} \rceil * (m-1) & otherwise \end{cases} \tag{11}$$

Proof. The maximum amount of energy storage capacity for a task occurs at the step when the maximum amount of energy is provided and the minimum amount of energy consumed. Given the definition of the greedy scheduler and the power supply based on (10), it can be easily shown that in Step $\lceil \frac{C_i - L_i}{m} \rceil$ the sum of the supplied energy is at the maximum amount. Similarly, it can be shown that the minimum energy consumption occurs when only one unit of energy is consumed in each step. Hence the residual energy in each step is equal to $(m-1)$, and the number of these steps is equal to $min(L_i, \lceil \frac{C_i - L_i}{m} \rceil)$. □

6 PALAP and PASAP, the proposed energy scheduling algorithms

In this section we present two algorithms that are designed to assess the schedulability of a core assignment. These algorithms are inspired by classic strip-packing heuristics, such as first-fit and last-fit [62][63][64]. However, the parallel energy supply of tasks presents a challenge to the straightforward application of these heuristics, as high-priority tasks may starve while waiting for energy to accumulate. In contrast, low-priority tasks consume energy in the battery. To overcome this challenge, we incorporate the idea of energy reservation into our algorithms. In the following sections, we present two algorithms based on first-fit and last-fit that implement energy reservation.

The two algorithms that will be introduced in this section share a common process. In each iteration, these algorithms assess the schedulability of all jobs associated with a task over the scheduling horizon. As a result, the number of iterations in these algorithms will equal the number of tasks in the system. Starting with the highest-priority task, the first iteration schedules energy demands for all its jobs within the time interval $[0, horizon]$. If it is not possible to fulfill each of energy demands within the desired deadline, the algorithm considers the deadline as missed. The energy scheduling for the next-highest-priority task is carried out only if all energy demand s for the current highest-priority task have been met. The task set is considered schedulable when the last job of the lowest-priority task receives energy and meets its deadline.

6.1 Framework of the energy scheduling algorithms

As a prerequisite to introducing the energy scheduling algorithm, we present a few data structures to track the energy consumption status. The energy consumed in each step is stored in the *used* vector, denoted by $used_s$, which is equal to the total energy consumed in Step s. Moreover, reserved energy in each Step s is tracked using the res vector, denoted by res_s. The details of these vectors are as follow:

- **used** vector: The energy consumed in each Step s, i.e., $used_s$, equals to the total energy consumed in Step s. Since the battery can transfer energy from previous steps to Step s, the value of $used_s$ can be greater than the energy replenished at Step s.
- **res** vector: Reserved energy in Step s, i.e., res_s, represents the quantity of replenished energy that will be consumed in the following steps or which has been transferred from the previous steps to the current one. The res_s can be as follows:
  - $\mathbf{res_s = 0}$: In this step, the battery does not need to transfer energy from previous steps since the energy replenished in Step s provides all the energy required in this step. In addition, no energy is reserved for the next steps from the total energy replenished.
  - $\mathbf{res_s > 0}$: As of this point in the scheduling, the energy demands in Step s have all been met by the energy replenished in Step s itself, which means energy was not transferred from other steps to this step from the battery. In addition, the battery reserves the res_s unit of the replenished energy for later use.

– $\mathbf{res_s < 0}$: The battery will transfer $|res_s|$ units of energy replenished from previous steps to this step in addition to consuming all the energy replenished in Step s. By the end of this step, $|res_s|$ units of battery energy will be unreserved and available to use.

When there is an energy demand of size **Dem** in Step s and the i^{th} iteration of the algorithm, the energy demand **Dem** can be supplied from two sources. The first is the replenished energy in the same Step s, which, of course, in iterations 0 to $i-1$ of the algorithm execution, some of this energy was consumed for higher priority tasks in the same step or reserved for their use in the upcoming steps. The second energy source is the energy transferred from previous steps to this one and is not reserved for higher priority tasks. In the following sections, we will discuss these two sources.

6.2 Available battery energy calculation

Algorithm 2: *BatteryEn*(s)

Input : s, P_R, IE, B, $used$, res
Output: Available battery energy at step s

```
1  r ← 0
2  ABE ← 0
3  while r ≤ s do
4      if r = 0 then
5          ABE ← min (ABE + P_R(r) + IE − used_r, B)
6      else
7          if (res_r ≥ 0) then
8              ABE ← min (ABE + P_R(r) − used_r, B)
9          else
10             ABE ← ABE + res_r
11     r ← r + 1
12 return ABE ;
```

At Step s, some energy replenished from the previous steps may be in the battery, with some reserved for a specific step in the future. The rest can be used in Step s itself. To determine the available battery energy, denoted by ABE, we must examine the $used$ and res vectors in each step and calculate the amount of unused energy. Accordingly, we calculate the value of ABE from step 0 to Step s as follows:

- If in Step r, $0 \leq r < s$, $\mathbf{res_r \geq 0}$, then this means that all previous energy demands in this step are met with the replenished energy, and no energy is transferred from previous steps. In this case, the battery is charged with the following amount of energy:

$$ABE = min(ABE + P_R(r) - used_r, B) \tag{12}$$

Where $used_r$ represents the amount of energy consumed in Step r. Likewise, the energy transferred through the battery cannot exceed its capacity. Note that in (12), the amount of energy added to the battery includes the amount of energy reserved in this step.

- If in Step r, $0 \leq r < s$, $\mathbf{res_r} < \mathbf{0}$, then this means that the replenished energy in Step r was insufficient to meet the demands of the higher priority tasks, and battery energy was consumed in the amount of $|res_r|$. In this case, the amount of energy in the battery will change as follows:

$$ABE = ABE + res_r \tag{13}$$

Not that in (13), the available battery's energy decreases due to the negative value of res_r.

Algorithm 2 calculate the amount of available battery energy at the end of Step s. This algorithm takes as input the desired step s, replenishment energy rate P_R, initial energy in the battery IE, battery capacity B, and vectors *used* and *res*. It returns the available battery energy at step s as output.

6.3 Unallocated replenished energy

At each iteration of algorithm, in Step s, the amount of energy left from the total replenished energy varies according to the values of *used* and *res* vectors. If the value of $res_s < 0$, it means that high-priority tasks have consumed all the replenished energy in Step s, and no energy is left for other tasks. Otherwise, available replenished energy equals to the replenished energy minus the sum of the energy consumed in this step and the amount of energy reserved for the next step. We show the unallocated energy from the replenishment with $ure(s)$ at each Step s. It is also important to consider the battery's initial energy at Step 0. Moreover, in Step 0, no energy can be transferred from previous steps, and res_0 cannot be negative. $ure(s)$ can be calculated as follows:

$$ure(s) = \begin{cases} P_R(s) + IE - used_s - res_s & s = 0 \\ P_R(s) - used_s - res_s & res_s \geq 0, s \neq 0 \\ 0 & res_s < 0, s \neq 0 \end{cases} \tag{14}$$

As a result, the total available energy in Step s will be as follows:

$$\mathrm{En}(s) = \mathrm{BatteryEn}(s-1) - \sum_{0 \leq r < s} res_r + \mathrm{ure}(s) \tag{15}$$

In each step, the energy is supplied by two sources; the first is the energy replenished in the same step, and the second is the energy transferred by the battery from previous steps. With an energy demand of size Dem at Step s, if $Dem \leq En(s)$, this energy demand can be met, and the energy demand can be supplied. Based on how energy is supplied in Step s, we will have the following conditions:

- $\mathbf{ure(s)} \geq \mathbf{Dem}$: In Step s, the energy required for this Dem has been replenished, and it can be supplied. No energy needs to be reserved in the previous steps, nor does the battery need to be used.

- **ure(s) < Dem**: The replenished energy in Step s will not be sufficient to satisfy the energy demand of Dem. The battery must reserve $R = Dem - \text{ure}(s)$ of the filled energy in the previous steps to transfer to Step s.

If there is a need to reserve energy, i.e., $R > 0$, then energy should be reserved in the previous steps. The two main approaches are to reserve energy from Step $g = s-1$ to Step 0, called ***backward reserve*** (see Algorithm 4), or to reserve energy from Step 0 to Step $g = s - 1$, called ***forward reserve*** (see Algorithm 3). Both of these algorithms take the amount of energy R that should be reserved in previous steps to use at step s as input and return the reservation vector res as output.

The forward reservation method occupies the battery for a longer period of time than the backward reservation method, which leads to more harvested energy waste. Therefore, we will only use the backward reservation method in this paper.

Algorithm 3: Forward Reserve

Input : R, s
Output: Updated reservation vector res

```
1 g ← 0
2 while (R > 0 and g < s) do
3     if (ure(g) > 0) then
4         temp ← Max(ure(g), R)
5         R ← R − temp
6         res_g ← res_g + temp
7     g ← g + 1
```

Algorithm 4: Backward Reserve

Input : R, s
Output: Updated reservation vector res

```
1 g ← s − 1
2 while (R > 0 and g ≥ s) do
3     if (ure(g) > 0) then
4         temp ← Max(ure(g), R)
5         R ← R − temp
6         res_g ← res_g + temp
7     g ← g − 1
```

6.4 Details of the algorithms

Two algorithms are presented in this section for determining the energy schedulability of parallel tasks. These algorithms evaluate the schedulability of tasks with a given core

Algorithm 5: PALAP

Input : τ, P_R, $used$, res, ω, B, IE
Output: True if task set is schedulable, False o.w.

```
1  start ← 0
2  Horizone ← ω
3  foreach task τ_i ∈ τ do
4      dl ← D_i − 1
5      start ← Φ_i
6      while (dl ≤ Horizone) do
7          s ← dl
8          j ← W_{i,m}^{max} − 1
9          while j ≥ 0 do
10             Dem ← φ_i^m(j)
11             if (Dem = 0) then
12                 Dem ← −1
13                 s ← s − 1
14             else
15                 while (s ≥ start and Dem ≠ −1) do
16                     if Dem ≤ (En(s)) then
17                         R ← 0
18                         if (ure(s) < Dem) then
19                             R ← Dem − ure(s)
20                         res_s ← res_s − R
21                         BackwardReserve(R, s)
22                         used_s ← used_s + Dem
23                         Dem ← −1
24                     s ← s − 1
25             if s < start and Dem ≠ −1 then
26                 return false
27             j ← j − 1
28         start ← start + D_i;
29         dl ← start + D_i;
30 return true
```

configuration over a scheduling horizon H and with an input energy function P_R. Both algorithms take as input the task set τ, replenishment energy rate P_R, vectors *used* and *res*, scheduling horizon ω, battery capacity B, and initial energy in the battery IE. The algorithm returns `True` if the task set is schedulable, and `False` otherwise.

In the first algorithm, which we call the PALAP method, energy demands are scheduled *as late as possible* following the principle of the last fit. A second algorithm termed the PASAP method, works on the idea of the first fit and attempts to schedule

Algorithm 6: PASAP

Input : τ, P_R, $used$, res, ω, B, IE
Output: True if task set is schedulable, False o.w.

```
1  start ← 0
2  Horizone ← ω
3  foreach task τ_i ∈ τ do
4      dl ← D_i − 1
5      start ← Φ_i
6      while (dl ≤ Horizone) do
7          s ← start
8          j ← 0
9          while j < W_{i,m}^{max} do
10             Dem ← φ_i^m(j)
11             if (Dem = 0) then
12                 Dem ← −1
13                 s ← s + 1
14             else
15                 while (s < dl and Dem ≠ −1) do
16                     if Dem ≤ (En(s)) then
17                         R ← 0
18                         if (ure(s) < Dem) then
19                             R ← Dem − ure(s)
20                         res_s ← res_s − R
21                         BackwardReserve(R, s)
22                         used_s ← used_s + Dem
23                         Dem ← −1
24                     s ← s + 1
25             if s ≥ start + D_i and Dem ≠ −1 then
26                 return false
27             j ← j + 1
28         start ← start + D_i;
29         dl ← start + D_i;
30 return true
```

energy demands *as soon as possible* to provide enough energy to meet an energy demand. Both algorithms, Algorithms 5 and 6, at first, check the energy scheduling of all jobs associated with the highest priority task during the scheduling horizon. If They are schedulable then the algorithms move on to the next highest priority task.

The energy scheduling aims to assign each energy demand Dem of task τ_i to a time interval between τ_i's release time and deadline. The PALAP algorithm checks

schedulability from the deadline to the release time of the task. In contrast, the PASAP algorithm checks schedulability from the release time to the deadline.

For each execution unit j of task τ_i, where $1 \leq j \leq W_{i,m}^{max}$, (10) is used to calculate the minimum energy allocation, i.e., $\varphi_i^m(j)$. Since the value of $\varphi_i^m(j)$ may be zero, both algorithms check for zero energy demand in Lines 11 to 13. If it is, they leave an empty step. After that, if an energy reserve is needed, energy is reserved according to Algorithm 3, and *res* and *used* vectors are updated. The feasibility of the energy supply is checked until the job release time in the PALAP algorithm and until the deadline is reached in the PASAP algorithm. A deadline miss occurs when the current job reaches its deadline under the PALAP method or reaches its release time under the PASAP method, and there is an unscheduled execution step.

