QoS-Aware Approximate Task Mapping on Heterogeneous Multicore Platforms with DVFS and Task Migration

Hengyan Song, Lei Mo, Tamim M Al-Hasan, Angeliki Kritikakou, Xiaojun Zhai, Shibo He, Olivier Sentieys

QoS-Aware Approximate Task Mapping on Heterogeneous Multicore Platforms with DVFS and Task Migration

HENGYAN SONG, School of Automation, Southeast University, China
LEI MO, School of Automation, Southeast University, China
TAMIM M. AL-HASAN, School of Computer Science and Electronic Engineering, University of Essex, UK
ANGELIKI KRITIKAKOU, University of Rennes, INRIA, IRISA, CNRS, France
XIAOJUN ZHAI, School of Computer Science and Electronic Engineering, University of Essex, UK
SHIBO HE, College of Control Science and Engineering, Zhejiang University, China
OLIVIER SENTIEYS, University of Rennes, INRIA, IRISA, CNRS, France

Heterogeneous Multicore Platforms (HMPs) have been widely adopted to execute tasks across a range of applications. Under limited system resources and diverse application requirements, allocating and executing dependent Approximate Computing (AC) tasks on these platforms to achieve high Quality-of-Service (QoS) is challenging. Dynamic Voltage and Frequency Scaling (DVFS) and task migration have proven effective for improving QoS while balancing time and energy consumption. However, existing approaches often overlook the migration overhead and the resulting dynamic changes in task dependencies, which can adversely affect mapping outcomes. To address these issues, this paper presents a novel AC task mapping method that maximizes system QoS under multiple constraints on HMPs, accounting for task migration overhead, DVFS, and changes in Directed Acyclic Graph (DAG) topology. We first formulate this joint design problem as a complex nonlinear programming problem. Next, we linearize the nonlinear terms without performance loss by introducing auxiliary variables and additional constraints. Building on this formulation, we propose an optimal (OPT) and a low-complexity Heuristic Algorithm (HEU), derived from problem decomposition and a greedy strategy, which divides the Mixed-Integer Non-Linear Programming (MINLP) problem into two smaller subproblems with fewer variables and constraints, solving them sequentially. The simulation results show that the proposed OPT method achieves higher QoS performance, measured at about 2.389 times on average and up to 4.115 times, while its feasibility is increased to about 3.263 times on average and up to 9.667 times, compared to other state-of-the-art methods. In addition, the average QoS of the proposed HEU method is about 0.577 times that of the proposed method, but its computation time is over a thousand times shorter.

Additional Key Words and Phrases: Approximate Computation Task, DVFS, Heterogeneous Multicore Platform, Quality-of-Service, Task Mapping, Task Migration

1 Introduction

Recently, a wide range of applications, such as multimedia, encryption, and control, have been deployed on flexible and high-performance embedded computation systems to meet demanding performance requirements [20]. Because different applications have distinct resource demands, traditional single-core or homogeneous multicore platforms cannot efficiently handle both light and heavy workloads simultaneously [13]. Consequently, Heterogeneous Multicore Platforms (HMPs), which integrate different types of processors on a single chip, are gaining popularity for improving application performance [15]. For example, Arm® big.LITTLE™ [7] configurations commonly feature high-performance processors (e.g., Arm® Cortex® A15) to manage compute-intensive applications, as well as energy-efficient processors (e.g., Arm® Cortex® A7) to run lightweight or background applications. This configuration supports flexible processor scheduling based on different operating conditions and requirements [29].

On the one hand, multicore embedded systems often operate under limited energy budgets, such as battery-powered devices (e.g., UAVs, AGVs, or mobile phones) used in mobile and edge computing [17]. On the other hand, these systems must guarantee real-time execution of tasks [2]. Missing a real-time deadline, for instance, in target tracking or feedback control, may result in

significant performance degradation (e.g., losing the target or experiencing greater control error). Increasing the Voltage/Frequency (V/F) level can accelerate execution and improve real-time performance, but it also raises energy consumption. As the processor count grows, higher energy usage can lead to thermal challenges [31]; thus affecting overall reliability.

A widely used approach to address these conflicting requirements is Dynamic Voltage and Frequency Scaling (DVFS), an adaptive method that dynamically adjusts a processor's operating voltage and frequency during task execution. By selecting appropriate V/F levels, DVFS-capable processors can trade off computation energy against execution time [25, 32]. In some applications, such as image processing or iterative algorithms, tasks can be formulated under an Approximate Computing (AC) model [26]. In this model, each AC task is divided into a mandatory part that ensures an acceptable baseline result and an optional part that refines accuracy. Quality-of-Service (QoS) of an AC task often corresponds to the number of optional cycles executed. A linear or concave function typically describes the relationship between executed optional cycles and resultant QoS [8, 36]. Such applications are usually computationally intensive but error-tolerant, where AC helps reduce resource consumption without a substantial impact on accuracy. For example, JPEG supports lossy and lossless image compressions [35], where some components, e.g., Tier-2 and iDWT, can be modeled as AC tasks, as the length of execution cycles affects image quality.

By mapping AC tasks onto multicore platforms that incorporate DVFS, it becomes possible to jointly optimize task-to-core assignments and V/F selections, enabling more optional cycles to be executed to improve QoS while adhering to constraints such as limited energy and real-time deadlines [8]. In addition, HMPs allow task migration (e.g., Arm big.LITTLE technology) and offer another lever to improve QoS through resource sharing. Task migration enables tasks to switch from one processor to another, enhancing energy efficiency and schedulability when processors differ in performance characteristics [1]. Several works have examined the migration of AC tasks on big.LITTLE systems [22]. Because big.LITTLE technology usually includes two processor clusters—*big* and *LITTLE*—where each cluster is homogeneous internally, and migrations typically occur only between clusters.

In contrast, this paper considers more general HMPs, where each processor may differ from the others, enabling task migration across multiple distinct processors. Representative platforms such as Arm® DynamIQ™ and MediaTek® CorePilot™ 3.0 SoCs integrate multiple heterogeneous CPU types and support task migration together with DVFS [5, 23]. Extending the concept of task migration from big.LITTLE systems to arbitrary HMPs is not straightforward, since multiple migrations could occur within a single task's lifetime. Consequently, both the number of migrations and the changes in execution cycles become mutually dependent optimization variables.

Some studies, such as [28], explore task migration in heterogeneous multicore architectures. However, these studies often rely on deterministic task models, in which tasks are assumed to have fixed execution times, and QoS is defined solely by whether deadlines are met. This approach, while ensuring timely execution, restricts the possibilities for improving QoS. Furthermore, the overheads of task migration (e.g., migration latency and additional energy usage) are often omitted in works such as [22, 31]. Nevertheless, in practical systems such as FreeRTOS™, migration incurs measurable delay and energy due to global scheduler locking, context save/restore, and cache/TLB refilling [14], which influence task migration decisions. To fully realize performance gains while avoiding excessive overhead, the task migration process must be thoroughly optimized.

Compared with existing work, we propose a joint optimization approach for mapping dependent real-time AC tasks to HMPs that incorporate DVFS and task migration, balancing conflicting demands for high system QoS under real-time and energy constraints. Our main contributions are as outlined:

(1) We formulate the AC real-time dependent task-mapping problem on HMPs supporting DVFS and task migration as a Mixed-Integer Non-Linear Programming (MINLP) problem, aiming to maximize QoS without violating real-time, energy, and dependency constraints. Specifically, we account for the time and energy overhead of task migration by modeling the topological changes in the task dependency DAG, which arise from the allocation and adjustment of AC tasks. Then, we linearize the nonlinear terms in the original problem, including products of binary and continuous variables and logical relationships, transforming it into a Mixed-Integer Linear Programming (MILP) problem that can be solved optimally.
(2) Although the MILP formulation can yield an exact solution, it is Non-deterministic Polynomial-time (NP)-hard, and its computational complexity grows quickly with the problem size. To address scalability, we develop a novel heuristic task-mapping framework. Following the principle of problem decomposition, we split the original problem into two subproblems and solve them sequentially via greedy algorithms. First, mandatory subtasks are mapped to balance execution time and energy consumption, then optional subtasks are allocated to maximize system QoS.
(3) We conduct extensive simulations to evaluate our proposed methods: an optimal method (OPT) and a heuristic method (HEU) in terms of QoS, computation time, task migration decisions, and feasibility. The simulation results demonstrate that, by employing task migration and DVFS, the OPT method achieves significantly higher QoS (about 238.9% on average and up to 411.5%) and greater feasibility (about 326.3% on average and up to 966.7%) compared to existing approaches. In addition, compared to the OPT method, the HEU approach reduces computation time by over a thousand times while still obtaining approximately 57.7% of the OPT QoS.

The remainder of this paper is organized and outlined such that Section 2 surveys related research. Section 3 introduces the system model and defines the task-mapping problem, and then Section 4 describes its linearization. Section 5 presents the low-complexity heuristic algorithm. Section 6 details the simulation results, and Section 7 concludes the paper.

2 Related Work

Table 1 summarizes several representative studies on task mapping in multicore platforms, covering various objectives (e.g., minimizing energy consumption or enhancing QoS), different task types (precise or approximate, dependent or independent), and a range of hardware platforms (homogeneous platforms or HMP). We further categorize the existing approaches by whether task migration is included in their design: no task migration (NOM), task migration excluding migration overhead (MEO), or task migration including migration overhead (MIO). In terms of methodology, task mapping problems are typically addressed using either heuristic or optimal methods.

2.1 Precise Computing Task Mapping

For precise computing tasks, many studies focus on minimizing energy consumption while respecting real-time constraints, often employing DVFS to reduce power [6, 16, 21, 25, 34]. For instance, [34] optimizes system configurations and scheduling parameters on a homogeneous multicore platform to lower energy usage while meeting task deadlines. Similarly, [16] proposes a scheduling strategy for real-time dependent tasks in battery-powered multicore systems under strict timing requirements. Other works consider heterogeneous platforms with DVFS to reduce energy consumption [6, 21, 25]. Specifically, [25] handles periodic real-time tasks with flexible deadlines, [6] exploits intra-task parallelism for energy-efficient scheduling of DAG-based real-time tasks, and [21] models and solves task-scheduling problems on three heterogeneous platforms to reduce

Table 1. Comparative Summary of Task Mapping Strategies in Multicore Systems.

Reference	Objective	Platform		DVFS	Dependency	AC	Task Migration			Solution	
		Homogeneous	Heterogeneous				NOM	MEO	MIO	Sub-optimal	Optimal
[34]	Energy	√		√	√	×	√			√	
[16]	Energy	√		×	√	×	√			√	√
[25]	Energy		√	√	√	×	√			√	
[6]	Energy		√	√	√	×	√			√	
[21]	Energy		√	√	√	×	√			√	
[31]	Workload Balance	√		√	×	×		√		√	
[33]	Response Time	√		×	√	×			√	√	
[28]	Migration Time		√	×	√	×			√	√	
[10]	Workload Balance	√		×	×	×			√	√	
[3]	Migration Time	√		×	×	×			√	√	
[26]	QoS	√		×	√	√	√			√	
[8]	QoS	√		√	√	√	√			√	√
[30]	QoS + Energy	√		×	√	√	√			√	√
[32]	QoS		√	√	√	√	√			√	
[24]	QoS		√	√	√	√		√		√	√
[22]	QoS		√	√	√	√		√		√	√
Proposed	QoS		√	√	√	√			√	√	√

energy usage. Compared to homogeneous platforms, heterogeneous systems often provide greater flexibility, leading to improved overall performance under time and energy constraints.

Although the aforementioned studies address precise task mapping on multicore platforms, they do not include task migration. In contrast, [3, 10, 28, 31, 33] incorporate migration to further boost flexibility and efficiency, aiming at objectives such as reducing response time [33], balancing workloads [10, 31], and handling migration overhead [3, 28]. For example, [31] uses task migration to balance workloads by minimizing average and peak processor temperatures, although it omits the time and energy overhead of migration. By contrast, [3, 10, 28, 33] explicitly consider the energy and/or time overheads of migrating dependent tasks across processors. In [10, 33], for example, the primary goal is to balance workloads, while [28] integrates a dynamic, probability-based migration mechanism along with lightweight migration timing analysis. In [3], a cluster-based scheduling technique for real-time mixed-criticality systems is proposed to reduce task migration and switch-time overheads.