7 Details of HEARTS

In the previous section, we introduced two methods for evaluating the energy scheduling of a set of tasks. These methods allowed for the assessment of task schedulability and provided detailed information on the energy scheduling of tasks at each step.

While PALAP and PASAP are effective under specific conditions, they are designed to function over a fixed scheduling horizon. In systems with complex task sets, determining an appropriate scheduling horizon—such as the hyperperiod—can lead to computationally impractical scheduling windows. This issue is particularly pronounced when the hyperperiod becomes exponentially large, rendering the direct application of PALAP and PASAP inefficient or even infeasible. Moreover, these algorithms are not inherently dynamic; they do not respond to real-time changes in harvested energy or system states. This lack of adaptability limits their practicality in environments where energy availability fluctuates.

To overcome these challenges, we propose HEARTS, which checks the energy scheduling and searches for the number of cores required to meet deadlines within smaller intervals of the scheduling time, called windows. By following this approach, a feasible schedule can be found with the required number of cores for each window and a possible energy scheduling can be provided for the next window before the current window's task execution is complete. In our proposed method, the window size can be determined according to system specifications. The window size indicated by the variable win_{size} in our algorithm is set to the maximum deadline of the system's set of tasks to ensure that at least one job of each system task is executed within a window. We use the variables α and β to mark the beginning and end of each window, respectively.

To maintain the scheduling status, the PASAP and PALAP algorithms use the *used* and *res* vectors as mentioned in Section 6.1. These algorithms schedule tasks from step zero to the desired *horizon*, i.e., window size. As shown in Algorithm 7, to schedule tasks and check the schedulability with a specific number of cores assigned to each task, we introduce temporary vectors named $used'$ and res', representing a part of the original *used* and *res* vectors. Specifically, the values between α and β in the *used* and *res* vectors are copied to the temporary vectors, starting from position 0 to $\beta - \alpha$. The scheduling algorithms update the $used'$ and res' vectors, and after

Algorithm 7: HEARTS (Harvesting-Energy Aware Runtime Task Scheduler)

$\alpha \leftarrow 0$
$\beta \leftarrow win_{size}$
while $(\beta \leq H)$ **do**
  $\omega \leftarrow \beta - \alpha$
  $winPR_i = PR_{\alpha+i}, \quad \text{for } i = 1, \ldots, \omega$
  $minCores \leftarrow \sum_{i=0}^{n} ca_{0,i}$
  `// Create a new virtual task set` τ^s `for scheduling of static energy of cores.`
  Add new virtual task $\tau_0^s \leftarrow (\Phi_0^s = 0, D_0^s = win_{size}, p_0^s = minCores * P_{static})$ to τ^s
  $totalReserverd \leftarrow 0$
  $\tau^d \leftarrow \tau$ `// Initializing the temporary task set`
  $j \leftarrow 0$
  while ($j <$ number of rows of ca) **do**
    if $(minCores < \sum_{i=0}^{n} ca_{j,i})$ **then**
      foreach *Core number for task* $\tau_i^d \in \tau^d$ *with* $ca_{j,i} > ca_{0,i}$ **do**
        add new virtual task $(\Phi^s = \Phi_i^d, D^s = \omega - ((\omega - \Phi_i^d) \bmod D_i^d), p^s = (ca_{j,i} - ca_{0,i}) * P_{static})$ to τ'
    $\tau' \leftarrow \tau^s + \tau^d$ `// Append the main tasks to the virtual tasks`
    $used'_i = used_{\alpha+i}, \quad \text{for } i = 1, 2, \ldots, \omega$
    $res'_i = res_{\alpha+i}, \quad \text{for } i = 1, 2, \ldots, \omega$
    $res'_0 \leftarrow res'_0 + totalReserverd$
    result =PALAP (τ',$winPR'$,$used'$,res',ω, B, IE)
    if (***not*** *result*) **then**
      result =PASAP (τ',$winPR'$,$used'$,res',ω, B, IE)
    if (*result* ***and*** $minCores < \sum_{i=0}^{n} ca_{j,i}$) **then**
      `// Considering the static energy of additional cores`
      for *(each task with core more than* $ca_{0,i}$*)* **do**
        Turn on $(ca_{j,i} - ca_{0,i})$ Cores at $\alpha + \Phi_i^d$
        And turn them off at $\beta - ((\omega - \Phi_i^d) \bmod D_i^d)$
    if (*result*) **then**
      break
    $j \leftarrow j + 1$
  $\chi = \min_{\tau_i^d \in \tau^d} (\omega - ((\omega - \Phi_i^d) \bmod D_i^d))$
  $\Phi_i^d = (\omega - ((\omega - \Phi_i^d) \bmod D_i)) - \chi) \quad \text{for } \tau_i^d \in \tau^d$
  $IE = BatteryEn(\chi - 1)$
  $totalReserverd = \sum_{0 \leq r < \beta} res_r$
  $used_{\alpha+i} = used'_i, \quad \text{for } i = 1, 2, \ldots, \omega$
  $res_{\alpha+i} = res'_i, \quad \text{for } i = 1, 2, \ldots, \omega$
  $\beta \leftarrow \beta + win_{size}$
  $\alpha \leftarrow \alpha + \chi$

finding a possible schedule, their values are copied to the main $used$ and res vectors in their respective intervals (Lines 33 to 34). This approach temporarily stores scheduling information during each window, ensuring that the main vectors remain unaltered until the schedule is validated. In addition, during the execution of the algorithm, we need to change some parameters of the tasks. For this purpose, we define the set of temporary tasks τ^d, which is initially equal to the set of main tasks of the system at the beginning of the algorithm.

In the first scheduling window, where $\alpha = 0$ and $\beta = win_{size}$, the $used'$ and res' vectors correspond to the elements from 0 to win_{size} in the $used$ and res vectors. Once a possible schedule is found, the values of $used'$ and res' are copied to the main $used$ and res vectors in the range from 0 to win_{size}.

To determine the endpoint of the next scheduling window, we increment the variable β by the window size. However, to determine the starting point α for the next window, we need to take into account the deadlines of the last job for each system task in the current window. Since some jobs may not finish executing in the current window, we need to determine the earliest deadline of last jobs of all tasks within the current window. This is achieved by introducing a parameter called χ, which is calculated as:

$$\chi = \min_{\tau_i \in \tau}(\beta - ((\omega - \Phi_i^d) \bmod D_i^d) \tag{16}$$

where Φ_i^d is the release time of the first job of task $\tau_i^d \in \tau^d$. Thus, the value of α for the next window is computed as the sum of the current value of α and χ.

After calculating the start time of the next window, we need to update the release time of the first job or phase of each task for the next window so that the tasks are released and scheduled at the exact time. The phase will be equal to the difference between χ and the deadline of the last job of each task, i.e., $\Phi_i^d = (\beta - ((\omega - \Phi_i^d) \bmod D_i^d)) - \chi)$ for $\tau_i^d \in \tau^d$.

Regarding the battery level available at the beginning of each scheduling window, in the first window the initial battery level is equal to IE, the system's starting battery level. For the next window, the initial amount of energy in the battery can be determined using Algorithm 2 at the χ step. A portion of this energy is reserved in the battery, and its value is equal to $\sum_{0 \leq r < s} res_r$. In the following window, this amount should be added to res'_0.

To determine the scheduleability of tasks within each scheduling window, we first utilize the PALAP algorithm with ca_0. If a feasible schedule is found, we schedule system tasks based on the provided PALAP schedule. If a feasible schedule is not found, we then attempt scheduleability using the PASAP algorithm. If neither algorithm yields a feasible schedule, we proceed to try the same procedure with the next coreAssignment, incrementing the number of cores until a feasible schedule is reached. In the event that no feasible scheduling is found with any of the core assignments or with neither the PALAP or PASAP methods, the scheduling process is deemed unsuccessful.

Maintaining static energy management is a crucial aspect of the scheduling process. In order to execute tasks and meet their deadlines, we must keep a minimum number of cores turned on, which cannot be turned off during execution. To manage static

energy consumption, a set of virtual tasks is defined. The virtual task set is designed specifically for managing the static energy supply of cores and does not require any computational output, total execution time, or critical path length. Each virtual task is represented as:

$$\tau_i^s = (\Phi_i^s, p_i^s, D_i^s) \tag{17}$$

where:

- Φ_i^s indicates the start time of energy supply for the virtual task τ_i.
- D_i^s indicates the time when the energy supply for the virtual task τ_i ends.
- p_i^s is the power consumption or energy consumption rate of the virtual task τ_i^s between Φ_i^s and D_i^s.

The virtual tasks signify that some cores will be operational between Φ_i^s and D_i^s. There is no slack available for these virtual tasks to receive energy. The execution time of these tasks is considered as $D_i^s - \Phi_i^s$, (10) always returns p_i^s for these tasks. Similar to other system tasks, virtual tasks have priority. The virtual task τ_0^s has the highest priority and is responsible for providing static energy to the minimum number of cores required to meet the deadlines of the system's main tasks.

If during the scheduleability check with a core assignment in a window, it is determined that any of the tasks require more cores than the minimum, a new virtual task is generated for static energy of the extra core. This virtual task has a release time in the scheduling window equal to the release time of the first job of the corresponding task and a deadline equal to the deadline of the last job of that task. Its energy consumption rate is set to $p^s = (ca_{j,i} - ca_{0,i}) * P_{static}$, and there is no slack to supply energy for this virtual task.

8 Time Complexity and Overheads of the Proposed Method

In this section, we analyze the time complexity of the proposed method. HEARTS algorithm comprises two primary elements: a loop for determining the appropriate number of active cores and, within this loop, the invocation of the PALAP and PASAP algorithms. The number of cores is constrained by the number of rows in matrix 1, which reflects the system's task and core count (both available and additional). Considering that the target systems typically feature a small number of tasks with diverse input data, and the extra cores are only a few percent beyond the minimum needed, the row count of this matrix remains manageable. Furthermore, the static energy consumption of the cores limits the feasible number of additional cores that can be feasibly used. Even with a larger number of matrix rows, we can maintain manageability by applying a threshold to restrict the number of rows used.

The computational complexity of the PALAP and PASAP algorithms depends on three main factors: the total number of tasks (n), the maximum worst-case execution time of each task ($W_{i,m}^{max}$) across m cores, and the duration of the scheduling horizon

(H). The overall complexity for these scheduling algorithms is represented by $O(n \times max(W_{i,m}^{max} \times H^2))$.

Additionally, HEARTS divides the scheduling horizon into distinct windows, invoking PALAP and PASAP. The size of these windows (win_{size}) is determined by the longest task deadline, resulting in a computational complexity of $O(n \times (win_{size})^3)$. This level of complexity is manageable due to the small number of tasks and the modest size of the windows. Additionally, by modifying the granularity of the time steps, we can further optimize and reduce the computational complexity.

A key advantage of our method is its flexibility in not requiring operation across an entire hyperperiod. Running algorithms over a complete hyperperiod can be highly inefficient and impractical, particularly due to the exponential growth in hyperperiod size and the challenges in predicting energy availability over extended periods. Instead, our scheduler utilizes smaller, adjustable windows that operate independently of a hyperperiod length.

To further assess the runtime overhead introduced by HEARTS, it is essential to examine the frequency and nature of its execution during system operation. Since HEARTS is invoked at each time window, several factors help mitigate the runtime overhead. The window size is adjustable, ensuring it remains manageable and allowing HEARTS to make scheduling decisions over intervals that are computationally feasible without overwhelming the system. Additionally, the algorithm's efficient window-based approach ensures that computational costs are distributed over time rather than concentrated at the start, as would be the case with a potentially exponential hyperperiod-based scheduler. HEARTS also dynamically manages core allocations, ensuring that only the necessary number of cores is active during each window. The relatively small number of tasks and cores in typical embedded systems further limits the overall computational burden. The complexity of PALAP and PASAP within each window remains bounded at $O(n \times (win_{size})^3)$, keeping the overhead manageable, even with frequent scheduler invocations. Moreover, HEARTS allows for dynamic assessments of the delay and energy cost involved in turning cores on and off. With minimal adjustments to the code, HEARTS efficiently calculates whether activating or deactivating additional cores is cost-effective, taking into account both the computational delay and the extra energy consumption required for core transitions.

Moreover, if necessary, the scheduler's execution and energy consumption can also be considered a high-priority periodic task scheduled within the time windows. This approach allows the time and energy overhead of our proposed scheduler to be accounted for as a task, managed by one of the two schemes mentioned at Section 3.5. By integrating the scheduler's execution within the overall system scheduling, HEARTS effectively manages its own time and energy overhead, ensuring its applicability and efficiency in real-world embedded systems without incurring prohibitive overhead.

9 Optimality of the proposed method

In this section we discuss about the optimality of the proposed method described in Section 7. Our proposed method utilized two energy schedulers, i.e., PALAP and

PASAP. In the following we prove that PALAP is optimal with infinite capacity battery. Then, we describe a condition when PASAP outperforms PALAP.

9.1 Optimality of PALAP

The battery capacity plays a significant role in the success of the scheduling process. In this section, we prove that PALAP algorithm is optimal with a battery with unlimited capacity. Then we will introduce the minimum battery capacity with which the PALAP is optimal.

Theorem 6. *When a battery of infinite capacity is used, the PALAP algorithm is optimal.*

Proof. When the battery has an unlimited capacity, Algorithm 2 for calculating the available battery energy and also (15) are wholly changed, resulting the following formula for calculating available energy in Step s:

$$En(s) = IE + \sum_{0 \leq r < s} (P_R(r) - used_r - res_r) \tag{18}$$

Suppose with energy level P_r And a battery with an infinite capacity, Algorithm A^* can schedule the set of tasks τ, but the $PALAP$ algorithm cannot. In this case, there is an execution step λ, $0 \leq \lambda < W_{i,m}^{max}$ of the task $\tau_i \in \tau$, in which the $PALAP$ cannot provide as much energy as $\phi_i^m(\lambda)$ for λ, but A^* has provided that amount of energy for λ and all other jobs in τ. Let's assume the deadline to supply energy to λ is step s, such that $(s \mod D_i) = D_i - W_{i,m}^{max} + \lambda$, where algorithm A^* has provided the energy to λ in Step s', and $s' \leq s$. We examine two general cases: in the first, the energy used up to Step s is less, which means $\sum_{0 \leq r < s} used_r^{A^*} < \sum_{0 \leq r < s} used_r^{PALAP}$ where $used_r^{A^*}$ and $used_r^{PALAP}$ are the $used$ vectors of A^* and $PALAP$ respectively. In the second case, the total energy reserved for the next steps in Step s is less, which means $\sum_{0 \leq r < s} res_r^{A^*} < \sum_{0 \leq r < s} res_r^{PALAP}$ where $res_r^{A^*}$ and res_r^{PALAP} are the res vectors of A^* and $PALAP$ respectively. These two cases will be examined in the following.

For the first case, higher-priority tasks must be executed in a way that the value of $\sum_{0 \leq r < s} used_r^{A^*}$ be less than the corresponding value in $PALAP$. Since jobs with deadlines less than s cannot be moved, jobs with deadlines greater than s must be executed in a way that leaves more energy for task τ_i. This can be accomplished by delaying higher-priority tasks as much as possible. In $PALAP$ each task τ_i have a delay of $D_i - W_{i,m}^{max}$, and if the A^* algorithm applies more delay, τ_i will miss its deadline. Therefore, A^* cannot have a lower sum of $used$ up to Step s than $PALAP$.

Suppose that the total energy reserved for the steps after s in the A^* algorithm is less than $PALAP$. In order to decrease the value of $\sum_{0 \leq r < s} res_r^{A^*}$ in Step s, the energy reservation for higher priority tasks should be moved after Step s (if it is moved before Step s, the result will remain unchanged). Suppose in Step s we have some energy Δ of the total energy $\sum_{0 \leq r < s} res_r^{A^*}$ reserved for task $\tau_{reserve}$ to be consumed in Step δ. Obviously, all the energy between s and δ is consumed. Therefore, there is no possibility of reserving Δ units of energy from another step between s and δ. Suppose algorithm A^* executes one of the tasks with higher priority than $\tau_{reserve}$

earlier than $PALAP$, such as in Step ν, so that a portion of the energy reserve Δ is transferred after Step s. If $s > \nu$, $\sum_{0 \leq r < s} res_r^{A^*}$ will remain unchanged, and if $s < \nu$, $\sum_{0 \leq r < s} used_r^{A^*}$ will be added exactly the same amount of energy as it is subtracted from $\sum_{0 \leq r < s} res_r^{A^*}$, and the final result will not change.

□

Since the delay of the PALAP algorithm for each task τ_i is equal to $D_i - W_{i,m}^{max}$, the energy required for each task must be stored within this delay time. The worst-case scenario occurs when all tasks are delayed simultaneously. Therefore, the PALAP algorithm is optimal if the battery capacity is greater than B_{PALAP}, where:

$$B_{PALAP} = \sum_{\tau_i \in \tau} \sum_{s=1}^{D_i - W_{i,m}^{max}} \varphi_i^m(s) \tag{19}$$

9.2 Scenarios in which PASAP outperforms PALAP

The previous section demonstrated that the PALAP algorithm is optimal when utilized with a battery of specific capacity as determined by (19). However, there may be situations where the available battery capacity is insufficient for the PALAP algorithm. In such cases, the PASAP algorithm is preferred over the PALAP algorithm as it can schedule the task set that the PALAP algorithm cannot. An example of such conditions is presented in the following.

Example 2. *Consider a system with two tasks, τ_1 and τ_2, where $D_1 = 4$ and $D_2 = 2$. Both tasks require the same minimum number of cores to meet their respective deadlines. Let's assume that the energy consumption rate of τ_1 during its three execution steps is x, while that of τ_2 during its single execution step is y, where $x > y$. We further assume that the energy replenishment rate is $P_R = x$, which is equal to the energy consumption rate of τ_1. In this scenario, both tasks have one slack step, and the execution time of τ_1 and τ_2 with the minimum core are $D_1 - 1$ and $D_2 - 1$, respectively.*

Suppose we have a B-capacity battery, where $y \leq B < 2y$, and initially, it is full. D_1 is the hyper-period in this case. Let's examine how these two algorithms perform scheduling:

- *In the PALAP algorithm, τ_1 is not executed in the first step due to having a one-step slack. In the subsequent steps, it can be executed without reserving energy up to D_0 because $P_R \geq x$. Similarly, τ_2 is also not executed in the first step due to a one-step slack. The delay in executing task τ_2 in the first step will be the same as the delay in executing task τ_1 at the first step. As there is no empty capacity and the battery is full, P_R units of energy will be lost. τ_2 will be executed in the second step because $P_R + B - x \geq y$. In the third step, the second job of τ_2 will be released. However, this job cannot be executed in the third step or the fourth step because $2P_R + B - 2x - y < y$ in the third step, and $3P_R + B - 3x - y < y$ in the fourth step.*
- *In the PASAP algorithm, τ_1 can be executed from Step 1 to Step 3 without the need for reserving energy and will have a slack in the fourth step. The first job of τ_2 is executed in the first step because $P_R + B - x > y$, and this job meets its deadline. In*

the third step, the second job of τ_1 is released. However, it is not possible to execute this job in the third step because $3P_R + B - 3x - y < y$. But, in the fourth step, this job can be executed because $4P_R + B - 3x - y > y$, and it meets its deadline.

10 Evaluation

In this section, we assess the performance of our proposed algorithms. Initially, in Subsection 10.1, we outline two comparing methods to evaluate our algorithm against. Our analysis is structured into three main categories. In Subsection 10.2, we aim to conduct a comprehensive evaluation of our algorithms across a diverse array of scenarios. We utilize a standardized approach [65][66] for generating synthetic task parameters, allowing us to compare our scheduling outcomes against the comparing methods described in Subsection 10.1. In Subsection 10.3, we delve into the specifics of scheduling performance under conditions of critical battery capacity. Lastly, in Section 10.4, we examine the schedulability of select digital signal processing applications from the STR2RTS benchmark suite [67], leveraging real energy harvesting data sourced from the Solar Radiation Lab (SRL) [68]. The findings from this investigation are then contrasted with the comparing methods introduced in Subsection 10.1.

10.1 Comparing Approaches

The HEARTS algorithm incorporates PALAP , which we've proven to be optimal for battery capacities larger than B_{PALAP}. However, it's not optimal for battery capacities smaller than this threshold. While PASAP can schedule tasks in some situations where PALAP cannot, HEARTS cannot be considered optimal when the battery is smaller than B_{PALAP}. To assess the performance of HEARTS under non-optimal conditions, we must compare it to alternative methods.

To the best of our knowledge, our work is the first to tackle the challenge of harvested energy-aware scheduling of parallel tasks on multicore systems with identical cores operating at a fixed frequency. Previous studies in energy-aware scheduling, such as the work by Chetto et al. [35], have predominantly focused on single-processor systems executing single-threaded tasks, without considering the complexities introduced by parallel task models or multicore architectures. Similarly, while Guo et al. [69] and Abusayeed et al. [5] have explored energy-efficient scheduling of parallel tasks on multicore systems, their objectives center on minimizing total energy consumption rather than addressing the constraints of harvested energy. Furthermore, recent efforts like [70] have introduced the DAG task model for energy-aware scheduling but are confined to uniprocessor systems. Due to these fundamental differences in system models and objectives, we cannot directly compare our work with existing studies, as they do not address the specific challenges of scheduling parallel tasks within their deadlines using only environmentally harvested energy on multicore platforms.

Given the distinctive nature of our approach to harvesting energy-aware scheduling for parallel tasks, we conduct comparative analyses with two alternative strategies, acknowledging the absence of directly comparable existing work.

Partitioned Energy And Battery (PEAB): We explore a method wherein harvested energy and battery capacity are distinctly partitioned between tasks based on

their utilization, treating each task as an independent entity with its dedicated energy harvesting and battery system. In this model, tasks are restricted to operating on their minimum required cores without being assigned additional cores. This paradigm closely aligns with the concepts discussed in [71] and [72]. Additionally, PEAB simulates the well-known PFP_{ASAP} [38] method when applied to a system with only one task, acting as an energy-aware scheduling model for single-core systems.

Hypothetical Optimal Algorithm (HOA): Given the complexity and computational impracticality of implementing a truly optimal algorithm, which would require exploring an exponential state space, we assumed the existence of a hypothetical algorithm. This algorithm is capable of scheduling tasks within a window only if the energy harvested during that window exceeds the minimum energy requirement. This requirement encompasses the energy demands of tasks within the window and the static power needed for the minimum number of active cores according to the scheduling policy—either Federated or Global—during that period. Additionally, we assume the algorithm operates with the battery fully charged at the beginning of each scheduling window. Naturally, if these conditions aren't met, task scheduling becomes unfeasible. Furthermore, if another optimal algorithm were to exist, it could, at best, perform at the same level as our hypothetical optimal algorithm or, more likely, perform even worse.

We introduce two variants of this hypothetical optimal algorithm: HOA-F and HOA-G. Both are simulated under pessimistic assumptions to approximate the boundary of optimal scheduling performance under different core management strategies.

- **HOA-F (Federated Scheduling)**: This variant assumes total static power consumption based on the minimum number of cores required for Federated scheduling. The static energy for this model is calculated as $P_{\text{static}} \times \text{minCores} \times (\beta - \alpha)$, reflecting the power usage necessary to maintain task operations throughout the scheduling window.
- **HOA-G (Global Scheduling)**: Unlike HOA-F, HOA-G calculates static power based on the minimum number of cores required for a Global Scheduling policy. This is done by summing the utilization of all tasks, where each task's utilization u_i is defined in Equation **??**. Consequently, the static energy for HOA-G is computed as $P_{\text{static}} \times \sum_{\tau_i \in \tau} u_i \times (\beta - \alpha)$.

At the beginning of each scheduling window, defined by α and β as the start and end of the window, respectively, we calculate the total energy harvested during that window $\sum_{j=\alpha}^{\beta} P_R(j)$. If this amount, combined with a fully charged battery B, surpasses the energy demands of tasks for that window $\sum_{\tau_i \in \tau} \left\lceil \frac{(\beta-\alpha)}{D_i} \right\rceil C_i p_i$ and the total static energy required for the core count during that window, then we consider the hypothetical optimal algorithm, whether HOA-F or HOA-G, capable of scheduling. The specific conditions for each are as follows:

For HOA-F:

$$\sum_{j=\alpha}^{\beta} P_R(j) + B \geq \sum_{\tau_i \in \tau} \left\lceil \frac{(\beta - \alpha)}{D_i} \right\rceil C_i p_i + P_{\text{static}} \times \text{minCores} \times (\beta - \alpha)$$

For HOA-G:

$$\sum_{j=\alpha}^{\beta} P_R(j) + B \geq \sum_{\tau_i \in \tau} \left\lceil \frac{(\beta - \alpha)}{D_i} \right\rceil C_i p_i + P_{\text{static}} \times \sum_{\tau_i \in \tau} u_i \times (\beta - \alpha)$$

These formulas encapsulate the operational logic behind each hypothetical optimal algorithm, and we have assumed that these two algorithms will be able to schedule only if the necessary energy is available. HOA simulates the boundary conditions for optimal performance by determining the minimum energy required for task scheduling. This serves as a reference point for measuring deviations in real-world conditions. A detailed proof of the HOA's optimality can be found in Appendix C.

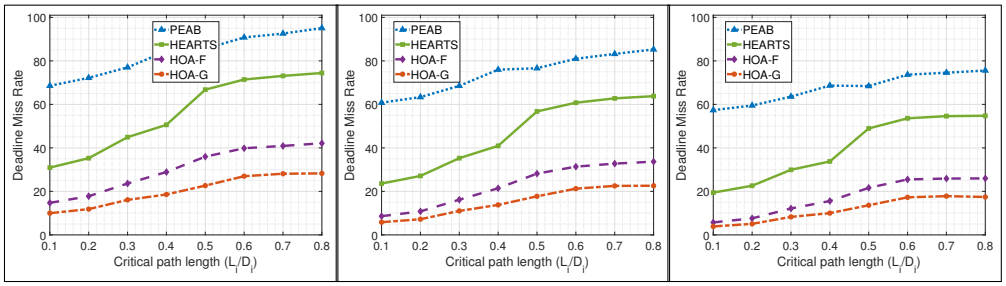
10.2 Comprehensive evaluation with synthetic task sets

In this simulations, we aimed to comprehensively evaluate our proposed algorithms across a wide range of scenarios that closely resemble diverse real-world applications. To achieve this, we employed the UUnifast algorithm [65] and the hyperperiod limitation technique [66] to generate random tasks, which are widely recognized methods in testing scheduling algorithms [73][57][74][75][76][77]. This approach enables systematic variation in task characteristics, ensuring a robust assessment of our scheduling algorithms under diverse conditions.