2.2 Approximate Computing Task Mapping

AC task mapping typically focuses on maximizing system QoS under limited resources [8, 22, 24, 26, 30, 32]. In [8, 26, 30], the authors consider homogeneous multicore systems. [26] proposes using PDDL+ (Planning Domain Definition Language+) to transform the AC scheduling problem into a planning problem. Meanwhile, [8] and [30] explore hybrid offline-online approximate real-time scheduling. In [8], tasks are scheduled offline under time and dependency constraints, whereas the online phase manages energy through DVFS. In [30], tasks are scheduled under energy and dependency constraints for QoS optimization, with the online stage switching off selected caches to conserve energy. Studies on heterogeneous platforms, such as [32] and [24], also emphasize maximizing QoS under constraints. Specifically, [32] addresses uncertainty in energy harvesting by

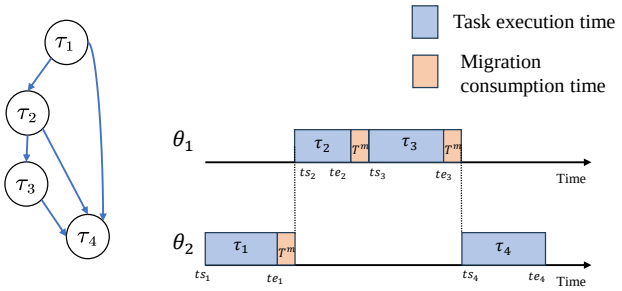


Fig. 1. An example of mapping dependent real-time tasks.

allocating AC tasks on multicore systems, and [24] applies approximate computation to enhance task execution quality under real-time and energy bounds.

None of the above approaches incorporates task migration. Although [22] discusses the mapping of AC tasks to HMPs with migration, the relationships among task migration, allocation, and scheduling variables are not fully explored. The task migration overheads are not optimized in [22]. Compared to these methods, our work provides a comprehensive solution to real-time AC task mapping on DVFS-enabled HMP under energy, time, and dependency constraints. Crucially, we incorporate task migration and explicitly account for migration overheads, which significantly affect system performance and feasibility.

3 System Model and Problem Formulation

3.1 System Model

3.1.1 *Task Model.*

Definition 3.1. *An AC task τ_i can be logically split into a mandatory subtask τ_i^M with M_i cycles and an optional subtask τ_i^O with o_i cycles. The mandatory subtask must finish before the deadline d_i to produce an acceptable result, whereas the optional subtask can be partially completed, at the expense of QoS. Here, M_i and o_i are measured in Worst-Case Execution Cycles (WCECs), where M_i is fixed and o_i is variable within $0 \le o_i \le O_i$, with O_i denoting the maximum allowable optional cycles.*

A real-time application with N dependent and periodic AC tasks $\mathcal{T} \triangleq \{\tau_1, \ldots, \tau_N\}$ can be represented as a Directed Acyclic Graph (DAG) $(\mathcal{V}, \mathcal{E})$, as illustrated in Fig. 1, where vertices in $\mathcal{V}$ denote tasks and directed edges in $\mathcal{E}$ reflect dependencies. A task τ_i may only begin execution once all of its predecessors have been completed. Upon finishing, τ_i immediately sends data to its successors. The dependency relations among tasks $\{\tau_1, \ldots, \tau_N\}$ can be captured by a matrix $\boldsymbol{s^0} \triangleq \left[s^0_{i,j}\right]_{N\times N}$. If $s^0_{i,j} = 1$, then τ_i is a predecessor of τ_j; otherwise, $s^0_{i,j} = 0$. This dependency matrix $\boldsymbol{s^0}$ is derived directly from the DAG structure.

Each task τ_i is characterized by a seven-element tuple $\{o_i, M_i, O_i, ts_i, te_i, d_i, g_i\}$, denoting the number of optional cycles, mandatory cycles, maximum optional cycles, start time, end time, deadline, and period, respectively. Define H as the *hyperperiod* of the task set, i.e., the least common multiple of all task periods $\{g_1, \ldots, g_N\}$ [9]. Hence, for mapping tasks, we set $te_i \le H$.

In imprecise computation, task QoS typically depends on how many optional cycles are executed [36, 37]. Denote $\mathcal{N} \triangleq \{1, \ldots, N\}$. Following common practice [37], the system QoS can be described as:

$$Q = Q_b + k \sum\nolimits_{i\in\mathcal{N}} o_i, \tag{1}$$

where Q_b is the basic QoS if all mandatory tasks are executed. $\sum_{i\in\mathcal{N}} o_i$ is the sum of cumulative executed optional cycles, and k is a constant.

If τ_i precedes τ_j (i.e., $s^0_{i,j} = 1$), then communication overhead (time and energy) arises between the processors to which τ_i and τ_j are assigned [22]. Denote T^m and E^m as the respective communication time and energy for each pair of dependent tasks placed on different processors. For instance, in Fig. 1, when tasks τ_1 and τ_2 execute on different processors θ_1 and θ_2, an additional communication time T^m is incurred. The same applies to the pairs (τ_2, τ_4) and (τ_3, τ_4).

3.1.2 Platform Model. We consider an HMP with M heterogeneous processors $\{\theta_1, \ldots, \theta_M\}$ that supports DVFS and task migration. Note that task migration occurs only between heterogeneous processors, e.g., for big.LITTLE platform, task migration occurs between the processors in big and LITTLE clusters. Let $\mathcal{M} \triangleq \{1, \ldots, M\}$. Each processor θ_k offers l_k discrete V/F levels $\{(v_{k,1}, f_{k,1}), \ldots, (v_{k,l_k}, f_{k,l_k})\}$, with $v_{k,l}$ and $f_{k,l}$ approximately linearly related [12].

Define $\lambda_{i,k} \in (0, 1]$ as the execution-efficiency factor of processor θ_k when running task τ_i, capturing hardware heterogeneity [21]. Hence, a task τ_i with $(M_i + o_i)$ total cycles takes $\frac{M_i+o_i}{\lambda_{i,k} f_{k,l}}$ units of time if assigned to processor θ_k at V/F level $(v_{k,l}, f_{k,l})$. Let $L = \max_{k\in\mathcal{M}} \{l_k\}$ be the maximum number of V/F levels among all M processors, and define $\mathcal{L} \triangleq \{1, \ldots, L\}$. We represent the V/F profiles of all processors with an $M \times L$ matrix:

$$VF = \begin{bmatrix} (v_{1,1}, f_{1,1}) & (v_{1,2}, f_{1,2}) & \cdots & (v_{1,L}, f_{1,L}) \\ (v_{2,1}, f_{2,1}) & (v_{2,2}, f_{2,2}) & \cdots & (v_{2,L}, f_{2,L}) \\ \vdots & \vdots & \ddots & \vdots \\ (v_{M,1}, f_{M,1}) & (v_{M,2}, f_{M,2}) & \cdots & (v_{M,L}, f_{M,L}) \end{bmatrix}.$$

For any l exceeding l_k, we have $v_{k,l} = v_{k,l_k}$ and $f_{k,l} = f_{k,l_k}$. To capture both task allocation and V/F settings, we introduce binary decision variables $\boldsymbol{c} \triangleq [c_{i,k,l}]_{N\times M\times L}$: if task τ_i is executed on processor θ_k at V/F level $(v_{k,l}, f_{k,l})$, then $c_{i,k,l} = 1$, otherwise $c_{i,k,l} = 0$. The resulting execution time of τ_i becomes:

$$\sum\nolimits_{k\in\mathcal{M}} \sum\nolimits_{l\in\mathcal{L}} \frac{c_{i,k,l}\,(M_i + o_i)}{\lambda_{i,k}\, f_{k,l}}.$$

A processor θ_k can be either *active* (busy executing) or *idle*. Mode-switching costs are comparatively small and thus often merged into task execution costs [11]. When running at V/F level $(v_{k,l}, f_{k,l})$, processor θ_k consumes power as denoted in (2):

$$P^c_{k,l} \;=\; P^s_{k,l} \;+\; P^d_{k,l} \;+\; P_{on}, \tag{2}$$

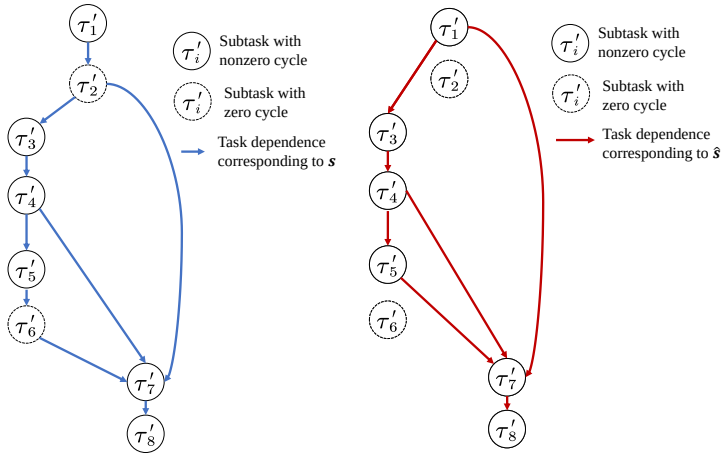
where $P^s_{k,l} = C^s_k\,(v_{k,l})^{\rho_k}$ is the static power of processor θ_k, $P^d_{k,l} = C^d_k\, f_{k,l}\,(v_{k,l})^2$ is the dynamic power during execution, and P_{on} is the inherent power consumed when the processor is powered on. The constants C^s_k, C^d_k, and ρ_k are hardware-specific. This power consumption model (2) is widely adopted [11, 22]. Consequently, the energy to execute task τ_i is

$$\sum\nolimits_{k\in\mathcal{M}} \sum\nolimits_{l\in\mathcal{L}} \frac{c_{i,k,l}\,(M_i + o_i)}{\lambda_{i,k}\, f_{k,l}}\, P^c_{k,l},$$

and the total execution energy over the hyperperiod H is

$$\sum\nolimits_{i\in N} \sum\nolimits_{k\in\mathcal{M}} \sum\nolimits_{l\in\mathcal{L}} \frac{c_{i,k,l}\,(M_i + o_i)}{\lambda_{i,k}\, f_{k,l}} \left(P^s_{k,l} + P^d_{k,l}\right) \;+\; MH\, P_{on}.$$

3.1.3 Task Splitting. We assume the platform allows task migration among the M heterogeneous processors. For the sake of problem formulation, each task is split into M subtasks at the beginning, i.e., each task can migrate among at most M processors, permitting up to $M-1$ migrations. Note that the actual number of subtasks may be fewer than M. This is because, after task splitting, we need to determine the execution cycles for each subtask by solving the task mapping problem. If there



(a) Example of subtask DAG. (b) Example of non-zero cycle subtask DAG.

Fig. 2. An example of the task dependency under splitting.

are zero-cycle subtasks, the migrations between these subtasks and their predecessors/successors do not exist, as they can be removed. Besides the execution cycles of each subtask, we also need to determine their allocations. If dependent subtasks are allocated and executed on the same processor in sequence, there is no task migration between them. Let $\mathcal{N}' = \{1, \ldots, M \times N\}$ be the set of all subtasks. For task τ_i, its M subtasks are denoted by $\{\tau'_{M\times(i-1)+1}, \ldots, \tau'_{M\times(i-1)+M}\}$. For example, Fig. 1 depicts a mapping of four dependent tasks $\{\tau_1, \ldots, \tau_4\}$ onto a two-processor system $\{\theta_1, \theta_2\}$. Under splitting, we obtain eight subtasks $\{\tau'_1, \ldots, \tau'_8\}$, forming the DAG in Fig. 2(a).