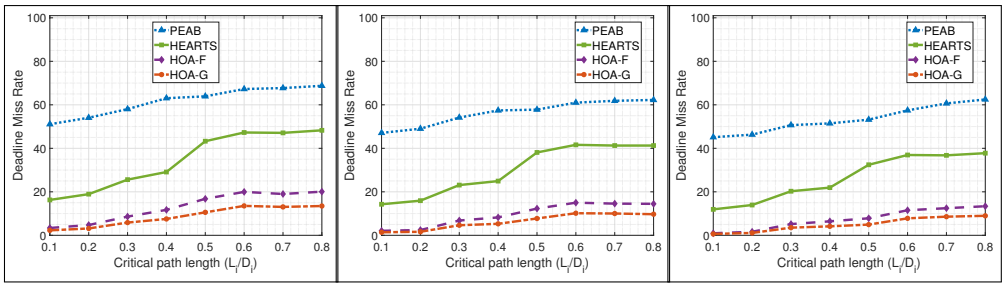
We follow a two-step process for each task set, with specific utilization requirements and a hyperperiod length (H). Firstly, we generate task periods $(\pi_1, \pi_2, ..., \pi_5)$ using the hyperperiod limitation technique [66]. These deadlines are constrained within the range of 1 to H, where n represents the number of tasks in the set. Secondly, we employ the UUnifast algorithm [65] to generate utilization values $(U_1, U_2, ..., U_n)$ for each task. Initially, this algorithm generates utilization values within the range of 0 to 1. To achieve high utilization tasks, we scale each utilization value in the range of [1, 4].

Following this, we calculate the total execution time (C_i) for each task τ_i (where $1 \leq i \leq n$) by multiplying the respective utilization (U_i) and deadline (D_i). Different values for L_i were considered in the experiments, with increments of 10, ranging from 10% to 80% of D_i. The power consumption for each task, i.e., p_i, was randomly chosen between 10 to 100. The phase, i.e., ϕ_i, for all tasks set to 0. All experiments were iterated over different static energy values, i.e., P_{static} for the cores, increasing in increments of 5, covering a range from 5% to 35% of the average dynamic energy of the task set ($\frac{1}{n}\sum_{i=1}^{n} p_i$). We generate 3000 task sets, comprising 1000 sets with $n = 3$, 1000 sets with $n = 5$, and 1000 sets with $n = 7$. The hyperperiod limits are set to 600, 1500, and 3500, respectively.

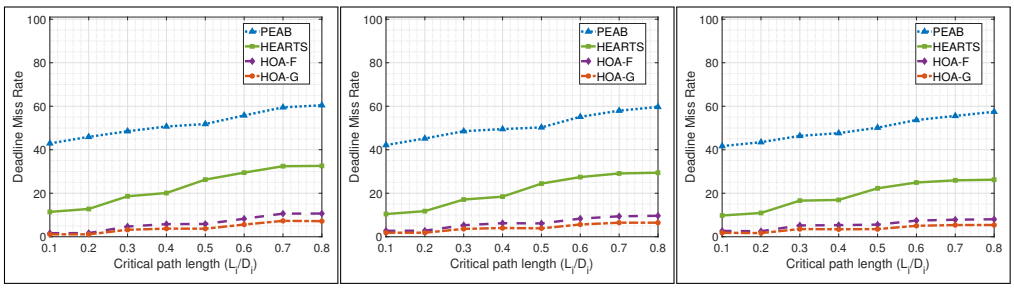
We used random values to generated harvested energy rates. initially, we generated 100 basic rates with random values selected from the range ρ to $100 + \rho$, where $\rho = 0.05\frac{1}{n}\sum_{i=1}^{n} p_i$ for each step. This process covers the entire duration of one hyperperiod for each task set. It is important to note that in our experiments, the static power consumption is at least 5% of $\frac{1}{n}\sum_{i=1}^{n} p_i$, and the dynamic power consumption of each task is at most 100. Next, we adopted the 100 distinct random energy rates for each task set by multiplying each step by the $minCores$ of the corresponding task set.



(a) $B = 10\%$ of B_{PALAP}. (b) $B = 20\%$ of B_{PALAP}. (c) $B = 30\%$ of B_{PALAP}.



(d) $B = 40\%$ of B_{PALAP}. (e) $B = 50\%$ of B_{PALAP}. (f) $B = 60\%$ of B_{PALAP}.



(g) $B = 70\%$ of B_{PALAP}. (h) $B = 80\%$ of B_{PALAP}. (i) $B = 90\%$ of B_{PALAP}.

Fig. 5: Effectiveness of HEARTS in task schedulability: comparing deadline miss rates under various battery capacities and critical path lengths to PEAB, HOA-F and HOA-G methods.

In each experiment, the number of system cores, i.e., N, was set to 130% of the minimum required for each task set. The window size, i.e., win_{size} in all experiments is set to the maximum deadline of the tasks.

We evaluated the effectiveness of HEARTS's task scheduling under non-optimal battery capacity conditions. This involved assessing the schedulability of all 3000 tasks across every scheduling window within a single hyperperiod for all 100 harvesting

rates, with battery capacities ranging from 10% to 100% of B_{PALAP}, in increments of 10%. The number of missed deadlines was recorded.

These experiments were conducted with static energy rates varying from 5% to 35%, in increments of 5%. Similar tests were carried out for critical path lengths ranging from 10% to 80%, in increments of 10%, and for different battery capacities from 10% to 100% of B_{PALAP}.

Finally, we compared the results obtained from our method with those of the PEAB, HOA-F and HOA-G methods. Figure 5 presents a comparative analysis between our proposed method PEAB, HOA-F and HOA-G.

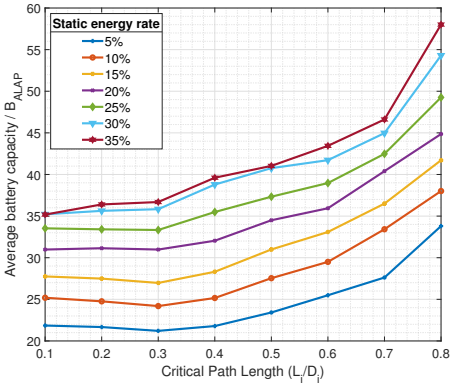
The results indicate that, across various battery capacity settings, HEARTS exhibits an average deviation of 19.05% and 21.34% from HOA-F and HOA-G, respectively, and performs 28.24% better on average than the PEAB method. Additionally, for critical path lengths shorter than 0.4 of D_i (indicating more parallel tasks), HEARTS shows an average deviation of 13.74% and 15.14% from HOA-F and HOA-G, respectively, and outperforms the PEAB method by 32.95%. These findings demonstrate that HEARTS performs better with an increased number of parallel tasks.

10.3 Schedulability under critical battery capacity

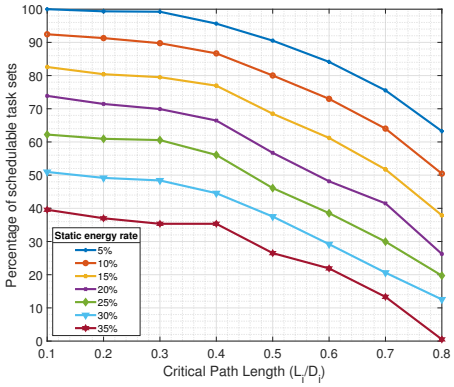
As mentioned earlier, the feasibility of our proposed scheduling algorithms depends directly on having sufficient battery capacity. We showed that the PALAP algorithm is optimal when the battery capacity is more than B_{PALAP}. We experimentally determined the minimum battery capacity required to successfully schedule tasks by performing a binary search over capacity for each combination of task set (3000 total) and energy harvesting rates (100 total). This identified the minimum capacity needed to schedule all tasks using HEARTS. In Figure 6a we reported the average amount of this empirically identified critical battery capacity averaged over all 3000 task sets and 100 harvesting rates (as described in Subsection 10.2) as a ratio of B_{PALAP} (see (19)) under different values of L_i and P_{static}. Having established this critical battery capacity threshold, we then conducted experiments with capacities constrained to this minimum.

During our testing, we also assessed the percentage of tasks that could not be scheduled with any battery capacity and reported the average percentage of schedulable tasks while varying the static power values of the cores and critical path lengths. Figure 6b displays the average task schedulability percentage with respect to the static energy rates of cores and the size of the critical path. As shown in this figure, lower static power percentages for the cores and smaller critical path values (indicating increased task parallelism) result in higher average schedulability percentages.

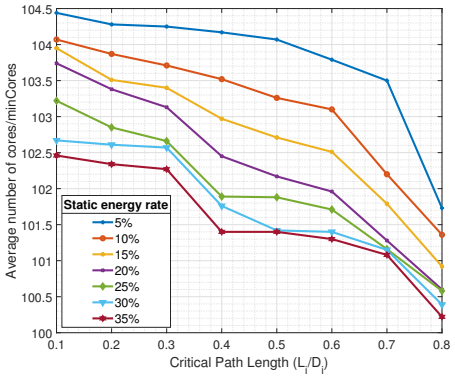
HEARTS may employ different core numbers for tasks within each window and select either PALAP or PASAP for each scheduling window. Figure 6c presents the average number of cores used across scheduling windows based on the minimum number of cores requirements ratio for each task set. This figure highlights that when core static power values are smaller and L_i is shorter, HEARTS deploys more cores to compensate for fluctuations in harvested energy, reducing the need for a larger battery. This observation aligns with the comparison between Figures 6a and 6c.



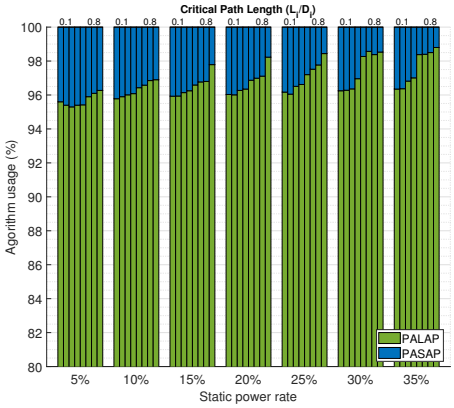
(a) Average battery capacity required to schedule tasks with respect to B_{PALAP}.



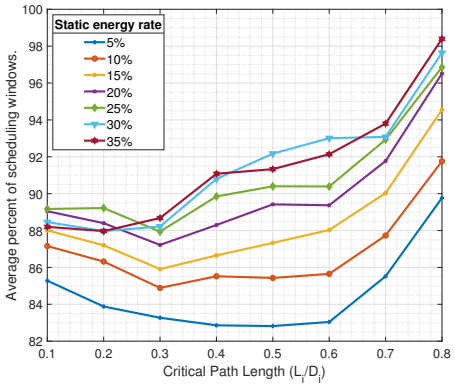
(b) Percentage of schedulable task sets.



(c) The average number of cores utilized with respect to minimum number of cores.



(d) PALAP / PASAP usage (%) by the HEARTS.



(e) Average percent of scheduling windows which HEARTS utilized the minimum number of cores.

Fig. 6: Evaluation of proposed method under critical battery capacity.

Table 2: Benchmark applications

Application	$C_i(Cycles)$	$L_i(Cycles)$	$C_i(Steps)$	$L_i(Steps)$
FIRBank	9,298,734	707,462	93	8
FFT2	7,641,493	3,836,011	77	39
MatrixMult	1,700,645	983,954	17	10
Filterbank	893,252	172,244	9	2
BeamFormer	697,450	129,725	7	2

Additionally, Figure 6d illustrates the average usage of PALAP and PASAP across the scheduling windows throughout the entire scheduling horizon. In most cases, PALAP is capable of scheduling task sets, with PASAP used sparingly. Figure 6e reveals the average number of windows in which HEARTS relied on the minimum core count for scheduling. Comparing this figure with Figure 6a underscores that smaller static power values for cores and shorter critical path lengths prompt HEARTS to increase core numbers and expedite certain tasks to mitigate energy fluctuations, ultimately resulting in reduced battery requirements.

10.4 Schedulability with real energy harvesting on benchmark applications

In this subsection, we evaluate our methodologies using a selection of digital signal processing applications from the STR2RTS benchmarks [67], alongside actual energy harvesting data from the Solar Radiation Lab (SRL) [68]. The STR2RTS benchmarks detail the DAG structures and the worst-case execution time (WCET) for nodes within these structures. We focus on five applications: FIRBank (PEG127-nocache), FFT2, MatrixMult(PEG143-nocache), DCT (PEG63-IEEE-nocache), and BeamFormer (12c-4b), using their C_i and L_i values based on processor cycles. To match the tasks' temporal granularity, we defined each step as 0.1 milliseconds, assuming tasks run on a 1 GHz processor. This setup translates the C_i and L_i cycle values into step counts for the fourth and fifth columns of our data table. Task periods $(\pi_1, \pi_2, ..., \pi_5)$ were established using the hyperperiod limitation technique [66] with a hyperperiod limit of 900000. The UUnifast algorithm [65] generated utilization values $(U_1, U_2, ..., U_5)$ within a [1, 4] range for each task. Our computational model is based on the ARM Cortex-A15 processor with ARMv7 architecture. Following [78], we set each core's static power consumption (P_{static}) at 0.11W. Dynamic power consumption (p_i) for each task was assigned randomly within a [0.22-1]W range.

We utilized a solar energy dataset from SRL [68], which covers 12 days, to simulate energy harvesting, specifically focusing on the period from 9:00 AM to 5:00 PM. This dataset, sourced from the SRL, contains solar energy data for the 15th day of each month in 2022, measured from a 100 cm x 100 cm solar panel. We adjusted these harvesting rates according to the $minCores$ requirement for each task set.

We conducted 1,000 trials using five specified tasks and scheduled them using harvested energy. Simulations were performed across battery capacities ranging from 10% to 90% of B_{PALAP}, with our methodologies compared against the PEAB, HOA-F and HOA-G methods. The outcomes are illustrated in Figure 7. The results indicate that, across various battery capacity settings, the proposed method exhibits an average

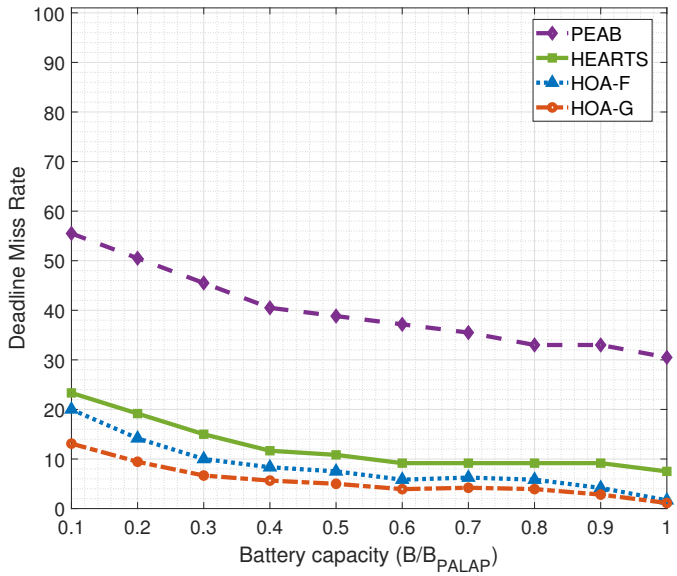


Fig. 7: Deadline miss ratio of benchmark tasks with SRL energy harvesting rates for different battery capacities (B/B_{PALAP}) compared to PEAB, HOA-F and HOA-G methods.

deviation of 4.04% and 5.81% from HOA-F and HOA-G, respectively, and performs an average of 27.53% better than the PEAB method.

10.5 Analyzing Evaluation Results

When compared to the PEAB and HOA baselines (HOA-F and HOA-G), HEARTS demonstrates substantial advantages, particularly in high-parallelism scenarios. Unlike PEAB's static partitioning of energy resources, HEARTS employs a dynamic and adaptive energy scheduling approach, allowing it to align core usage more effectively with real-time energy availability and task demands. Notably, for parallel tasks with shorter critical path lengths, HEARTS consistently aligns more closely with HOA-F and HOA-G, while significantly surpassing PEAB's performance. HEARTS predominantly defaults to PALAP, as shown in Figure 6d. Section 9 establishes PALAP as the optimal scheduling approach under certain conditions, which explains why PALAP is the preferred strategy in most scheduling windows, as evidenced in Figure 6d. However, certain energy conditions necessitate switching to PASAP for task scheduling, validating the theoretical analyses presented in Section 9.

HEARTS' core allocation and energy management further highlight its adaptability. Figures 6c and 6a illustrate that HEARTS dynamically increases core usage within scheduling windows, especially when core static power is low and task critical path lengths L_i are shorter. By increasing core usage under these conditions, HEARTS compensates for delays in energy supply, accelerating task execution and

reducing dependency on the need for larger battery capacities. This adaptability is further emphasized in Figure 6e, which shows HEARTS' dynamic response to real-time energy variations. When core static power values are low and critical paths are shorter (indicating high-parallelism conditions), HEARTS increases core counts or expedites specific tasks to prevent delays due to energy shortages. This flexibility minimizes the impact of energy fluctuations on task execution, ensuring operational stability while reducing battery requirements. Figures 6a and 6b further emphasize that HEARTS requires a lower critical battery capacity to achieve high schedulability, especially in high-parallelism settings.

In summary, with its dual energy-scheduling strategies—PALAP and PASAP—HEARTS is a highly effective method for task scheduling in environments with energy fluctuations, particularly when tasks exhibit high parallelism.

11 Related work

In recent years, extensive research conducted on the energy management of CPSs. Related work in this area can be divide into two general categories of *energy-aware* and *energy-efficient* researches. The objective in the area of the energy-aware system is to comply with time constraints and deadlines according to amount of harvested energy. However, in the field of energy-efficient systems, the goal is to minimize total energy consumption. The primary purpose of the current research is to provide an energy-aware schedule of a cyber-physical system with parallel tasks running on a multiprocess platform.

In the following, we will first review the essential work done in energy-aware systems, and then, considering the importance of work done in the field of energy efficiency, several important cases that have used the parallel task model are also considered.

11.1 Harvested energy-aware scheduling

In one of the earliest studies on energy-aware scheduling, [79] proposed a simple method for scheduling tasks in a frame-based system with maximum and minimum energy constraints. Chetto et al. [35] proposed an algorithm called Earliest Deadline with Energy Guarantee (EDeg). This algorithm executes tasks according to the earliest deadline first while ensuring that energy constraints are not violated. Upon detecting future energy failures, the system is suspended as long as possible according to timing constraints or until the energy storage unit is full. They, therefore, define the concepts of slack energy and slack time, which are used to determine the execution and replenishment times, respectively. Several other heuristics were examined in [36]. The Lazy Scheduling Algorithm (LSA) [37] and the Preemptive Fixed Priority as-soon-as-possible (PFP_{ASAP}) [38] are two energy-aware fixed priority scheduling algorithms. In the former, task's energy consumption is assumed to depend on its worst-case execution time. However, this hypothesis is not always true for embedded systems [80]. In the later, when there is sufficient energy to execute one time unit of the highest priority active task, PFP_{ASAP} schedules jobs; otherwise, task execution is suspended to replenish the energy supply. There is sufficient schedulability testing for PFP_{ASAP}

when the rate of harvested energy is fixed over time [59]. The case of non-optimal energy storage in [81] is investigated, and a scheduling algorithm based on PFP_{ASAP} is presented.

Moreover, there have been recent works that have explored the scheduling of energy-aware real-time tasks. For instance, in [75], the authors discuss an energy-harvesting real-time system with periodic tasks of varying performance levels. In a different context, [82] introduced CyEnSe, a scheduling challenge that deals with cyclic energy patterns and varying energy arrivals. The authors provide two solutions, a heuristic approach, and a linear programming method. Furthermore, [83] and [84] delve into the topic of mixed-criticality scheduling for energy-harvesting Systems.

The abovementioned studies primarily focused on single-processor systems employing a single-threaded task model. However, there is a recent work that introduces the use of the DAG task model [70], particularly for addressing hard real-time scheduling of DAG tasks. These tasks consist of dependent sub-tasks under precedence constraints and are managed within an energy harvesting system. Importantly, this study is confined to uniprocessor systems and does not encompass scheduling in multi-core systems.

11.2 Energy-efficient scheduling

There have been numerous studies that deal with energy in multicore systems where there is no input energy limit, and their goal is to optimize overall energy consumption. These works may employ a variety of techniques, such as turning off rarely used cores [85], reducing dynamic and leakage power consumption with Dynamic Voltage and Frequency Scaling (DVFS) and Dynamic Power Management (DPM)[86][87][88], managing power on heterogeneous systems with DVFS [89][90][91][92][93], or dividing cores into blocks with DVFS-enabled power and applying the DVFS to each block separately [94][50].

In most of these studies, single-threaded tasks are used. Similar techniques are used in some recent studies that use parallel tasks to optimize energy consumption. For instance, in [69], federated scheduling of DAGs was considered to minimize the platform's overall power consumption. Another study in [95] investigated real-time scheduling of sporadic DAG tasks on cluster-based multicores to minimize the overall power consumption of the system. Abusayeed et al. [5] address hard real-time scheduling of parallel DAG tasks while minimizing CPU energy consumption on multicore embedded systems. Their technique is to determine the execution speeds of the nodes of the DAGs to minimize the overall energy consumption.

Our research distinguishes itself from the existing literature discussed in this section by focusing primarily on the scheduling of parallel tasks within their deadlines, leveraging energy harvested from the environment. Unlike other works that might deal with energy-aware scheduling in a broader sense, our approach is specifically tailored to systems equipped with identical cores operating at a fixed frequency. The core objective of this paper is to explore and develop methodologies for harvested energy-aware scheduling, ensuring that all parallel tasks are executed within their respective deadlines. This emphasis on utilizing **environmentally harvested energy** for task execution sets apart our work from generalized energy management and optimization

strategies, such as those presented in [96], [97], [98], [99], [100], [101], [102], [103], [104], [105], [106], [107], [108], [109], which do not specifically address the unique challenges and opportunities presented by energy-harvesting systems.

11.3 Other studies

There are some other related works in other fields as well. For example, in [110] Karbasioun et al. proposed a linear time algorithm for supplying electricity to a set of N customers with malleable demands. Each customer requires a certain amount of electrical energy, which has to be supplied during a time interval. Similar works has been done in the field of Smart Grid [111] [112] [113] [114] or Cloud Computing [115][116][117][118]. However, power requests are completely different from parallel tasks. These systems do not have batteries for storing energy or periodic requests, and each demand must be met without interruption at all times.

12 Conclusion

In conclusion, this paper has unveiled an innovative energy scheduling methodology tailored for efficiently managing high-utilization parallel tasks within energy harvesting systems characterized by fluctuating energy supplies. The proposed method demonstrates adaptability and effectiveness by leveraging a runtime scheduler equipped with two distinct energy scheduling algorithms—one adhering to the first-fit strategy and the other to the last-fit strategy. The runtime scheduler's ability to dynamically allocate cores and employ alternative strategies ensures a responsive approach to the demands of task deadlines. The established optimality of the last-fit-based algorithm for batteries surpassing a predefined threshold, coupled with the superior performance of the first-fit-based algorithm for smaller battery capacities, underscores the versatility inherent in our approach. In essence, the proposed energy scheduling method addresses the intricate challenges posed by energy harvesting systems and establishes itself as a reliable and adaptable solution. Its success rates underscore its potential applicability in real-world scenarios, particularly in energy and battery-constrained environments.

Future research will focus on refining the energy model by incorporating more realistic battery dynamics, such as charge-dependent leakage and energy reservation strategies for mission-critical phases. Additionally, we aim to explore how energy-aware scheduling can be further optimized in small battery scenarios by studying transient systems and intermittent energy sources, while also implementing strategies to manage task prioritization during periods of battery depletion. These enhancements will improve the practicality of our model and its applicability across a broader range of real-world systems.

References

[1] Lee, E.A.: Cyber physical systems: Design challenges. In Proceedings of the 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC), 363–369 (2008) https://doi.org/10.