Analogous to $\boldsymbol{s}^{\mathbf{0}}$, we define the subtask dependency matrix $\boldsymbol{s} \triangleq [s_{i,j}]_{MN\times MN}$, where $s_{i,j} = 1$ indicates that τ'_i precedes τ'_j, and zero otherwise. Because τ_i is real-time, its completion time must satisfy $te_i \le d_i$. The last subtask of τ_i is $\tau'_{M\times(i-1)+M}$, whose deadline matches d_i. Other subtasks of τ_i have earlier finishing times determined by dependency constraints. To describe subtask-to-processor allocations, we introduce binary variables $\boldsymbol{x} = [x_{i,k}]_{MN\times M}$. If τ'_i is executed on processor θ_k, then $x_{i,k} = 1$; otherwise, $x_{i,k} = 0$.

Let $\varphi_{M\times(i-1)+t}$ be the execution-cycle count of subtask $\tau'_{M\times(i-1)+t}$, and let $\boldsymbol{\varphi} = [\varphi_i]_{MN\times 1}$. We introduce real variables $\mu_{M\times(i-1)+t} \in [0, 1]$ to represent the fractional portion of $(M_i + o_i)$ cycles assigned to subtask $\tau'_{M\times(i-1)+t}$:

$$\sum\nolimits_{t\in\mathcal{M}} \mu_{M\times(i-1)+t} \;=\; 1, \quad \forall i \in \mathcal{N}.$$

Hence, $\varphi_{M\times(i-1)+t} = \mu_{M\times(i-1)+t}\,(M_i + o_i)$, which allows distributing the total cycles $(M_i + o_i)$ among the M subtasks. Although $\varphi_{M\times(i-1)+t}$ is inherently integer-valued, we treat it as continuous to reduce problem complexity. After finding a solution, each $\varphi_{M\times(i-1)+t}$ can be rounded to the nearest integer. Because tasks usually require hundreds or thousands of cycles, this rounding introduces negligible errors in practice.

3.2 Problem Formulation

We now formally define the task-mapping problem under energy, time, and dependency constraints, incorporating task splitting, DVFS, and migration overhead. Along with the previously introduced

variables $c_{i,k,l}$, $x_{i,k}$, o_i, ts_i, te_i, φ_i, and μ_i, we introduce additional binary indicators to capture subtask existence, dependency, and migration. Let:

- $e_i \in \{0,1\}$ indicate whether subtask τ'_i has nonzero execution cycles ($e_i = 1$) or not ($e_i = 0$).
- $\hat{s}_{i,j} \in \{0,1\}$ represent the dependency between *nonzero-cycle* subtasks τ'_i and τ'_j.
- $\alpha_{i,j} \in \{0,1\}$ denote whether migration occurs between dependent subtasks τ'_i and τ'_j executed on different processors.
- $u_{i,j} \in \{0,1\}$ capture the execution order for two independent subtasks τ'_i and τ'_j that share the same processor.

Our objective is to maximize the system QoS. Note that Q_b and k in (1) are both constants, the objective function can be defined as $\sum_{i\in\mathcal{N}} o_i$. Subject to the constraints listed below, the entire formulation is expressed as the Primal Problem (PP):

$$PP: \max_{\boldsymbol{x},\boldsymbol{c},\boldsymbol{u},\boldsymbol{ts},\boldsymbol{te},\boldsymbol{o}} \sum\nolimits_{i\in\mathcal{N}} o_i, \tag{3}$$

$$s.t. \begin{cases} (4) - (22), \\ 0 \le o_i \le O_i, \quad \forall i \in \mathcal{N}, \\ 0 \le ts_i \le te_i \le H, \quad \forall i \in \mathcal{N}', \\ x_{i,k}, c_{i,k,l}, u_{i,j} \in \{0,1\}, \quad \forall i,j \in \mathcal{N}', \ \forall k \in \mathcal{M}, \ \forall l \in \mathcal{L}. \end{cases}$$

We detail each group of constraints below.

3.2.1 Task Assignment Constraints. Recall that each task τ_i can be split into M subtasks $\{\tau'_{M(i-1)+1}, \ldots, \tau'_{M(i-1)+M}\}$, and each subtask must execute on exactly one processor:

$$\sum\nolimits_{k\in\mathcal{M}} x_{i,k} = 1, \quad \forall i \in \mathcal{N}'. \tag{4}$$

3.2.2 Frequency Selection Constraints. With DVFS, each subtask τ'_i is bound to a single V/F level on the assigned processor. Hence,

$$\sum\nolimits_{k\in\mathcal{M}} \sum\nolimits_{l\in\mathcal{L}} c_{i,k,l} = 1, \quad \forall i \in \mathcal{N}'. \tag{5}$$

Since $c_{i,k,l} = 1$ implies $x_{i,k} = 1$, we have

$$x_{i,k} = \sum\nolimits_{l\in\mathcal{L}} c_{i,k,l}, \quad \forall i \in \mathcal{N}', \ \forall k \in \mathcal{M}. \tag{6}$$

3.2.3 Task Execution Cycle Constraints. Each subtask $\tau'_{M(i-1)+t}$ of task τ_i executes some portion $\varphi_{M(i-1)+t}$ of the total $M_i + O_i$ cycles, with

$$0 \le \varphi_{M(i-1)+t} \le M_i + O_i, \quad \forall i \in \mathcal{N}, \ \forall t \in \mathcal{M}, \tag{7}$$

and the sum of cycles for all M subtasks of τ_i must equal its total cycles:

$$\sum\nolimits_{t\in\mathcal{M}} \varphi_{M(i-1)+t} = M_i + o_i, \quad \forall i \in \mathcal{N}. \tag{8}$$

Since an AC task τ_i can vary from having only mandatory cycles M_i to a maximum of $M_i + O_i$, we also require:

$$M_i \le \sum\nolimits_{t\in\mathcal{M}} \varphi_{M(i-1)+t} \le M_i + O_i, \quad \forall i \in \mathcal{N}. \tag{9}$$

3.2.4 Task Existence Constraint. A subtask τ'_i is considered *existent* if its number of execution cycles is at least one. Let $e_i = 1$ if $\varphi_i \geq 1$, otherwise $e_i = 0$. Formally,

$$e_i = \begin{cases} 1, & \varphi_i \geq 1, \\ 0, & 0 \leq \varphi_i < 1, \end{cases} \qquad \forall i \in \mathcal{N}'. \tag{10}$$

These binary indicators simplify the modeling of migration and DAG dependencies when some subtasks have zero cycles.

3.2.5 Task Migration Constraints. In our formulation, a binary matrix $\hat{s} \triangleq [\hat{s}_{i,j}]_{MN \times MN}$ captures whether subtasks τ'_i and τ'_j are dependent after removing any subtasks with zero cycles. Its value depends jointly on the original dependency matrix $s_{i,j}$ and the subtask existence indicators e_i. Below, we show how to derive $\hat{s}_{i,j}$ so that it accurately reflects changes in the DAG once migration and zero-cycle subtasks are considered.

As an example, consider Fig. 2, where the execution-cycle counts of τ'_2 and τ'_6 are zero, while τ'_1, τ'_3, τ'_4, τ'_5, τ'_7, and τ'_8 each have nonzero cycles. Consequently, $\boldsymbol{e} = [1\,0\,1\,1\,1\,0\,1\,1]^T$. This means τ'_2 and τ'_6 are effectively removed from the DAG. We now detail how this removal updates the original subtask dependency $s_{i,j}$ into $\hat{s}_{i,j}$.

- **Identifying all paths in the subtask DAG:** We first enumerate all paths from *enter* subtasks (zero in-degree) to *exit* subtasks (zero out-degree), using a standard path search algorithm (e.g., depth-first search, DFS). Let $Path$ be the set of such paths, and W the total number of paths. In Fig. 2(a), τ'_1 is an entry subtask, τ'_8 is an exit subtask, and with $W = 3$:

$$Path = \left\{ (\tau'_1, \tau'_2, \tau'_3, \tau'_4, \tau'_5, \tau'_6, \tau'_7, \tau'_8),\ (\tau'_1, \tau'_2, \tau'_3, \tau'_4, \tau'_7, \tau'_8),\ (\tau'_1, \tau'_2, \tau'_7, \tau'_8) \right\}.$$

- **Marking paths and nonzero subtasks:** We define a binary (auxiliary) matrix $\boldsymbol{p} \triangleq [p_{i,n}]_{MN \times W}$, where $p_{i,n} = 1$ if subtask τ'_i lies in the n-th path; otherwise, $p_{i,n} = 0$. For instance, from Fig. 2(a), we obtain

$$\boldsymbol{p} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}^T.$$

  Next, we incorporate the subtask-existence variable e_i into $\boldsymbol{p}$ to form $\boldsymbol{pe} \triangleq [pe_{i,n}]$, using

$$pe_{i,n} = p_{i,n} \times e_i, \quad \forall i \in \mathcal{N}',\ \forall n \in \mathcal{W}. \tag{11}$$

  Hence, if τ'_i is in the n-th path and $\varphi_i > 0$, then $pe_{i,n} = 1$; otherwise, $pe_{i,n} = 0$. For the example in Fig. 2(b), we obtain

$$\boldsymbol{pe} = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}^T.$$

- **Forming an intermediate dependency variable $\mathring{s}_{i,j,n}$:** We still have an original subtask dependency matrix $\boldsymbol{s} \triangleq [s_{i,j}]_{MN \times MN}$, but subtasks that no longer exist (i.e., $\varphi_i = 0$) should be excluded. We define $\mathring{s}_{i,j,n}$, which indicates that τ'_i and τ'_j appear consecutively along the n-th path *and* both have nonzero cycles. Formally,

$$\mathring{s}_{i,j,n} = \begin{cases} 1, & \sum_{a=i}^{j} pe_{a,n} = 2,\ pe_{i,n} = 1,\ pe_{j,n} = 1, \\ 0, & \text{else.} \end{cases} \tag{12}$$

  Thus, $\mathring{s}_{i,j,n} = 1$ only if τ'_i and τ'_j are adjacent in path n and both exist (nonzero cycles).

- **Combining across multiple paths to form $\hat{s}_{i,j}$:** Since there could be multiple paths containing the same subtasks, τ'_i and τ'_j remain dependent if they lie on *any* path together. We use a logical "or" across all $n \in \mathcal{W}$:

$$\hat{s}_{i,j} = \bigvee\nolimits_{n\in\mathcal{W}} \mathring{s}_{i,j,n}, \quad \forall i, j \in \mathcal{N}'. \tag{13}$$

Hence, $\hat{s}_{i,j} = 1$ if and only if τ'_i and τ'_j are both nonzero subtasks on at least one path. Here, $\bigvee$ is the "or" operator, considering the number of paths. An illustrative example is shown in Fig. 2(b), yielding

$$\mathring{s}_{i,j,1} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

- **Defining Task Migration:** Once we have $\hat{s}_{i,j} = 1$, subtasks τ'_i and τ'_j are considered dependent in the pruned DAG. If they execute on different processors, a migration occurs. Let $\alpha_{i,j} = 1$ if subtasks τ'_i and τ'_j migrate, and 0 otherwise. We formulate:

$$\begin{aligned} \alpha_{i,j} &= \frac{\hat{s}_{i,j}}{2} \sum\nolimits_{k\in\mathcal{M}} \left(x_{i,k} + x_{j,k} - 2x_{i,k}x_{j,k}\right) \\ &= \frac{1}{2} \sum\nolimits_{k\in\mathcal{M}} \left(\hat{s}_{i,j}x_{i,k} + \hat{s}_{i,j}x_{j,k} - 2\hat{s}_{i,j}x_{i,k}x_{j,k}\right), \quad \forall i, j \in \mathcal{N}'. \end{aligned} \tag{14}$$

If τ'_i and τ'_j are mapped to the same processor, $\alpha_{i,j} = 0$; otherwise, $\alpha_{i,j} = 1$. We formally state this in *Lemma* 3.1.