1109/ISORC.2008.25

[2] Marwedel, P.: Embedded system design: embedded systems foundations of cyber-physical systems. In: Proceedings of the 2010 International Conference on Embedded Computer Systems (SAMOS), pp. 1–8 (2010). https://doi.org/10.1007/978-94-007-0257-8

[3] Yi, S., Kim, T.-W., Kim, J.-C., Dutt, N.: Easyr: Energy-efficient adaptive system reconfiguration for dynamic deadlines in autonomous driving on multicore processors. ACM Trans. Embed. Comput. Syst. **22**(3) (2023) https://doi.org/10.1145/3570503

[4] Tessler, C., Modekurthy, P., Fisher, N., Saifullah, A., Murphy, A.: Co-located parallel scheduling of threads to optimize cache sharing. Accepted to appear in IEEE RTSS '23 (The 43rd IEEE Real-Time Systems Symposium), 1–12 (2023)

[5] Saifullah, A., Fahmida, S., Modekurthy, V.P., Fisher, N., Guo, Z.: CPU Energy-Aware Parallel Real-Time Scheduling. In: Völp, M. (ed.) 32nd Euromicro Conference on Real-Time Systems (ECRTS 2020). Leibniz International Proceedings in Informatics (LIPIcs), vol. 165, pp. 2–1226. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2020). https://doi.org/10.4230/LIPIcs.ECRTS.2020.2

[6] Intel Corporation: Intel Documents. https://www.intel.cn/content/dam/www/public/us/en/documents/technology-briefs/intel-labs-single-chip-cloud-architecture-brief.pdf (Accessed on 2024-09-05)

[7] Wikipedia contributors: Tilera TILE-Gx Family of Multicore Processors. Accessed: 2023-10-28

[8] Wikipedia contributors: PicoChip - Wikipedia. Accessed: 2023-10-28

[9] Blumofe, R.D., Joerg, C.F., Kuszmaul, B.C., Leiserson, C.E., Randall, K.H., Zhou, Y.: Cilk: An efficient multithreaded runtime system. SIGPLAN Not. **30**(8), 207–216 (1995) https://doi.org/10.1145/209937.209958

[10] Corporation, I.: Intel Corporation. Intel Cilk Plus Language Extension Specification. https://www.cilkplus.org/sites/default/files/open_specifications/Intel_Cilk_plus_lang_spec_1.2.htm Accessed 2021-07-14

[11] Tardieu, O., Wang, H., Lin, H.: A work-stealing scheduler for x10's task parallelism with suspension. SIGPLAN Not. **47**(8), 267–276 (2012) https://doi.org/10.1145/2370036.2145850

[12] Arrieta Caballero, G.R., Huacho Rojas, G.M., Rodriguez Lengua, P.R.: Red de sensores para monitoreo de radiación no ionizante de los servicios de telefonía móvil en colegios y hospitales del distrito de la punta en la región callao (2019)

[13] Priya, S., Inman, D.J.: Energy Harvesting Technologies vol. 21. Springer, New York, USA (2009). https://doi.org/10.1007/978-0-387-76464-1

[14] Fridley, D.: Nine Challenges of Alternative Energy. Post Carbon Institute, California, USA (2010)

[15] Zhou, J., Yan, J., Wei, T., Chen, M., Hu, X.S.: Energy-adaptive scheduling of imprecise computation tasks for qos optimization in real-time mpsoc systems. In: Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017, pp. 1402–1407 (2017). https://doi.org/10.23919/DATE.2017.7927212 . IEEE

[16] Bonifaci, V., Marchetti-Spaccamela, A., Stiller, S., Wiese, A.: Feasibility analysis in the sporadic dag task model. In: 25th Euromicro Conference on Real-time Systems, pp. 225–233 (2013). https://doi.org/10.1109/ECRTS.2013.32 . IEEE

[17] Li, J., Agrawal, K., Lu, C., Gill, C.: Analysis of global edf for parallel tasks. In: 25th Euromicro Conference on Real-Time Systems, pp. 3–13 (2013). https://doi.org/10.1109/ECRTS.2013.12 . IEEE

[18] Baruah, S., Bonifaci, V., Marchetti-Spaccamela, A., Stougie, L., Wiese, A.: A generalized parallel task model for recurrent real-time processes. In: 2012 IEEE 33rd Real-Time Systems Symposium, pp. 63–72 (2012). https://doi.org/10.1145/3322809 . IEEE

[19] Chwa, H.S., Lee, J., Phan, K.-M., Easwaran, A., Shin, I.: Global edf schedulability analysis for synchronous parallel tasks on multicore platforms. In: 25th Euromicro Conference on Real-Time Systems, pp. 25–34 (2013). https://doi.org/10.1109/ECRTS.2013.14 . IEEE

[20] Liu, C., Anderson, J.H.: Supporting soft real-time parallel applications on multicore processors. In: IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, pp. 114–123 (2012). https://doi.org/10.1109/RTCSA.2012.55 . IEEE

[21] Li, J., Chen, J.J., Agrawal, K., Lu, C., Gill, C., Saifullah, A.: Analysis of federated and global scheduling for parallel real-time tasks. In: 26th Euromicro Conference on Real-Time Systems, pp. 85–96 (2014). https://doi.org/10.1109/ECRTS.2014.23

[22] Baruah, S.: Federated scheduling of sporadic dag task systems. In: IEEE International Parallel and Distributed Processing Symposium, pp. 179–186 (2015). https://doi.org/10.1109/IPDPS.2015.33 . IEEE

[23] Baruah, S.: The federated scheduling of constrained-deadline sporadic dag task systems. In: Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 1323–1328 (2015). https://doi.org/10.7873/DATE.2015.0200 . IEEE

[24] Li, J., Agrawal, K., Gill, C., Lu, C.: Federated scheduling for stochastic parallel real-time tasks. In: IEEE 20th International Conference on Embedded and Real-Time Computing Systems and Applications, pp. 1–10 (2014). https://doi.org/10.1109/RTCSA.2014.6910549 . IEEE

[25] Guan, F., Peng, L., Qiao, J.: A new federated scheduling algorithm for arbitrary-deadline dag tasks. IEEE Transactions on Computers **72**(8), 2264–2277 (2023) https://doi.org/10.1109/TC.2023.3244632

[26] Lin, C.-C., Shi, J., Ueter, N., Günzel, M., Reineke, J., Chen, J.-J.: Type-aware federated scheduling for typed dag tasks on heterogeneous multicore platforms. IEEE transactions on computers **72**(5), 1286–1300 (2022) https://doi.org/10.1109/TC.2022.3202748

[27] Moiş, G.D., Sanislav, T., Folea, S.C., Zeadally, S.: Performance evaluation of energy-autonomous sensors using power-harvesting beacons for environmental monitoring in internet of things (iot). Sensors **18**(6) (2018) https://doi.org/10.3390/s18061709

[28] Berndt, R., Wabeke, L., Van Rensburg, V.J., Potgieter, F., Kloke, K.: Ground-based surveillance and classification radar for wildlife protection. In: 2023 IEEE International Radar Conference (RADAR), pp. 1–5 (2023). https://doi.org/10.1109/RADAR54928.2023.10371030

[29] Alatise, M.B., Hancke, G.P.: A review on challenges of autonomous mobile robot and sensor fusion methods. IEEE Access **8**, 39830–39846 (2020) https://doi.org/10.1109/ACCESS.2020.2975643

[30] Steccanella, L., Bloisi, D.D., Castellini, A., Farinelli, A.: Waterline and obstacle detection in images from low-cost autonomous boats for environmental monitoring. Robotics and Autonomous Systems **124**, 103346 (2020) https://doi.org/10.1016/j.robot.2019.103346

[31] Sotelo-Torres, F., Alvarez, L.V., Roberts, R.C.: An unmanned surface vehicle (usv): Development of an autonomous boat with a sensor integration system for bathymetric surveys. Sensors **23**(9) (2023) https://doi.org/10.3390/s23094420

[32] Catlett, C.E., Beckman, P.H., Sankaran, R., Galvin, K.K.: Array of things: a scientific research instrument in the public way: platform design and early lessons learned. SCOPE '17, pp. 26–33. Association for Computing Machinery, New York, NY, USA (2017). https://doi.org/10.1145/3063386.3063771 . https://doi.org/10.1145/3063386.3063771

[33] Hossein Motlagh, N., Kortoçi, P., Su, X., Lovén, L., Hoel, H.K., Bjerkestrand Haugsvær, S., Srivastava, V., Gulbrandsen, C.F., Nurmi, P., Tarkoma, S.: Unmanned aerial vehicles for air pollution monitoring: A survey. IEEE Internet of Things Journal **10**(24), 21687–21704 (2023) https://doi.org/

10.1109/JIOT.2023.3290508

[34] Brennan Whitfield: High-performance computing, or HPC, may sound niche, but it influences basically everything. https://builtin.com/hardware/high-performance-computing-applications (Accessed on 2024-04-10)

[35] EL Ghor, H., Chetto, M., Chehade, R.H.: A real-time scheduling framework for embedded systems with environmental energy harvesting. Computers and Electrical Engineering **37**(4), 498–510 (2011) https://doi.org/10.1016/j.compeleceng.2011.05.003

[36] Chetto, M., Masson, D., Midonnet, S.: Fixed priority scheduling strategies for ambient energy-harvesting embedded systems. In: IEEE/ACM International Conference on Green Computing and Communications, pp. 50–55 (2011). https://doi.org/10.1109/GreenCom.2011.17 . IEEE

[37] Moser, C., Brunelli, D., Thiele, L., Benini, L.: Real-time scheduling with regenerative energy. In: 18th Euromicro Conference on Real-Time Systems (ECRTS'06), pp. 10–270 (2006). https://doi.org/10.1109/ECRTS.2006.23

[38] Abdeddaim, Y., Chandarli, Y., Masson, D.: The optimality of pfpasap algorithm for fixed-priority energy-harvesting real-time systems. In: 25th Euromicro Conference on Real-Time Systems, pp. 47–56 (2013). https://doi.org/10.1109/ECRTS.2013.16

[39] Blumofe, R.D., Leiserson, C.E.: Scheduling multithreaded computations by work stealing. J. ACM **46**(5), 720–748 (1999) https://doi.org/10.1145/324133.324234

[40] Agrawal, K., Leiserson, C.E., He, Y., Hsu, W.J.: Adaptive work-stealing with parallelism feedback. ACM Trans. Comput. Syst. **26**(3) (2008) https://doi.org/10.1145/1394441.1394443

[41] Grama, A., Karypis, G., Kumar, V., Gupta, A.: Introduction to Parallel Computing (2nd Edition), (2003)

[42] He, Y., Leiserson, C.E., Leiserson, W.M.: The cilkview scalability analyzer. In: Proceedings of the Twenty-Second Annual ACM Symposium on Parallelism in Algorithms and Architectures. SPAA '10, pp. 145–156. Association for Computing Machinery, New York, NY, USA (2010). https://doi.org/10.1145/1810479.1810509

[43] Schardl, T.B., Kuszmaul, B.C., Lee, I.-T.A., Leiserson, W.M., Leiserson, C.E.: The cilkprof scalability profiler. In: Proceedings of the 27th ACM Symposium on Parallelism in Algorithms and Architectures. SPAA '15, pp. 89–100. Association for Computing Machinery, New York, NY, USA (2015). https://doi.org/10.1145/2755573.2755603

[44] He, Q., jiang, x., Guan, N., Guo, Z.: Intra-task priority assignment in real-time scheduling of dag tasks on multi-cores. IEEE Transactions on Parallel and Distributed Systems **30**(10), 2283–2295 (2019) https://doi.org/10.1109/TPDS.2019.2910525

[45] Wang, L., Khan, S.U., Chen, D., Kołodziej, J., Ranjan, R., Xu, C.-z., Zomaya, A.: Energy-aware parallel task scheduling in a cluster. Future Generation Computer Systems **29**(7), 1661–1670 (2013) https://doi.org/10.1016/j.future.2013.02.010

[46] Aydin, H., Yang, Q.: Energy-aware partitioning for multiprocessor real-time systems. In: Proceedings of the 17th International Symposium on Parallel and Distributed Processing. IPDPS '03, pp. 113–2. IEEE Computer Society, USA (2003)

[47] Devadas, V., Aydin, H.: Coordinated power management of periodic real-time tasks on chip multiprocessors. In: International Conference on Green Computing, pp. 61–72 (2010). https://doi.org/10.1109/GREENCOMP.2010.5598261

[48] Bhuiyan, A., Guo, Z., Saifullah, A., Guan, N., Xiong, H.: Energy-efficient real-time scheduling of dag tasks. ACM Trans. Embed. Comput. Syst. **17**(5) (2018) https://doi.org/10.1145/3241049

[49] Juarez, F., Ejarque, J., Badia, R.M.: Dynamic energy-aware scheduling for parallel task-based application in cloud computing. Future Generation Computer Systems **78**, 257–271 (2018) https://doi.org/10.1016/j.future.2016.06.029

[50] Pagani, S., Chen, J.-J.: Energy efficient task partitioning based on the single frequency approximation scheme. In: IEEE 34th Real-Time Systems Symposium, pp. 308–318 (2013). https://doi.org/10.1109/RTSS.2013.38

[51] Knap, V., Vestergaard, L.K., Stroe, D.-I.: A review of battery technology in cubesats and small satellite solutions. Energies **13**(16) (2020) https://doi.org/10.3390/en13164097

[52] Zhao, S., Dai, X., Bate, I., Burns, A., Chang, W.: Dag scheduling and analysis on multiprocessor systems: Exploitation of parallelism and dependency. In: IEEE Real-Time Systems Symposium (RTSS), pp. 128–140 (2020). https://doi.org/10.1109/RTSS49844.2020.00022

[53] Eager, D., Zahorjan, J., Lazowska, E.: Speedup versus efficiency in parallel systems. IEEE Trans. Computers **38**, 408–423 (1989)

[54] Brent, R.P.: The parallel evaluation of general arithmetic expressions. J. ACM **21**(2), 201–206 (1974) https://doi.org/10.1145/321812.321815

[55] Sen, S.: Dynamic processor allocation for adaptively parallel work-stealing jobs. PhD thesis, Massachusetts Institute of Technology (2004)

[56] Graham, R.L.: Bounds on multiprocessing timing anomalies. SIAM Journal on Applied Mathematics **17**(2), 416–429 (1969)

[57] Melani, A., Bertogna, M., Bonifaci, V., Marchetti-Spaccamela, A., Buttazzo, G.C.: Response-time analysis of conditional dag tasks in multiprocessor systems. In: 27th Euromicro Conference on Real-Time Systems, pp. 211–221 (2015). https://doi.org/10.1109/ECRTS.2015.26

[58] Fonseca, J., Nelissen, G., Nélis, V.: Improved response time analysis of sporadic dag tasks for global fp scheduling. In: Proceedings of the 25th International Conference on Real-Time Networks and Systems. RTNS '17, pp. 28–37. Association for Computing Machinery, New York, NY, USA (2017). https://doi.org/10.1145/3139258.3139288

[59] Abdeddaïm, Y., Chandarli, Y., Davis, R.I., Masson, D.: Response time analysis for fixed priority real-time systems with energy-harvesting. Real-Time Systems **52**(2), 125–160 (2016) https://doi.org/10.1007/s11241-015-9239-7

[60] Lorenzon, A.F., Oliveira, C.C., Souza, J.D., Beck, A.C.S.: Aurora: Seamless optimization of openmp applications. IEEE Transactions on Parallel and Distributed Systems **30**(5), 1007–1021 (2019) https://doi.org/10.1109/TPDS.2018.2872992

[61] Shafik, R.A., Das, A., Yang, S., Merrett, G., Al-Hashimi, B.M.: Adaptive energy minimization of openmp parallel applications on many-core systems. In: Proceedings of the 6th Workshop on Parallel Programming and Run-Time Management Techniques for Many-Core Architectures. PARMA-DITAM '15, pp. 19–24. Association for Computing Machinery, New York, NY, USA (2015). https://doi.org/10.1145/2701310.2701311

[62] Lesh, N., Marks, J., McMahon, A., Mitzenmacher, M.: Exhaustive approaches to 2d rectangular perfect packings. Inf. Process. Lett. **90**(1), 7–14 (2004) https://doi.org/10.1016/j.ipl.2004.01.006

[63] Bezerra, V.M.R., Leao, A.A.S., Oliveira, J.F., Santos, M.O.: Models for the two-dimensional level strip packing problem – a review and a computational evaluation. Journal of the Operational Research Society **71**(4), 606–627 (2020) https://doi.org/10.1080/01605682.2019.1578914

[64] Bortfeldt, A.: A genetic algorithm for the two-dimensional strip packing problem with rectangular pieces. European Journal of Operational Research **172**(3), 814–837 (2006) https://doi.org/10.1016/j.ejor.2004.11.016

[65] Bini, E., Buttazzo, G.C.: Measuring the performance of schedulability tests. Real-Time Systems **30**, 129–154 (2005)

[66] Goossens, J., Macq, C.: Limitation of the hyper-period in real-time periodic task

set generation. In: In Proceedings of the 9th International Conference on Real-time Systems, pp. 133–148 (2001). https://api.semanticscholar.org/CorpusID:18430064

[67] Rouxel, B., Puaut, I.: Str2rts: Refactored streamit benchmarks into statically analyzable parallel benchmarks for wcet estimation & real-time scheduling. In: 17th International Workshop on Worst-Case Execution Time Analysis (WCET 2017) (2017). https://doi.org/10.4230/OASIcs.WCET.2017.1 . Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik

[68] Ramos, J., Andreas, A.: University of texas panamerican (utpa): Solar radiation lab (srl); edinburg, texas (data). Technical report, National Renewable Energy Lab.(NREL), Golden, CO (United States) (2011). https://doi.org/10.7799/1052555

[69] Guo, Z., Bhuiyan, A., Saifullah, A., Guan, N., Xiong, H.: Energy-efficient multi-core scheduling for real-time dag tasks. In: 29th Euromicro Conference on Real-time Systems (ECRTS 2017), pp. 156–168 (2017). https://doi.org/10.4230/LIPICS.ECRTS.2017.22 . Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik

[70] Chetto, M., Osta, R.E.: Real-time scheduling of dag tasks in self-powered sensors with scavenged energy. In: 4th International Conference on Emerging Trends in Electrical, Electronic and Communications Engineering (ELECOM), pp. 1–6 (2022). https://doi.org/10.1109/ELECOM54934.2022.9965240

[71] Singhal, C.: Sustainable application support in battery-less iot sensing network system. In: 2023 IEEE International Conference on Communications Workshops (ICC Workshops), pp. 1277–1282 (2023). https://doi.org/10.1109/ICCWorkshops57953.2023.10283554

[72] Hester, J., Sitanayah, L., Sorber, J.: Tragedy of the coulombs: Federating energy storage for tiny, intermittently-powered sensors. SenSys '15, pp. 5–16. Association for Computing Machinery, New York, NY, USA (2015). https://doi.org/10.1145/2809695.2809707

[73] Emberson, P., Stafford, R., Davis, R.I.: Techniques for the synthesis of multiprocessor tasksets. In: Proceedings 1st International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS 2010), pp. 6–11 (2010)

[74] Nasri, M., Nelissen, G., Brandenburg, B.B.: Response-time analysis of limited-preemptive parallel dag tasks under global scheduling. In: 31st Conference on Real-Time Systems, pp. 21–1 (2019). https://doi.org/LIPIcs.ECRTS.2019.21

[75] Shirazi, M., Thiele, L., Kargahi, M.: Energy-resilient real-time scheduling. IEEE Transactions on Computers **72**(1), 69–81 (2023) https://doi.org/10.1109/TC.2022.3202754

[76] Hasanloo, M., Kargahi, M., Jalilian, S.: Dynamic harvesting- and energy-aware real-time task scheduling. Sustainable Computing: Informatics and Systems **28**, 100413 (2020) https://doi.org/j.suscom.2020.100413

[77] Wang, K., Lin, Y., Deng, Q.: Response time analysis for energy-harvesting mixed-criticality systems. In: 2022 Design, Automation and Test in Europe Conference and Exhibition (DATE), pp. 1293–1298 (2022). https://doi.org/10.23919/DATE54114.2022.9774646

[78] Cui, Y., Zhang, W., Chaturvedi, V., He, B.: Decentralized thermal-aware task scheduling for large-scale many-core systems. IEEE Transactions on Very Large Scale Integration (VLSI) Systems **24**(6), 2075–2088 (2015) https://doi.org/10.1109/TVLSI.2015.2497469

[79] Allavena, A., Mosse, D.: Scheduling of frame-based embedded systems with rechargeable batteries. In: Workshop on Power Management for Real-time and Embedded Systems (in Conjunction with RTAS 2001) (2001)

[80] Jayaseelan, R., Mitra, T., Li, X.: Estimating the worst-case energy consumption of embedded software. In: 12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'06), pp. 81–90 (2006). https://doi.org/10.1109/RTAS.2006.17 . IEEE

[81] Faramarzi, K., Hasanloo, M., Kargahi, M.: The pfpasap algorithm for energy harvesting real-time systems with a non-ideal supercapacitor. In: 5th International Conference on Computer and Knowledge Engineering (ICCKE), pp. 279–284 (2015). https://doi.org/10.1109/ICCKE.2015.7365842

[82] Aerabi, E., Fazeli, M., Hély, D.: Cyense: Cyclic energy-aware scheduling for energy-harvested embedded systems. Microprocessors and Microsystems **89**, 104421 (2022) https://doi.org/j.micpro.2021.104421

[83] Wang, K., Lin, Y., Deng, Q.: Response time analysis for energy-harvesting mixed-criticality systems. In: Design, Automation and Test in Europe Conference and Exhibition (DATE), pp. 1293–1298 (2022). https://doi.org/10.23919/DATE54114.2022.9774646

[84] Wang, K., Deng, Q.: Mixed-criticality scheduling of energy-harvesting systems. In: IEEE Real-Time Systems Symposium (RTSS), pp. 435–446 (2022). https://doi.org/10.1109/RTSS55097.2022.00044

[85] Lee, W.Y.: Energy-efficient scheduling of periodic real-time tasks on lightly loaded multicore processors. IEEE Transactions on Parallel and Distributed Systems **23**(3), 530–537 (2012) https://doi.org/10.1109/TPDS.2011.87

[86] Seo, E., Jeong, J., Park, S., Lee, J.: Energy efficient scheduling of real-time tasks on multicore processors. IEEE Transactions on Parallel and Distributed Systems

19(11), 1540–1552 (2008) https://doi.org/10.1109/TPDS.2008.104

[87] Narayana, S., Huang, P., Giannopoulou, G., Thiele, L., Prasad, R.V.: Exploring energy saving for mixed-criticality systems on multi-cores. In: IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), pp. 1–12 (2016). https://doi.org/10.1109/RTAS.2016.7461336

[88] Chen, G., Huang, K., Knoll, A.: Abstract: Energy optimization for real-time multiprocessor system-on-chip with optimal dvfs and dpm combination. In: The 11th IEEE Symposium on Embedded Systems for Real-time Multimedia, pp. 40–40 (2013). https://doi.org/10.1109/ESTIMedia.2013.6704501

[89] Wang, L., Lu, Y.: Efficient power management of heterogeneous soft real-time clusters. In: Real-Time Systems Symposium, pp. 323–332 (2008). https://doi.org/10.1109/RTSS.2008.31

[90] Liu, C., Li, J., Huang, W., Rubio, J., Speight, E., Lin, X.: Power-efficient time-sensitive mapping in heterogeneous systems. In: Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques. PACT '12, pp. 23–32. Association for Computing Machinery, New York, NY, USA (2012). https://doi.org/10.1145/2370816.2370822

[91] Chen, J.-J., Schranzhofer, A., Thiele, L.: Energy minimization for periodic real-time tasks on heterogeneous processing units. In: IEEE International Symposium on Parallel and Distributed Processing, pp. 1–12 (2009). https://doi.org/10.1109/IPDPS.2009.5161024

[92] Zhang, Y.-W.: Dvfs-based energy-aware scheduling of imprecise mixed-criticality real-time tasks. Journal of Systems Architecture **137**, 102849 (2023) https://doi.org/10.1016/j.sysarc.2023.102849

[93] Yang, W., Zhao, M., Li, J., Zhang, X.: Energy-efficient dag scheduling with dvfs for cloud data centers. The Journal of Supercomputing, 1–25 (2024) https://doi.org/10.1007/s11227-024-06035-7

[94] Qi, X., Zhu, D.-K.: Energy efficient block-partitioned multicore processors for parallel applications. Journal of Computer Science and Technology **26**(3), 418–433 (2011) https://doi.org/10.1007/s11390-011-1144-5

[95] Guo, Z., Bhuiyan, A., Liu, D., Khan, A., Saifullah, A., Guan, N.: Energy-efficient real-time scheduling of dags on clustered multi-core platforms. In: IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), pp. 156–168 (2019). https://doi.org/10.1109/RTAS.2019.00021

[96] Chen, J., He, Y., Zhang, Y., Han, P., Du, C.: Energy-aware scheduling for dependent tasks in heterogeneous multiprocessor systems. Journal of Systems Architecture **129**, 102598 (2022) https://doi.org/10.1016/j.sysarc.2022.102598

[97] Khaleghzadeh, H., Fahad, M., Shahid, A., Manumachu, R.R., Lastovetsky, A.: Bi-objective optimization of data-parallel applications on heterogeneous hpc platforms for performance and energy through workload distribution. IEEE Transactions on Parallel and Distributed Systems **32**(3), 543–560 (2021) https://doi.org/10.1109/TPDS.2020.3027338

[98] Sharma, Y., Moulik, S.: Fats-2tc: A fault tolerant real-time scheduler for energy and temperature aware heterogeneous platforms with two types of cores. Microprocessors and Microsystems **96**, 104744 (2023) https://doi.org/10.1016/j.micpro.2022.104744

[99] Sharma, Y., Moulik, S.: Rt-seat: A hybrid approach based real-time scheduler for energy and temperature efficient heterogeneous multicore platforms. Results in Engineering **16**, 100708 (2022) https://doi.org/10.1016/j.rineng.2022.100708

[100] Chen, J., Han, P., Zhang, Y., You, T., Zheng, P.: Scheduling energy consumption-constrained workflows in heterogeneous multi-processor embedded systems. Journal of Systems Architecture **142**, 102938 (2023) https://doi.org/10.1016/j.sysarc.2023.102938

[101] Niu, L., Rawat, D.B., Musselwhite, J., Gu, Z., Deng, Q.: Energy-constrained scheduling for weakly hard real-time systems using standby-sparing. ACM Trans. Des. Autom. Electron. Syst. **29**(2) (2024) https://doi.org/10.1145/3631587

[102] Zhang, Y.-W., Zheng, H., Gu, Z.: Energy-aware adaptive mixed-criticality scheduling with semi-clairvoyance and graceful degradation. ACM Trans. Embed. Comput. Syst. **23**(1) (2024) https://doi.org/10.1145/3632749

[103] Fan, K., D'Antonio, M., Carpentieri, L., Cosenza, B., Ficarelli, F., Cesarini, D.: Synergy: Fine-grained energy-efficient heterogeneous computing for scalable energy saving. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 1–13 (2023). https://doi.org/10.1145/3581784.3607055

[104] Nazeri, M., Soltanaghaei, M., Khorsand, R.: A predictive energy-aware scheduling strategy for scientific workflows in fog computing. Expert Systems with Applications **247**, 123192 (2024) https://doi.org/10.1016/j.eswa.2024.123192

[105] Rattihalli, G., Hogade, N., Dhakal, A., Frachtenberg, E., Enriquez, R.P.H., Bruel, P., Mishra, A., Milojicic, D.: Fine-grained heterogeneous execution framework with energy aware scheduling. In: 2023 IEEE 16th International Conference on Cloud Computing (CLOUD), pp. 35–44 (2023). https://doi.org/10.1109/CLOUD60044.2023.00014 . IEEE