Lemma 3.1. *Assume subtasks τ'_i and τ'_j both have nonzero cycles and $\hat{s}_{i,j} = 1$ (i.e., they remain dependent in the updated DAG). If τ'_i and τ'_j are assigned to different processors, $\alpha_{i,j} = 1$; otherwise, $\alpha_{i,j} = 0$.*

Proof. Let $\beta_{i,j,k} = x_{i,k} + x_{j,k} - 2\,x_{i,k}x_{j,k}$ for each processor θ_k, we have two cases:

- *Same-processor case*: If τ'_i and τ'_j both run on θ_{k_1}, then $x_{i,k_1} = x_{j,k_1} = 1$, implying $\beta_{i,j,k_1} = 0$, and $\beta_{i,j,k} = 0$ for all $k \neq k_1$. Therefore, $\sum_{k\in\mathcal{M}} \beta_{i,j,k} = 0$, so $\alpha_{i,j} = 0$.
- *Different-processor case*: Suppose τ'_i is on θ_{k_1} and τ'_j on $\theta_{k_2} \neq k_1$. Then $\beta_{i,j,k_1} = 1$ and $\beta_{i,j,k_2} = 1$, with $\beta_{i,j,k} = 0$ otherwise. Hence, $\sum_{k\in\mathcal{M}} \beta_{i,j,k} = 2$, leading to $\alpha_{i,j} = 1$.

□

3.2.6 Task Non-preemption Constraints. For two *independent* subtasks τ'_i and τ'_j (i.e., $s_{i,j} = 0$) assigned to the same processor θ_k, we must ensure they do not overlap in time. Introducing binary variables $u_{i,j}$ to indicate which subtask executes first, the following constraints enforce non-overlapping execution:

$$\widehat{te}_i \leq ts_j + \left(2 - x_{i,k} - x_{j,k}\right) H + (1 - u_{i,j})H, \tag{15}$$

$$\widehat{te}_j \leq ts_i + \left(2 - x_{i,k} - x_{j,k}\right) H + u_{i,j}H, \tag{16}$$

for any $i \neq j \in \mathcal{N}'$ and $k \in \mathcal{M}$. Here, H is a sufficiently large value (the hyperperiod) ensuring that if τ'_i and τ'_j lie on different processors, the constraints are trivially satisfied.

3.2.7 Subtask End Time and Dependency. The end time of a subtask τ'_i is

$$te_i \;=\; ts_i \;+\; \sum_{k\in\mathcal{M}} \sum_{l\in\mathcal{L}} \frac{c_{i,k,l}\,\varphi_i}{\lambda_{i,k}\,f_{k,l}}, \quad \forall i \in \mathcal{N}'. \tag{17}$$

If one or more migrations occur after τ'_i, the subtask completion must also include the total migration time:

$$\widehat{te}_i \;=\; te_i \;+\; T^m \sum_{j\in\mathcal{N}'} \alpha_{i,j}, \quad \forall i \in \mathcal{N}'. \tag{18}$$

For *dependent* subtasks τ'_i and τ'_j (i.e., $s_{i,j} = 1$), τ'_i must finish before τ'_j can start:

$$\widehat{te}_i \;\le\; ts_j + (1 - s_{i,j})\,H, \quad \forall i \neq j \in \mathcal{N}'. \tag{19}$$

When $s_{i,j} = 1$, this simplifies to $\widehat{te}_i \le ts_j$; otherwise, it is trivially satisfied.

3.2.8 Real-time Constraints. Since each task τ_i must complete by its deadline d_i, all subtasks of τ_i share the same global deadline:

$$te_{M\times(i-1)+t} \;\le\; d_i, \quad \forall i \in \mathcal{N},\ \forall t \in \mathcal{M}. \tag{20}$$

3.2.9 Energy Consumption Constraint. When subtask τ'_i executes φ_i cycles on processor θ_k at V/F level $(v_{k,l}, f_{k,l})$, the total execution time for that level is:

$$t^d_{k,l} \;=\; \sum_{i\in\mathcal{N}'} \frac{c_{i,k,l}\,\varphi_i}{\lambda_{i,k}\,f_{k,l}}, \quad \forall k \in \mathcal{M},\ \forall l \in \mathcal{L}. \tag{21}$$

The processor consumes $\left(P^s_{k,l} + P^d_{k,l}\right)$ power at this level, plus the inherent P_{on} when active. Summing across levels and processors yields the total task execution energy. In addition, each migration costs E^m energy. Defining E_t as the total system energy, and E_s as the available energy supply in the hyperperiod H, the constraint is:

$$E_t \;=\; \underbrace{E^m \sum_{i\in\mathcal{N}'} \sum_{j\in\mathcal{N}'} \alpha_{i,j}}_{\text{task migration energy}} \;+\; \underbrace{\sum_{k\in\mathcal{M}} \sum_{l\in\mathcal{L}} \left(P^s_{k,l} + P^d_{k,l}\right) t^d_{k,l} + MHP_{on}}_{\text{task execution energy}} \;\le\; E_s. \tag{22}$$

Together, these constraints define a comprehensive MINLP framework that captures task splitting, DVFS, migration overhead, and subtask scheduling. The objective is to maximize the sum of executed optional cycles while satisfying real-time and energy constraints.

3.2.10 Discussions. Note that the migration overheads T^m and E^m are uniform in the above formulations (e.g., (18) and (22)). Our model can be extended to distinct overheads. We assume that when task migration occurs between subtasks τ'_i and τ'_j, the amount of data transferred from τ'_i to τ'_j is $D_{i,j}$, and let $T^{unit}_{k_1,k_2}$ and $E^{unit}_{k_1,k_2}$ denote the time and the energy required to transfer one unit of task data from processor θ_{k_1} to processor θ_{k_2}, respectively. Therefore, the migration time and energy overheads from τ'_i (executed on θ_{k_1}) to τ'_j (executed on θ_{k_2}) are $T^m_{i,j,k_1,k_2} = D_{i,j} \times T^{unit}_{k_1,k_2}$ and $E^m_{i,j,k_1,k_2} = D_{i,j} \times E^{unit}_{k_1,k_2}$, respectively. T^m_{i,j,k_1,k_2} and E^m_{i,j,k_1,k_2} are constants, under the given $D_{i,j}$, $T^{unit}_{k_1,k_2}$, and $E^{unit}_{k_1,k_2}$.

Note that $x_{i,k}$ is the task allocation variable, while $\alpha_{i,j}$ indicates whether there is a migration from τ'_i to τ'_j. We introduce an auxiliary (binary) variable x^m_{i,j,k_1,k_2}, which is set to 1 if τ'_i and τ'_j are assigned to θ_{k_1} and θ_{k_2}, respectively, and task migration occurs from τ'_i to τ'_j. Thus, we have $x^m_{i,j,k_1,k_2} = x_{i,k_1} \wedge x_{j,k_2} \wedge \alpha_{i,j}$, where $\wedge$ is the logical "and" operation, which can be handled by the linear method in Section 4.3. With x^m_{i,j,k_1,k_2}, T^m_{i,j,k_1,k_2}, and E^m_{i,j,k_1,k_2}, the end time $\widehat{te}_i$ of τ'_i and the total system energy E_t can be updated as follows:

$$\widehat{te}_i = te_i + \sum_{j\in\mathcal{N}'} \sum_{k_1\in\mathcal{M}} \sum_{k_2\in\mathcal{M}} x^m_{i,j,k_1,k_2}\, T^m_{i,j,k_1,k_2}, \quad \forall i \in \mathcal{N}'.$$

$$E_t = \underbrace{\sum_{i\in\mathcal{N}'}\sum_{j\in\mathcal{N}'}\sum_{k_1\in\mathcal{M}}\sum_{k_2\in\mathcal{M}} x^m_{i,j,k_1,k_2}\, E^m_{i,j,k_1,k_2}}_{\text{task migration energy}} + \underbrace{\sum_{k\in\mathcal{M}}\sum_{l\in\mathcal{L}}\left(P^s_{k,l}+P^d_{k,l}\right)t^d_{k,l}+MHP_{on}}_{\text{task execution energy}}.$$

4 Problem Linearization

The original optimization problem in (3) is formulated as a MINLP model, primarily due to products of continuous and binary variables (e.g., $c_{i,k,l}\varphi_i$) and logical *if-else* relationships (e.g., (10)). Such nonlinearities make the problem challenging to solve directly with common solvers such as Gurobi or CPLEX. In this section, we present a linearization strategy that transforms the problem into an equivalent MILP model that can be optimally solved, and the optimal solution can be used to evaluate the performance of the proposed heuristic method.

4.1 Nonlinear Terms from the Product of Continuous and Binary Variables

Recall that $c_{i,k,l} \in \{0,1\}$ specifies the processor V/F assignment for subtask τ'_i, whereas $\varphi_i \in [0, M_i + O_i]$ is the number of cycles that subtask τ'_i will actually execute. The product $c_{i,k,l}\varphi_i$ appears in constraints such as (17) (subtask completion time) and (21) (processor-level execution time). To handle this product, we introduce an auxiliary continuous variable $\Phi_{i,k,l}$ to replace $c_{i,k,l}\varphi_i$ and impose the following linear constraints:

$$\begin{aligned} c_{i,k,l}\varphi_{i,\min} \le \Phi_{i,k,l} \le c_{i,k,l}\varphi_{i,\max}, \quad \Phi_{i,k,l} - c_{i,k,l}\varphi_{i,\min} - \varphi_i + \varphi_{i,\min} \le 0, \\ \Phi_{i,k,l} - c_{i,k,l}\varphi_{i,\max} - \varphi_i + \varphi_{i,\max} \ge 0, \end{aligned} \tag{23}$$

where the constraints hold for all $i \in \mathcal{N}'$, $k \in \mathcal{M}$, and $l \in \mathcal{L}$, with $\varphi_{i,\min} = 0$ and $\varphi_{i,\max} = M_i + O_i$ (i.e., the possible range of cycles for each subtask).

We then replace every occurrence of $c_{i,k,l}\varphi_i$ with $\Phi_{i,k,l}$ in the relevant constraints. For example, (17) and (21) become

$$te_i = ts_i + \sum_{k\in\mathcal{M}}\sum_{l\in\mathcal{L}} \frac{\Phi_{i,k,l}}{\lambda_{i,k} f_{k,l}}, \quad \forall i \in \mathcal{N}', \tag{24}$$

$$t^d_{k,l} = \sum_{i\in\mathcal{N}'} \frac{\Phi_{i,k,l}}{\lambda_{i,k} f_{k,l}}, \quad \forall k \in \mathcal{M},\ \forall l \in \mathcal{L}. \tag{25}$$

Because constraints (23) are linear, these substitutions eliminate the continuous-binary product terms from the model.

4.2 Nonlinear Terms from the Product of Binary Variables

Another source of nonlinearity is the product of two (or more) binary variables, such as $p_{i,n}e_i$ in constraint (11), and $x_{i,k}x_{j,k}$ or $\hat{s}_{i,j}x_{i,k}$ in migration-related constraint (14). To linearize these products, we apply a standard transformation:

Lemma 4.1 (Binary Product Linearization). *If $x_1, x_2 \in \{0,1\}$, then the product $y = x_1x_2$ can be replaced by an auxiliary binary variable $y \in \{0,1\}$ together with the constraints*

$$y \le x_1, \quad y \le x_2, \quad y \ge x_1 + x_2 - 1.$$

Using this technique, each product of binary variables (e.g., $\hat{s}_{i,j}\, x_{i,k}$ or $x_{i,k}x_{j,k}$) is replaced by an auxiliary binary variable. As an example, consider $\alpha_{i,j}$ from (14), which depends on $\hat{s}_{i,j}x_{i,k}$, $\hat{s}_{i,j}x_{j,k}$, and $\hat{s}_{i,j}x_{i,k}x_{j,k}$. We define:

$$\hat{s}x^1_{i,j,k} = \hat{s}_{i,j}x_{i,k}, \quad \hat{s}x^2_{i,j,k} = \hat{s}_{i,j}x_{j,k}, \quad X_{i,j,k} = x_{i,k}x_{j,k}, \quad \hat{s}X_{i,j,k} = \hat{s}_{i,j}X_{i,j,k}. \tag{26}$$

Each of these is then constrained by linear inequalities as per *Lemma* 4.1, enabling us to rewrite (14) in linear form as in (27):

$$\alpha_{i,j} \;=\; \frac{1}{2}\sum\nolimits_{k\in\mathcal{M}}\left(\hat{s}x^1_{i,j,k}+\hat{s}x^2_{i,j,k} \;-\; 2\,\hat{s}X_{i,j,k}\right), \quad \forall i,j\in\mathcal{N}',\ \forall k\in\mathcal{M}. \tag{27}$$

Similarly, we have $pe_{i,n} = p_{i,n}\times e_i$. On this basis, we add the following constraints ($\forall i,j\in\mathcal{N}',\ \forall k\in\mathcal{M},\ \forall n\in\mathcal{W}$) into (3):

$$pe_{i,n} \;\le\; p_{i,n}, \quad pe_{i,n} \;\le\; e_i, \quad pe_{i,n} \;\ge\; p_{i,n}+e_i-1, \tag{28}$$

$$\hat{s}x^1_{i,j,k} \;\le\; \hat{s}_{i,j}, \quad \hat{s}x^1_{i,j,k} \;\le\; x_{i,k}, \quad \hat{s}x^1_{i,j,k} \;\ge\; \hat{s}_{i,j}+x_{i,k}-1, \tag{29}$$

$$\hat{s}x^2_{i,j,k} \;\le\; \hat{s}_{i,j}, \quad \hat{s}x^2_{i,j,k} \;\le\; x_{j,k}, \quad \hat{s}x^2_{i,j,k} \;\ge\; \hat{s}_{i,j}+x_{j,k}-1, \tag{30}$$

$$X_{i,j,k} \;\le\; x_{i,k}, \quad X_{i,j,k} \;\le\; x_{j,k}, \quad X_{i,j,k} \;\ge\; x_{i,k}+x_{j,k}-1, \tag{31}$$

$$\hat{s}X_{i,j,k} \;\le\; \hat{s}_{i,j}, \quad \hat{s}X_{i,j,k} \;\le\; X_{i,j,k}, \quad \hat{s}X_{i,j,k} \;\ge\; \hat{s}_{i,j}+X_{i,j,k}-1. \tag{32}$$

4.3 Nonlinearities from Logical Relationships

So far, we have discussed nonlinearities arising from products of continuous and binary variables, as well as from products of two or more binary variables. A third source of nonlinearity arises from logical *if-else* comparisons and logical operators ("and" / "or") present in constraints such as (10), (12), and (13). None of these is a standard linear expression. In what follows, we introduce *Lemma* 4.2–*Lemma* 4.5 to linearize such logical constructs.

Firstly, consider the logical "or" operation in constraint (13), where

$$\hat{s}_{i,j} \;=\; \bigvee\nolimits_{n\in\mathcal{W}} \mathring{s}_{i,j,n}.$$

We employ *Lemma* 4.2 to transform this "or" into linear form:

Lemma 4.2. *Suppose $x_1,\ldots,x_n \in \{0,1\}$ are n binary variables. Then the term $\bigvee_{k=1}^{n} x_k$ can be replaced by an auxiliary binary variable y, subject to*

$$y \;\ge\; x_k, \quad k \;=\; 1,\ldots,n, \quad \textit{and} \quad y \;\le\; \sum\nolimits_{k=1}^{n} x_k.$$

Proof. If $x_1 = \cdots = x_m = 1$ for some m with $1 \le m \le n$, then we must have $y \ge 1$ and $y \le m$. Since $y \in \{0,1\}$, $y = 1$. Conversely, if $x_1 = \cdots = x_m = 0$, then $y \ge 0$ and $y \le 0$, i.e., $y = 0$. □

Consequently, (13) can be linearized via:

$$\hat{s}_{i,j} \ge \mathring{s}_{i,j,n}, \quad \hat{s}_{i,j} \le \sum\nolimits_{n\in\mathcal{W}} \mathring{s}_{i,j,n}, \quad \forall i,j\in\mathcal{N}',\ \forall n\in\mathcal{W}. \tag{33}$$

Secondly, constraint (10) states that $e_i = 1$ if and only if $\varphi_i \ge 1$, and $e_i = 0$ otherwise. We convert this *if-else* relationship to linear constraints using *Lemma* 4.3:

Lemma 4.3. *Let x be a continuous variable, a be a binary variable, and $f(x)$ be a function of x. The logical expression*

$$\textit{if } f(x) \;\ge\; 0, \quad \textit{then } a \;=\; 1; \quad \textit{else } a \;=\; 0$$

can be encoded as:

$$\begin{gathered} f(x) \;\le\; M(1-z_1)-\epsilon, \quad -(1-z_1) \;\le\; a \;\le\; (1-z_1), \\ f(x) \;\ge\; M(1-z_2), \quad -(1-z_2) \;\le\; a-1 \;\le\; (1-z_2), \quad z_1+z_2 \;=\; 1, \end{gathered} \tag{34}$$

where M is sufficiently large, ϵ is a small positive constant, and z_1, z_2 are auxiliary binary variables.

Proof. Because $z_1 + z_2 = 1$ and both are binary, if $z_1 = 1$ then $z_2 = 0$, and vice versa. In the first case, $z_1 = 1$ forces $f(x) \le -\epsilon$ and $0 \le a \le 0$, implying $f(x) < 0$ and $a = 0$. In the second case, $z_1 = 0$, $z_2 = 1$ leads to $f(x) \ge 0$ and $a = 1$. Detailed inequalities appear directly in (34). □

Applying this to $f(x) = \varphi_i - 1$ with constants $M_1,\ \epsilon_1 > 0$ and two new binary variables z_i^1, z_i^2, we linearize (10) as:

$$\begin{aligned} \varphi_i - 1 \ \le\ M_1(1 - z_i^1) - \epsilon_1, \quad -(1 - z_i^1) \ \le\ e_i \ \le\ (1 - z_i^1), \\ \varphi_i - 1 \ \ge\ M_1(1 - z_i^2), \quad -(1 - z_i^2) \ \le\ e_i - 1 \ \le\ (1 - z_i^2), \quad z_i^1 + z_i^2 \ =\ 1, \quad \forall i \in \mathcal{N}'. \end{aligned} \tag{35}$$

Finally, in constraint (12), if $\sum_{a=i}^{j} pe_{a,n} = 2$ *and* $pe_{i,n} = pe_{j,n} = 1$, we set $\mathring{s}_{i,j,n} = 1$, else, $\mathring{s}_{i,j,n} = 0$. Because (12) involves the logical "and" and an *if-else* condition, we invoke *Lemma* 4.4 (for "and") and *Lemma* 4.5 (for the zero-equality check).

Lemma 4.4. *Let $x_1, \ldots, x_n \in \{0, 1\}$. The logical expression $\bigwedge_{k=1}^{n} x_k$ can be replaced by an auxiliary binary variable y satisfying*

$$y \ \le\ x_k, \quad k = 1, \ldots, n, \quad \text{and} \quad y \ \ge\ \left(\sum\nolimits_{k=1}^{n} x_k\right) \ -\ (n - 1).$$

Proof. If any $x_k = 0$, then $y \le 0$ and $y \ge (n-1) - (n-1) = 0$, hence $y = 0$. If $x_1 = \cdots = x_n = 1$, then $y \le 1$ and $y \ge n - (n-1) = 1$, i.e., $y = 1$. □

In constraint (12), $pe_{i,n}$ and $pe_{j,n}$ are binary, but $\sum_{a=i}^{j} pe_{a,n}$ is an integer. Hence, we first introduce a binary $Spe_{i,j,n}$ to indicate whether $\sum_{a=i}^{j} pe_{a,n} = 2$. To linearize this *if-else* comparison, we apply *Lemma* 4.5.

Lemma 4.5. *Let x be continuous, a be binary, and $f(x)$ a function of x. The condition*

$$\text{if } f(x) \ =\ 0, \quad \text{then } a \ =\ 1; \quad \text{else } a \ =\ 0$$

can be expressed via additional constraints:

$$\begin{aligned} f(x) \ \le\ M(1 - z_1) - \epsilon, \quad -(1 - z_1) \ \le\ a \ \le\ (1 - z_1), \\ -M(1 - z_2) - \epsilon \ \le\ f(x) \ \le\ \epsilon + M(1 - z_2), \quad -(1 - z_2) \ \le\ a - 1 \ \le\ (1 - z_2), \\ f(x) \ \ge\ -M(1 - z_3) + \epsilon, \quad -(1 - z_3) \ \le\ a \ \le\ (1 - z_3), \quad z_1 + z_2 + z_3 \ =\ 1, \end{aligned} \tag{36}$$

where M is large, ϵ is small, and z_1, z_2, z_3 are auxilary (binary).

Proof. We want to enforce: if $f(x) = 0$, then $a = 1$; otherwise, $a = 0$. Equivalently, a) $f(x) < 0 \Rightarrow a = 0$, b) $f(x) = 0 \Rightarrow a = 1$, and c) $f(x) > 0 \Rightarrow a = 0$. Because $z_1, z_2, z_3 \in \{0, 1\}$ with $z_1 + z_2 + z_3 = 1$, exactly one of the three cases follows holds:

- $z_1 = 1, z_2 = z_3 = 0$. From (36), $f(x) \le -\epsilon$ and $0 \le a \le 0$ imply $f(x) < 0 \Rightarrow a = 0$.
- $z_2 = 1, z_1 = z_3 = 0$. Here, $-\epsilon \le f(x) \le \epsilon$ and $0 \le a - 1 \le 0$ force $f(x) = 0 \Rightarrow a = 1$.
- $z_3 = 1, z_1 = z_2 = 0$. We have $f(x) \ge \epsilon$ and $0 \le a \le 0$, therefore $f(x) > 0 \Rightarrow a = 0$.

These three cases cover all possibilities for $f(x)$ being negative, zero, or positive, ensuring that (36) correctly encodes the original condition.

□

Applying this to $f(x) = -2 + \sum_{a=i}^{j} pe_{a,n}$, we obtain the linear form:

$$\begin{aligned} \sum\nolimits_{a=i}^{j} pe_{a,n} - 2 \ \le\ M_2(1 - z_{i,j,n}^3) - \epsilon_2, \quad -(1 - z_{i,j,n}^3) \ \le\ Spe_{i,j,n} \ \le\ (1 - z_{i,j,n}^3), \\ -M_2(1 - z_{i,j,n}^4) - \epsilon_2 \ \le\ \sum\nolimits_{a=i}^{j} pe_{a,n} - 2 \ \le\ \epsilon_2 + M_2(1 - z_{i,j,n}^4), \\ -(1 - z_{i,j,n}^4) \ \le\ Spe_{i,j,n} - 1 \ \le\ (1 - z_{i,j,n}^4), \\ \sum\nolimits_{a=i}^{j} pe_{a,n} - 2 \ \ge\ -M_2(1 - z_{i,j,n}^5) + \epsilon_2, \quad -(1 - z_{i,j,n}^5) \ \le\ Spe_{i,j,n} \ \le\ (1 - z_{i,j,n}^5), \\ z_{i,j,n}^3 + z_{i,j,n}^4 + z_{i,j,n}^5 \ =\ 1, \quad \forall i \le j \in \mathcal{N}',\ \forall n \in \mathcal{W}, \end{aligned} \tag{37}$$

where M_2 is a positive constant large enough, ϵ_2 is a positive constant small enough, and $z^3_{i,j,n}$, $z^4_{i,j,n}$, and $z^5_{i,j,n}$ are the binary variables.