[106] Hu, B., Yang, X., Zhao, M.: Online energy-efficient scheduling of dag tasks on heterogeneous embedded platforms. Journal of Systems Architecture **140**, 102894 (2023) https://doi.org/10.1016/j.sysarc.2023.102894

[107] Stewart, R., Raith, A., Sinnen, O.: Optimising makespan and energy consumption in task scheduling for parallel systems. Computers & Operations Research **154**, 106212 (2023) https://doi.org/10.1016/j.cor.2023.106212

[108] Cui, M., Kritikakou, A., Mo, L., Casseau, E.: Near-optimal energy-efficient partial-duplication task mapping of real-time parallel applications. Journal of Systems Architecture **134**, 102790 (2023) https://doi.org/10.1016/j.sysarc.2022.102790

[109] Zhang, Y.-W.: Energy efficient non-preemptive scheduling of imprecise mixed-criticality real-time tasks. Sustainable Computing: Informatics and Systems **37**, 100840 (2023) https://doi.org/10.1016/j.suscom.2022.100840

[110] Karbasioun, M.M., Shaikhet, G., Kranakis, E., Lambadaris, I.: Power strip packing of malleable demands in smart grid. In: IEEE International Conference on Communications (ICC), pp. 4261–4265 (2013). IEEE

[111] Liu, F.-H., Liu, H.-H., Wong, P.W.: Non-preemptive scheduling in a smart grid model and its implications on machine minimization. Algorithmica **82**(12), 3415–3457 (2020) https://doi.org/10.1007/s00453-020-00733-3

[112] Raj, B.D., Sarkar, A., Goswami, D.: An efficient framework for brownout based appliance scheduling in microgrids. Sustainable Cities and Society, 103936 (2022) https://doi.org/10.1016/j.scs.2022.103936

[113] Yang, P., Chavali, P., Gilboa, E., Nehorai, A.: Parallel load schedule optimization with renewable distributed generators in smart grids. IEEE Transactions on Smart Grid **4**(3), 1431–1441 (2013) https://doi.org/10.1109/TSG.2013.2264728

[114] Lee, J., Kim, H.-J., Park, G.-L., Kang, M.: Energy consumption scheduler for demand response systems in the smart grid. Journal of Information Science & Engineering **28**(5), 955–969 (2012)

[115] Padhi, S., Subramanyam, R.: User request-based scheduling algorithms by managing uncertainty of renewable energy. Cluster Computing, 1–18 (2023) https://doi.org/10.1007/s10586-023-04057-z

[116] Ertem, M.: Renewable energy-aware machine scheduling under intermittent energy supply. IEEE Access **12**, 23613–23625 (2024) https://doi.org/10.1109/ACCESS.2024.3365074

[117] Min, S.-H., Lee, S.-W., Kim, H.-J.: Parallel machine scheduling with peak energy consumption limits. IEEE Transactions on Automation Science and Engineering, 1–13 (2024) https://doi.org/10.1109/TASE.2024.3366949

[118] Park, H.-G., Kang, D.-K.: Renewable-aware frequency scaling approach for energy-efficient deep learning clusters. Applied Sciences **14**(2) (2024) https:

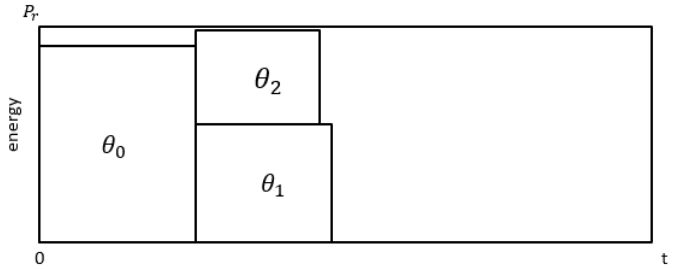


Fig. A1: An example of the power supply of three power demands θ_0, θ_1, and θ_2 is given here. The power supply of θ_1 and θ_2 is done in parallel, and both of these demands must wait for the power supply of θ_0 to be completed.

//doi.org/10.3390/app14020776

[119] Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman and Company., USA (1979)

Appendix A Proof of Theorem 2

First we consider Problem 1 more straightforwardly, without considering the battery.

Let the set of power demand θ consist of n_{pd} power requirements from the system tasks, represented as:

$$\theta = \bigcup \theta_i, 1 \leq i \leq n \tag{A1}$$

where each θ_i is defined as $\theta_i = (\pi_i, \omega_i)$ and represents π_i steps of constant power requirement of size ω_i.

Suppose that in the time period $[0, t_1]$, there is a constant input energy rate of P_r. This input power can be considered as a strip with a fixed width of P_r and a length of t_1. An example of the power supply of three power demands is shown in Figure A1. In this example, the power supply of demand θ_0 delays demand θ_2, but the power supply of demand θ_1 does not delay the power supply of θ_0 and can be done in parallel. In practice, when there are many power demands, several steps of the θ_i power demand may be provided in parallel with the θ_j, and the rest of the steps may be provided in series.

In these circumstances, the problem resembles a so-called strip-packing problem (2SP) of n_{pd} power demands θ_i each with a length of π_i and a width of ω_i on the power strip with a width of P_r and a length of t_1, which is an NP-hard problem in a strong sense (since the particular case where all items have the same height is equivalent to the one-dimensional bin packing) [119].

In our problem, we have a similar energy scheduling problem, but with the addition of a battery with capacity E_{max}. This allows energy replenished in one step to be utilized in subsequent steps or used to respond to demands exceeding the input energy rate. This problem, too, is NP-hard, as the presence of a polynomial time algorithm

for this problem would also lead to a polynomial time solution for the NP-hard strip packing problem by simply setting the battery capacity to zero.

Appendix B Proof of theorem 4

Proof. To prove theorem 4, we must demonstrate that the energy allocation in Step s, where $1 \leq s \leq W_{i,m}^{max}$, to the task τ_i with m dedicated cores, using $\varphi_i^m(s)$, is both necessary and sufficient. For its necessity, in each Step s, any amount of energy less than $\varphi_i^m(s)$ can simply be regarded as insufficient. This is evident from the energy requests stemming from the worst-case execution time of task τ_i or (3), which can serve as counterexamples.

For the sufficiency, the worst case of energy demand in each step is m units, which in the first $\lceil \frac{(C_i - L_i)}{m} \rceil$ steps of Formula 10 the same amount of energy provided. We need to show that in the following steps, firstly, there is enough energy for the rest of the execution, and secondly, there is enough time to run with this power supply function. The total energy supplied in Step 1 to Step $\lfloor \frac{C_i - L_i}{m} \rfloor + L_i$ is equal to:

$$\sum_{s=1}^{\lfloor \frac{C_i - L_i}{m} \rfloor + L_i} \varphi_i^m(s) = p_i * (m * \lceil \frac{(C_i - L_i)}{m} \rceil + C_i - m * \lceil \frac{C_i - L_i}{m} \rceil) = C_i * p_i$$

It should be noted that in each step, a maximum of m units of work can be performed. Suppose that in Step $\lceil \frac{(C_i - L_i)}{m} \rceil + 1$, there is a request to execute m units of work. In this step, only one unit of energy is provided, and the execution of this step is delayed until the following m steps (one unit of energy is provided in each step).

Now, considering that in each step, a maximum of m units of the task can be executed, we will have the following conditions:

- If $m > L$, then given that the sum of the remaining work is at most equal to L, then in one step there cannot be as much work left as all the m cores, and therefore in the last step (i.e., Step $\lfloor \frac{(C_i - L_i)}{m} \rfloor + L_i$) All the required energy is provided and the remaining work can be executed in the last step.
- If $m < L$, then it is possible to use all cores in each step, so in each m step, all cores will be used, and the number of these executions will be equal to L/m.

Now consider a situation where in the initial steps, we have a minimum of execution (only one core per step) and most of the execution is postponed to the last steps.

If in the first steps only one core is used in each step, according to Lemma1, the length of these steps, which are all incomplete, is equal to L_i, and the task must be executed in the next $\lfloor \frac{C_i - L_i}{m} \rfloor$ steps and again based on Lemma1 All of these steps are complete, and all cores are utilized. We must show that in each Step s where $L_i < s \leq \lfloor \frac{C_i - L_i}{m} \rfloor + L_i$ there is at least m units of energy, And the execution of each step are done without any delay due to lack of energy, i.e.:

$$\forall s \text{ sush that } L_i < s \leq \lfloor \frac{C_i - L_i}{m} \rfloor + L_i,$$
$$\underbrace{(m * \lceil \frac{C_i - L_i}{m} \rceil)}_{\text{(B2)}} + \underbrace{min((s - \lfloor \frac{C_i - L_i}{m} \rfloor), C_i - m * \lceil \frac{C_i - L_i}{m} \rceil)}_{\text{(B3)}}$$
$$- \underbrace{(C_i - m * \lfloor \frac{C_i - L_i}{m} \rfloor}_{\text{(B4)}} - \underbrace{((s - L_i - 1) * m)}_{\text{(B5)}} \geq m$$

which each part of this equation is as follows:

- (B2): In the first $\lceil \frac{C_i - L_i}{m} \rceil$ steps, m units of energy are received in each step until Step s.
- (B3): Total one unit energy received until Step s.
- (B4): Total one unit energy consumed.
- (B5): After Step L_i, m units of energy are consumed in each step.

Now we can consider two cases for s:

- First case, $L_i < s \leq \lceil \frac{C_i - L_i}{m} \rceil$: In this case, according to the function $\varphi(s)$ in each Step m unit of energy will be provided.
- The second case $s > \lceil \frac{C_i - L_i}{m} \rceil$: Given that in this case, the energy input rate is 1 and the energy consumption rate is m and $m \geq 1$, so the worst case of energy shortage occurs in the last step, i.e., Step $s = \lfloor \frac{C_i - L_i}{m} \rfloor + L_i$:

$$\begin{aligned} & m * \lceil \frac{C_i - L_i}{m} \rceil + C_i - m * \lceil \frac{C_i - L_i}{m} \rceil \\ & \quad - C_i - m * \lfloor \frac{C_i - L_i}{m} \rfloor - (\lfloor \frac{C_i - L_i}{m} \rfloor + L_i - L_i - 1) * m \\ & = m \geq m \end{aligned}$$

□

Appendix C Proof of HOA Optimality

To prove the optimality of the Hypothetical Optimal Algorithm (HOA), we demonstrate that it performs optimally within the defined constraints and assumptions. Although HOA is not fully implemented due to its exponential state space complexity, we prove its optimality by showing that, under pessimistic assumptions, no algorithm can outperform it.

Assumptions and Problem Definition

The problem is to schedule parallel tasks in an energy-harvesting environment, where the system must decide whether tasks can be scheduled within a given time window $[\alpha, \beta]$, based on both:

- **Dynamic energy**: Energy required to execute the tasks.
- **Static energy**: Energy required to keep cores active during the scheduling window.

We consider two variants of the Hypothetical Optimal Algorithm:

- **HOA-F**: Assumes Federated Scheduling, where static energy is based on the minimum number of cores required for federated scheduling.
- **HOA-G**: Assumes Global Scheduling, where static energy is based on the sum of task utilizations.

Conditions for Scheduling

The HOA schedules tasks within a window $[\alpha, \beta]$ if and only if the total energy harvested during that window, combined with the energy stored in a fully charged battery, is greater than or equal to the energy required to:

- Complete the tasks within the window.
- Sustain the static energy required by the cores.

For **HOA-F**, the scheduling condition is:

$$\sum_{j=\alpha}^{\beta} P_R(j) + B \geq \sum_{\tau_i \in \tau} \left\lceil \frac{(\beta - \alpha)}{D_i} \right\rceil C_i p_i + P_{\text{static}} \times \text{minCores} \times (\beta - \alpha)$$

For **HOA-G**, the scheduling condition is:

$$\sum_{j=\alpha}^{\beta} P_R(j) + B \geq \sum_{\tau_i \in \tau} \left\lceil \frac{(\beta - \alpha)}{D_i} \right\rceil C_i p_i + P_{\text{static}} \times \sum_{\tau_i \in \tau} u_i \times (\beta - \alpha)$$

Proof by Contradiction

Step 1: Suppose there exists a better algorithm A' Assume there exists another algorithm A' that can schedule tasks more efficiently than HOA within the same constraints (harvested energy, fully charged battery, and the same task set).
Step 2: Analyze A'**'s behavior** If A' schedules the same tasks in the same time window $[\alpha, \beta]$, it must satisfy the same energy constraints (both dynamic and static energy) as HOA. Thus, the total energy required by A' for dynamic and static consumption is at least the same as for HOA.
Step 3: Contradiction based on energy constraints Since A' cannot use less energy for scheduling the tasks (because static energy is a physical constraint of the cores, and dynamic energy is determined by task execution), A' must either:

- Harvest more energy than HOA (which is impossible under the same energy harvesting model).
- Use less energy, implying that it bypasses the static or dynamic energy constraints and violates the physical requirements of task execution.

Step 4: Conclusion Therefore, no algorithm A' can schedule tasks more efficiently than HOA under the given constraints. This shows that HOA represents the **optimal boundary** for task scheduling under energy-harvesting conditions, as it maximally utilizes the available energy while considering both dynamic and static energy consumption.