Next, we incorporate the "and" operation $Spe_{i,j,n} \wedge pe_{i,n} \wedge pe_{j,n}$ using *Lemma* 4.4. We define a new binary variable $y^{and}_{i,j,n}$:

$$\begin{aligned} y^{and}_{i,j,n} \;\le\; Spe_{i,j,n}, \quad y^{and}_{i,j,n} \;\le\; pe_{i,n}, \quad y^{and}_{i,j,n} \;\le\; pe_{j,n}, \\ y^{and}_{i,j,n} \;\ge\; Spe_{i,j,n} + pe_{i,n} + pe_{j,n} - 2, \quad \forall i \le j \in \mathcal{N}', \; \forall n \in \mathcal{W}. \end{aligned} \tag{38}$$

With $Spe_{i,j,n}$ and $y^{and}_{i,j,n}$ now linearized, the original condition "if $y^{and}_{i,j,n} = 1$, then $\mathring{s}_{i,j,n} = 1$, else, $\mathring{s}_{i,j,n} = 0$" is itself another *if-else*. Let $f(x) = y^{and}_{i,j,n} - 1$, and apply *Lemma* 4.5 again to obtain (39):

$$\begin{aligned} y^{and}_{i,j,n} - 1 \;\le\; M_2(1 - z^6_{i,j,n}) - \epsilon_2, \quad -(1 - z^6_{i,j,n}) \;\le\; \mathring{s}_{i,j,n} \;\le\; (1 - z^6_{i,j,n}), \\ -M_2(1 - z^7_{i,j,n}) - \epsilon_2 \;\le\; y^{and}_{i,j,n} - 1 \;\le\; \epsilon_2 + M_2(1 - z^7_{i,j,n}), \\ -(1 - z^7_{i,j,n}) \;\le\; \mathring{s}_{i,j,n} - 1 \;\le\; (1 - z^7_{i,j,n}), \\ y^{and}_{i,j,n} - 1 \;\ge\; -M_2(1 - z^8_{i,j,n}) + \epsilon_2, \quad -(1 - z^8_{i,j,n}) \;\le\; \mathring{s}_{i,j,n} \;\le\; (1 - z^8_{i,j,n}), \\ z^6_{i,j,n} + z^7_{i,j,n} + z^8_{i,j,n} \;=\; 1, \quad \forall i \le j \in \mathcal{N}', \; \forall n \in \mathcal{W}, \end{aligned} \tag{39}$$

where $M_3 > 0$ is large, $\epsilon_3 > 0$ is small, and $z^6_{i,j,n}$, $z^7_{i,j,n}$, $z^8_{i,j,n}$ are binary. Taken together, these steps fully linearize the logical relationships in (12) and thereby remove all "and" and *if-else* nonlinearities.

4.4 Resulting MILP Formulation

Applying the above linearization steps to all nonlinear items in the original MINLP (3) yields a fully linear model. In particular, we replace:

- $c_{i,k,l}\,\varphi_i$ with $\Phi_{i,k,l}$ and add constraints (23)–(25),
- products of binary variables (e.g., $\hat{s}_{i,j}x_{i,k}$, $x_{i,k}x_{j,k}$, and $p_{i,n}e_i$) with auxiliary variables and *Lemma* 4.1 constraints.
- logical relationships (e.g., (10), (12), and (13)) with *Lemma* 4.2 – *Lemma* 4.5 constraints.

The resulting MILP problem is structurally equivalent to (3) but expressed in linear form:

$$\max_{\boldsymbol{var}_1, \boldsymbol{var}_2} \sum\nolimits_{i \in \mathcal{N}} o_i, \tag{40}$$

$$\text{s.t.} \begin{cases} \text{Original linear constraints (4)-(9), (18)-(19), (22),} \\ \text{Plus auxiliary constraints (23)-(33), (35), (37)-(39),} \\ 0 \;\le\; o_i \;\le\; O_i, \quad \forall i \in \mathcal{N}, \quad 0 \;\le\; ts_i \;\le\; H, \quad \forall i \in \mathcal{N}', \\ \boldsymbol{var}_1 \in \{0,1\}, \quad \boldsymbol{var}_2 \in \mathbb{R}^+. \end{cases}$$

Here, $\boldsymbol{var}_1$ collects all binary decision variables (such as $x_{i,k}$, $c_{i,k,l}$, e_i, $\alpha_{i,j}$, and the auxiliary variables introduced for binary products), while $\boldsymbol{var}_2$ denotes the continuous variables (such as φ_i, o_i, $\Phi_{i,k,l}$, ts_i, and te_i). Off-the-shelf solvers can then solve this MILP to obtain the optimal solution under the same constraints as the original MINLP.

5 Heuristic Task Mapping Algorithm

Although the MINLP-based Primal Problem (3) can be linearized to the MILP-based problem (40) and solved optimally by state-of-the-art solvers (e.g., CPLEX or Gurobi), finding such an exact solution becomes increasingly time-consuming as the problem size grows. Hence, we propose a heuristic task mapping algorithm that employs a two-step strategy to balance solution quality and runtime. The details of our proposed heuristic are presented in the following subsections.

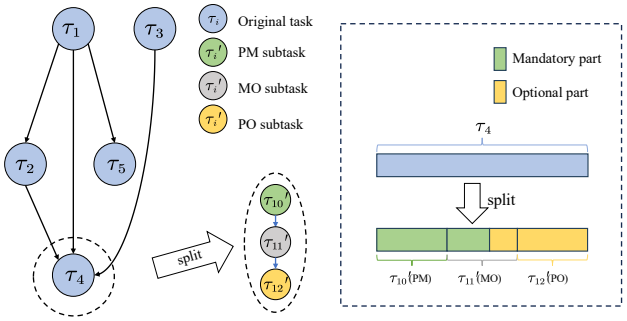


Fig. 3. An example of original task splitting.

5.1 Initialization: Task Splitting and Subtask Layering

5.1.1 Task Splitting. To handle task migration in a platform with M heterogeneous processors, we first partition each original task τ_i into M subtasks, similar to the method described in Section 3.1.3. Recall that the maximum execution cycle of τ_i is $M_i + O_i$. We split these cycles equally among the M subtasks so that each subtask $\tau'_{M\times(i-1)+t}$ (for $t \in \mathcal{M}$) has

$$\varphi^H_{M\times(i-1)+t} = \frac{M_i + O_i}{M}, \quad \forall i \in \mathcal{N},\ \forall t \in \mathcal{M}.$$

After splitting, each task τ_i yields three possible types of subtasks:

- **Purely Mandatory (PM) subtask**: Contains only mandatory cycles.
- **Mixed Mandatory-Optional (MO) subtask**: Contains both mandatory and optional cycles.
- **Purely Optional (PO) subtask**: Contains only optional cycles.

Among the M subtasks of a task τ_i, there can be multiple PM or PO subtasks but *at most one* MO subtask. For instance, Fig. 3 shows a scenario with $M = 3$ processors, where the subtasks τ'_{10}, τ'_{11}, and τ'_{12} for task τ_4 are identified as PM, MO, and PO, respectively. All three subtasks have $\frac{M_4+O_4}{M}$ cycles, but their composition of mandatory and optional parts differs.

Since all mandatory cycles must be finished before the deadline, our heuristic approach splits the scheduling into two steps:

- Schedule the PM and MO subtasks so that every mandatory cycle is mapped and remains feasible.
- Schedule the remaining PO subtasks (if time and energy permit) to increase the overall QoS.

Splitting each task into M subtasks in the heuristic is an initialization cap that provides up to M mapping slots; it does not force execution across all M processors. In Step 2, PO cycles are decision variables (possibly zero), and placement decisions jointly consider migration time/energy, DVFS, and timing feasibility.

5.1.2 Subtask Layering. After splitting, each original task DAG (with adjacency matrix $\boldsymbol{s}^0$) transforms into a subtask DAG (with adjacency matrix $\boldsymbol{s}$). In this subtask DAG, a subtask can start only after all its predecessors have finished. To systematically assign subtasks to processors, we layer the DAG based on the matrix $\boldsymbol{s}$, as follows:

- An *entry* subtask (no predecessors) has a layer index $LC_i = 1$.
- Any other subtask τ'_i has $LC_i = 1 + \max_{\tau'_j \in Pre^s_i}\{LC_j\}$, where Pre^s_i is the set of all predecessors of τ'_i.

By sorting subtasks in ascending order of LC_i, and breaking ties by subtask index, we get a valid allocation sequence Seq_s. Fig. 4 depicts an example, where $Seq_s = \{1, 7, 2, 8, 3, 9, 4, 13, 5, 14, 6, 15, 10, 11, 12\}$.

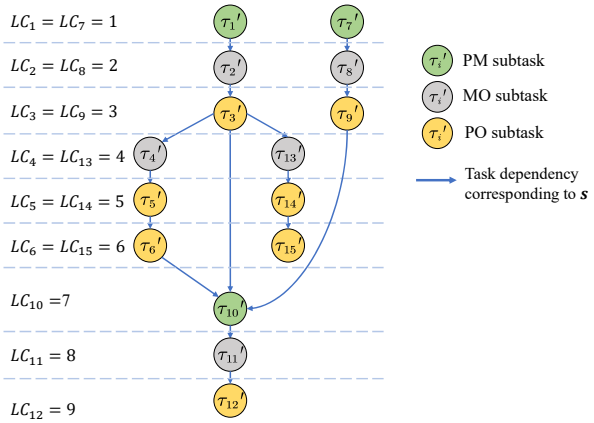


Fig. 4. An example of subtask DAG and task layering.

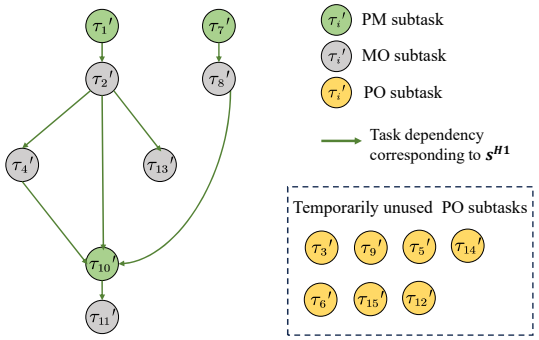


Fig. 5. An example of the subtask DAG without PO subtasks (i.e., matrix s^{H1}).

5.2 Step 1: The Mapping of PM and MO Subtasks

In this step, we mainly consider the PM and MO subtasks, ensuring that all mandatory cycles can be scheduled within the time and energy constraints. Any PO subtasks are deferred to Step 2. After removing PO subtasks from the subtask DAG, we get a reduced DAG with adjacency matrix s^{H1}, such as the example in Fig. 5.

5.2.1 Formulation of PM and MO Subtasks Mapping. Let $\mathcal{N}^{PM}$ and $\mathcal{N}^{MO}$ be the sets of PM and MO subtasks, respectively, and define

$$\mathcal{N}_1 \triangleq \mathcal{N}^{PM} \cup \mathcal{N}^{MO}.$$

We seek an assignment of each subtask $\tau_i' \in \mathcal{N}_1$ to a processor and a V/F level to minimize overall execution time and energy consumption. Therefore, more system resources (i.e., time and energy) can be left for executing PO subtasks in the next step, obtaining a higher QoS. Let H be the hyperperiod and E_s the energy supply over H. We consider the following bi-objective model:

$$PP1: \min_{\boldsymbol{x},\boldsymbol{c},\boldsymbol{u},\boldsymbol{ts}} \omega_1 \frac{t_{tot}}{H} + \omega_2 \frac{E_{tot}}{E_s}, \tag{41}$$

$$s.t. \begin{cases} (4)-(6), (15)-(22), \\ 0 \le ts_i \le H,\ x_{i,k}, c_{i,k,l}, u_{i,j} \in \{0,1\}, \quad \forall i,j \in \mathcal{N}^{PM} \cup \mathcal{N}^{MO},\ \forall k \in \mathcal{M},\ \forall l \in \mathcal{L}, \end{cases}$$

Algorithm 1: PM and MO Subtasks Mapping Scheme

Input : Subtask DAG (s^{H1}) without PO subtasks, platform parameters, energy supply E_s, hyperperiod H
Output: Mapping $\boldsymbol{c}, \boldsymbol{x}, \boldsymbol{ts}, \boldsymbol{te}$, total energy E_{tot}, total time t_{tot}

Initialize $E_{tot} = M\,H\,P_{on}$, $t_{tot} = 0$, $goal_{\min} = +\infty$, $\boldsymbol{ts} = +\infty$, $\boldsymbol{te} = -1$, $\omega_1 = \omega_2 = 0.5$;
Split tasks into PM, MO, and PO subtasks and build s^{H1};
Compute layering to get $Seq_{s^{H1}}$;
foreach *subtask* τ'_i *in* $Seq_{s^{H1}}$ **do**
  $k^* \leftarrow -1$, $l^* \leftarrow -1$, $goal_{\min} \leftarrow +\infty$;
  Update $te_i^{Pre^{s^{H1}}}$: the largest end time among predecessors;
  for $k = 1$ **to** M **do**
    if (θ_k *already assigned a subtask on the same layer as* τ'_i) **then**
      continue;
    for $l = 1$ **to** L **do**
      $t_{i,k,l} \leftarrow \dfrac{\varphi_i^H}{\lambda_{i,k} f_{k,l}}$, $E_{i,k,l} \leftarrow t_{i,k,l}\left(P^d_{k,l} + P^s_{k,l}\right)$;
      if $(E_{tot} + E_{i,k,l} \le E_s)$ ***and*** $\max(t_k^{core}, te_i^{Pre^{s^{H1}}}) + t_{i,k,l} \le d'_i$ **then**
        if $\omega_1 \dfrac{t_{tot} + t_{i,k,l}}{H} + \omega_2 \dfrac{E_{tot} + E_{i,k,l}}{E_s} < goal_{\min}$ **then**
          $goal_{\min} \leftarrow \omega_1 \dfrac{t_{tot} + t_{i,k,l}}{H} + \omega_2 \dfrac{E_{tot} + E_{i,k,l}}{E_s}$;
          $k^* \leftarrow k$, $l^* \leftarrow l$;
  if $k^* \neq -1$ **then**
    $c_{i,k^*,l^*} \leftarrow 1$, $x_{i,k^*} \leftarrow 1$;
    $ts_i \leftarrow \max\left(t_k^{core}, te_i^{Pre^{s^{H1}}}\right)$;
    $te_i \leftarrow t_{k^*}^{core} \leftarrow ts_i + t_{i,k^*,l^*}$;
    $E_{tot} \leftarrow E_{tot} + E_{i,k^*,l^*}$, $t_{tot} \leftarrow t_{tot} + t_{i,k^*,l^*}$;
  else
    Mapping fails for subtask τ'_i**, break.**;

where $\mathcal{N}^{PM}$ and $\mathcal{N}^{MO}$ are the sets of PM and MO subtasks, respectively. ω_1 and ω_2 are the weights of subtask execution time $\frac{t_{tot}}{H}$ and energy $\frac{E_{tot}}{E_s}$. Since time t_{tot} and energy E_{tot} have different units, we normalize them by using the scheduling horizon H and energy supply E_s. Note that in PP1 (41), only PM and MO subtasks are considered. Therefore, replacing $\forall i \in \mathcal{N}'$ with $\forall i \in \mathcal{N}^{PM} \cup \mathcal{N}^{MO}$ in (4)-(6) and (15)-(22), we can get the constraints for PP1 (41).

5.2.2 The PM and MO Subtasks Mapping Scheme. Algorithm 1 details how we map PM and MO subtasks, denoted as $PP1$ in (41). Let t_k^{core} be the time at which processor θ_k finishes the last assigned subtask. For each subtask τ'_i in a topological order $Seq_{s^{H1}}$, we iterate over all processors k and all V/F levels l to compute the subtask's execution time $t_{i,k,l}$ and energy $E_{i,k,l}$. We only retain those allocations that a) do not exceed the total energy budget E_s and b) meet the subtask deadline d'_i. Among these feasible assignments, we select the one that yields the lowest combined objective:

$$\omega_1 \frac{(t_{tot} + t_{i,k,l})}{H} + \omega_2 \frac{(E_{tot} + E_{i,k,l})}{E_s}.$$

This balances shorter execution time against lower energy consumption. Once the best allocation is identified, we update $c_{i,k^*,l^*} = 1$, $x_{i,k^*} = 1$, set the subtask's start/end times (ts_i, te_i), and update t_{tot} and E_{tot}. The result is a partial mapping of all PM and MO subtasks under time/energy constraints, as shown in Fig. 6.

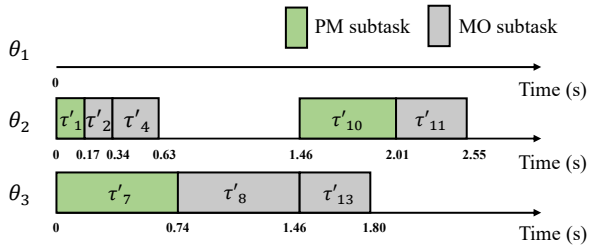


Fig. 6. The result of PM and MO subtasks mapping in Step 1.

Fig. 6 shows an example outcome of mapping PM and MO subtasks in Step 1, where each subtask is assigned to a processor and V/F level without violating energy and deadline constraints. Notice that subtasks within the same layer are independent and can be executed in parallel across different processors to reduce the overall makespan. For instance, in Fig. 6, subtasks τ'_1 and τ'_7 are placed on two distinct processors, allowing concurrent execution.

Note that task migration overhead is not considered in Algorithm 1. This is because in this step, we mainly focus on mapping PM and MO subtasks, while mapping PO subtasks has not been explored. The exact execution cycles of PO subtasks, their dependencies, and the allocation of all subtasks are unknown, as they are determined by Algorithm 2. On the other hand, if we consider migration overhead between PM and MO subtasks in Algorithm 1, then, when PO subtasks are handled in Algorithm 2, the dependencies among PM, MO, and PO will change. Consequently, we need to re-evaluate the migration overhead among PM, MO, and PO, which increases the computational complexity of Algorithm 2 (e.g., we need to recompute subtask start/end times, possibly moving subtasks again).

5.3 Step 2: The Mapping of PO Subtasks

In Step 1, all mandatory subtasks are executed (including partial optional cycles, if present in MO subtasks). In this second step, we address the Purely Optional (PO) subtasks, intending to maximize overall QoS.

5.3.1 Formulation of PO Subtask Mapping. Let $\mathcal{N}^{PO}$ be the set of PO subtasks. In this phase, we assign each PO subtask τ'_i to a processor and select a voltage-frequency level, determining $x_{i,k}$, $c_{i,k,l}$, $u_{i,j}$, ts_i, te_i, and o_i. Combined with the results of Step 1 (which fixed the assignments for PM and MO subtasks), we finalize the entire task mapping. Task migration overheads apply here as well, since the mapping of PO subtasks may influence the already assigned tasks.

Formally, the PO subtask assignment is:

$$PP2: \quad \max_{\boldsymbol{x},\boldsymbol{c},\boldsymbol{u},\boldsymbol{ts},\boldsymbol{te},\boldsymbol{o}} \sum\nolimits_{i\in\mathcal{N}} o_i, \tag{42}$$

$$\text{s.t.} \begin{cases} (4)-(14),\ (15)-(22), \\ 0 \le o_i \le O_i,\ \forall i \in \mathcal{N}, \quad 0 \le ts_i \le H,\ \forall i \in \mathcal{N}^{PM} \cup \mathcal{N}^{MO} \cup \mathcal{N}^{PO}, \\ x_{i,k},\ c_{i,k,l},\ u_{i,j} \in \{0,1\}, \forall i, j \in \mathcal{N}^{PM} \cup \mathcal{N}^{MO} \cup \mathcal{N}^{PO},\ \forall k \in \mathcal{M},\ \forall l \in \mathcal{L}. \end{cases}$$

All constraints from the original Primal Problem (3) remain in force since the new PO allocations can affect the scheduling of already-placed PM and MO subtasks (e.g., due to migration overhead).

5.3.2 The PO Subtask Mapping Scheme. Algorithm 2 attempts to allocate each PO subtask τ'_i onto a feasible time slot of some processor θ_k and then selects a V/F level l that allows the largest possible optional cycles, subject to constraints on timing, energy, and migration overhead. Fig. 7 shows an

Table 2. Variables used in Algorithm 2

Binary Variables	
s^{H2}_{old}	dependency matrix of the previous loop
s^{H2}_{new}	dependency matrix of the current loop (by inserting τ'_i to the s^{H2}_{old} DAG)
Integer Variables	
$NumPreMig_{i,k}$	number of task migrations from τ'_i's predecessors to τ'_i, if τ'_i is assigned to θ_k
$NumSucMig_{i,k}$	number of task migrations from τ'_i to its successors, if τ'_i is assigned to θ_k
$NumMig_i$	number of task migration from τ'_i to its successors, when the task mapping is determined
$NumAct_k$	number of occupied time period of θ_k
Continuous Variables	
$tsFea_{i,k}$	earliest start time of τ'_i, if τ'_i is assigned to θ_k
$teFea_{i,k}$	latest end time of τ'_i, if τ'_i is assigned to θ_k
$EMig_{i,k}$	migration energy overhead, if τ'_i is assigned to θ_k
$EMig_{tot}$	total migration energy overhead, when task allocation is determined
$tsCoreAct_{k,q}$	start time of the q'th active time peroid of θ_k
$teCoreAct_{k,q}$	end time of the q'th active time peroid of θ_k
$\varphi^{time}_{i,k,l}$	maximum optional cycle of τ'_i under time constraint, if it is assigned to θ_k with $(V_{k,l}, f_{k,l})$
$\varphi^{energy}_{i,k,l}$	maximum optional cycle of τ'_i under energy constraint, if it is assigned to θ_k with $(V_{k,l}, f_{k,l})$
$\varphi^{o,Fea}_{i,k,l}$	maximum optional cycle of τ'_i under time and energy constraint, if it is assigned to θ_k with $(V_{k,l}, f_{k,l})$
t^{core}_j	end time of the subtask before τ'_j on the same core

example for assigning PO subtask τ'_5. Table 2 summarizes the key variables, and the main ideas are described as follows.

- **Inserting τ'_i into the DAG:** We treat τ'_i as added to the existing subtask DAG (s^{H2}_{old}) to create s^{H2}_{new} (Line 7). If previous PO subtasks were rejected due to infeasibility, they are not present in the DAG.
- **Earliest Start & Successor Constraints:** We update $te_i^{Pre^{s^{H2}_{new}}}$ and $ts_i^{Suc^{s^{H2}_{new}}}$ to reflect the new DAG structure (Line 8).
- **Migration Calculations (Algorithm 3):** For each candidate processor θ_k, we can compute $NumPreMig_{i,k}$, $NumSucMig_{i,k}$, the earliest feasible start time $tsFea_{i,k}$, and the associated migration energy $EMig_{i,k}$.
- **Idle Interval Analysis:** We examine the current schedule on θ_k to find an idle interval that can accommodate τ'_i. The feasible end time $teFea_{i,k}$ is limited by τ'_i's deadline, the start time of its successors (minus any required migration overhead), and the next active interval on θ_k.
- **Maximizing τ'_i's Executed Cycles:** Given time and energy constraints, we pick the V/F level $(v_{k,l}, f_{k,l})$ that yields the largest feasible optional cycles $\varphi^{o,Fea}_{i,k,l}$ (Lines 21–23).

5.3.3 Algorithm 4: Parameter Updating and Subtask Adjustment. After selecting a processor k^*, V/F level l^*, and feasible optional cycles $\varphi_i^{o,\max}$ for a PO subtask τ'_i, we invoke Algorithm 4 to update the associated parameters (e.g., $\boldsymbol{c}$, $\boldsymbol{o}$, $\boldsymbol{ts}$, $\boldsymbol{te}$, $\widehat{\boldsymbol{te}}$, $NumMig_i$, E_{tot}, $EMig_{tot}$) and to integrate τ'_i into or remove it from the DAG, depending on feasibility. When PO subtask τ'_i is inserted or removed, the dependencies and execution time of its predecessors and successors will change. Through the allocation of τ'_i and its predecessors and successors, we can adjust task execution time and calculate task migration costs.

- **Successful Mapping:** If τ'_i can be assigned a positive number of optional cycles ($\varphi_i^{o,\max} > 0$), then Lines 1–14 finalize its allocation on processor θ_{k^*} at V/F level l^*, updating the subtask's start/end times, energy consumption, and migration parameters.

Algorithm 2: PO Subtasks Mapping Scheme

Input : Subtask DAG, platform parameters, current mappings $\boldsymbol{c}, \boldsymbol{x}, \boldsymbol{ts}, \boldsymbol{te}, E_{tot}, t_{tot}$
Output: Updated mappings $\boldsymbol{c}, \boldsymbol{x}, \boldsymbol{ts}, \boldsymbol{te}, E_{tot}, t_{tot}$, and $\boldsymbol{o}$

1 Initialize $\boldsymbol{s}_{old}^{H2} = \boldsymbol{s}^{H1}$;
2 $IfOccupy_{i,k} = 0$;
3 $\varphi_i^{o,\max} = -1$;
4 $tsCoreAct = +\infty$;
5 **for** $i = Seq_s$ ***and*** $\tau_i' \in \mathcal{N}^{PO}$ **do** // Assign PO subtasks in DAG order
6 $l^* \leftarrow -1,\ k^* \leftarrow -1$;
7 Insert τ_i' into $\boldsymbol{s}_{old}^{H2}$, creating new DAG with matrix $\boldsymbol{s}_{new}^{H2}$;
8 Update $te_i^{Pre^{s_{new}^{H2}}}$ and $ts_i^{Suc^{s_{new}^{H2}}}$;
9 Call **Algorithm 3** to compute migration parameters;
10 Calculate the number of occupied periods of each processor θ_k as $NumAct$;
11 Update $tsCoreAct,\ teCoreAct$;
12 **for** $k = 1 : M$ **do** // Check each processor θ_k
13 **for** $q = 1 : NumAct - 1$ **do** // Identify idle interval in θ_k
14 **if** $teCoreAct_{q,k} \le tsFea_{i,k}$ ***and*** $tsCoreAct_{q+1,k} \ge tsFea_{i,k}$ **then** // Check if τ_i' can start in an idle gap
15 **break**;
16 $teFea_{i,k} \leftarrow \min\left(d_i',\ ts_i^{Suc^{s_{new}^{H2}}} - T^m \times NumSucMig_{i,k},\ tsCoreAct_{q+1,k}\right)$
17 **if** $teFea_{i,k} - tsFea_{i,k} > 0$ ***and*** $E_s - E_{tot} - EMig_{i,k} \ge 0$ **then** // Check time and energy feasibility
18 **for** $l = 1 : L$ **do**
19 $\varphi_{i,k,l}^{time} \leftarrow (teFea_{i,k} - tsFea_{i,k})\,\lambda_{i,k} f_{k,l}$;
20 $\varphi_{i,k,l}^{energy} \leftarrow \dfrac{(E_s - E_{tot} - EMig_{i,k})\,\lambda_{i,k} f_{k,l}}{P_{i,k}^d + P_{i,k}^s}$;
21 $\varphi_{i,k,l}^{o,Fea} \leftarrow \min\{\varphi_i^H,\ \varphi_{i,k,l}^{time},\ \varphi_{i,k,l}^{energy}\}$;
22 **if** $\varphi_{i,k,l}^{o,Fea} > \varphi_i^{o,\max}$ **then** // Track maximum feasible cycles
23 $\varphi_i^{o,\max} \leftarrow \varphi_{i,k,l}^{o,Fea},\quad l^* \leftarrow l,\quad k^* \leftarrow k$;
24 Call **Algorithm 4** to update parameters for τ_i';
25 $QoS \leftarrow \sum_{i \in \mathcal{N}} o_i$;

- **Subtask Rejection:** If $\varphi_i^{o,\max} \le 0$, τ_i' is dropped. To maintain consistency, we recalculate the global task migration matrix $\alpha_{i,j}$ (Line 15) and adjust previously assigned subtasks if necessary. If the total energy $E_{tot} + EMig_{tot}$ still satisfies E_s (Line 18), we can fine-tune the start/end times of affected subtasks (Lines 27–31). Otherwise, the schedule fails on energy grounds.

Overall, Algorithm 2 completes the PO subtask allocation, generating a feasible or partially feasible configuration that aims to maximize system QoS. Finally, the total QoS is computed as $QoS = \sum_{i \in \mathcal{N}} o_i$ (Line 25).

6 Simulation Results

In this section, we evaluate the performance of the proposed task mapping methods through a series of simulation experiments.

6.0.1 Simulation Setup. Table 3 summarizes the main parameters for the simulation. Each task τ_i has mandatory cycles M_i and optional cycles O_i within the range $[4 \times 10^7, 6 \times 10^8]$ [24], given by the practical applications. The execution efficiency parameter satisfies $\lambda_{i,k} \in [0.4, 1]$ [18]. Let

Algorithm 3: Subpart 1 of Algorithm 2

Input : Subtask DAG, platform parameters, index i, DAG s_{new}^{H2}
Output: $NumPreMig_{i,k}$, $NumSucMig_{i,k}$, $tsFea_{i,k}$, $EMig_{i,k}$

Initialize $NumSucMig_{i,k} \leftarrow 0$;
for $k = 1 : M$ **do**
  for $j = 1 : MN$ **do**
    if $\tau'_j \in Pre_i^{s_{new}^{H2}}$ **then** // Check predecessor mapping
      if τ'_j *assigned to* θ_k **then** // If predecessor in same processor
        $tsFea_{i,k} \leftarrow te_i^{Pre^{s_{new}^{H2}}}$;
        $NumPreMig_{i,k} \leftarrow 0$;
      else
        $tsFea_{i,k} \leftarrow T^m + te_i^{Pre^{s_{new}^{H2}}}$;
        $NumPreMig_{i,k} \leftarrow 1$;
    if $\tau'_j \in Suc_i^{s_{new}^{H2}}$ **then** // Check successor mapping
      if τ'_j *not assigned to* θ_k **then**
        $NumSucMig_{i,k} \leftarrow NumSucMig_{i,k} + 1$;
for $k = 1 : M$ **do** // Compute total migration overhead
  Calculate $\boldsymbol{\alpha}$ via (14) (assume $x_{i,k} = 1$ and $\widehat{s} = s_{new}^{H2}$);
  $EMig_{i,k} \leftarrow E^m \times \sum_{i \in \mathcal{N}'} \sum_{j \in \mathcal{N}'} \alpha_{i,j}$;

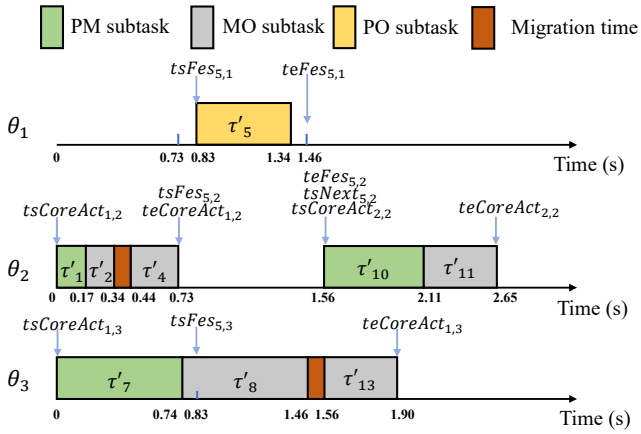


Fig. 7. Example of assigning PO subtask τ'_5: choosing a processor and V/F level to maximize its execution cycles.

$d_{\max}$ denote the time to execute all tasks along the Critical Path (CPT) using the platform's lowest frequency, while $d_{\min,i}$ is the time to execute task τ_i using the platform's highest frequency. We set each task's deadline to $d_i = (d_{\max} - d_{\min,i})\epsilon + d_{\min,i}$, where $\epsilon \in [0, 1]$ is a time factor. The mapping horizon is $H = \max_{i \in \mathcal{N}}\{d_i\}$. For the energy supply E_s, we define two benchmarks: $E_{\min}$, the energy required to execute all tasks at the most energy-efficient setting, and $E_{\max}$, the energy required to execute all tasks at the least energy-efficient setting. Then, $E_s = (E_{\max} - E_{\min})\delta + E_{\min} + MHP_{on}$, where $\delta \in [0, 1]$ is an energy factor and MHP_{on} is the inherent (power-on) energy for the M processors over the horizon H.

Table 4 shows the parameters of a heterogeneous multicore platform with six processors, each offering different V/F levels. In the experiments, the selections of M processors are as follows:

Algorithm 4: Subpart 2 of Algorithm 2

Input : Subtask DAG, platform parameters, index i, partial mappings $\boldsymbol{c}$, DAGs $\boldsymbol{s}^{H2}_{new}$, $\boldsymbol{s}^{H2}_{old}$, $\varphi_i^{o,\max}$, $\boldsymbol{te}$, $\boldsymbol{ts}$, $k^*, l^*, tsFea_i, NumSucMig_{i,k}, NumPreMig_{i,k}, E_{tot}, EMig_{i,k}$

Output: Updated $\boldsymbol{c}$, $\boldsymbol{o}$, $\boldsymbol{ts}$, $\boldsymbol{te}$, $\widehat{\boldsymbol{te}}$, $NumMig_i, E_{tot}, EMig_{tot}, \boldsymbol{s}^{H2}_{old}$

if $\varphi_i^{o,\max} > 0$ **then** // Case 1: Subtask is mapped successfully
  $c_{i,k^*,l^*} \leftarrow 1, \quad x_{i,k^*} \leftarrow 1;$
  $o_{(1+i) \bmod M} \leftarrow \varphi_i^{o,\max};$
  $ts_i \leftarrow tsFea_{i,k^*};$
  $te_i \leftarrow tsFea_{i,k^*} + \dfrac{\varphi_i^{o,\max}}{\lambda_{i,k^*} f_{k^*,l^*}};$
  $\widehat{te}_i \leftarrow te_i + T^m \times NumSucMig_{i,k^*};$
  $NumMig_i \leftarrow NumSucMig_{i,k^*};$
  $E_{tot} \leftarrow E_{tot} + (te_i - ts_i)\left(P^d_{i,k^*} + P^s_{i,k^*}\right);$
  $EMig_{tot} \leftarrow EMig_{i,k^*};$
  for $j = 1 : MN$ **do**
    if $\tau'_j \in Pre_i^{\boldsymbol{s}^{H2}_{new}}$ **then**
      $NumMig_j \leftarrow NumPreMig_{i,k^*};$
      $\widehat{te}_j \leftarrow te_j + T^m \times NumMig_j;$
  $\boldsymbol{s}^{H2}_{old} \leftarrow \boldsymbol{s}^{H2}_{new};$
if $\varphi_i^{o,\max} \leq 0$ **then** // Case 2: Subtask is not mapped
  Calculate $\alpha_{i,j}$ via (14) (using $\widehat{\boldsymbol{s}} = \boldsymbol{s}^{H2}_{old}$);
  $EMig_{tot} \leftarrow E^m \times \sum_{i \in \mathcal{N}'} \sum_{j \in \mathcal{N}'} \alpha_{i,j};$
  if $E_{tot} + EMig_{tot} > E_s$ **then** // Check energy feasibility for re-adjustment
    break // energy constraint not satisfied
  else
    for $j = 1 : MN$ **do**
      if τ'_j *assigned to* θ_k *with* $(V_{k,l}, f_{k,l})$ **then** // Adjust previously assigned subtasks
        Calculate $\widehat{\boldsymbol{te}}$ via (18);
        Update $te^{Pre^{\boldsymbol{s}^{H2}_{old}}}$;
        Update $tsCoreAct$, $teCoreAct$, t_j^{core};
        $t_j^{temp} \leftarrow te_j - ts_j;$
        $ts_j \leftarrow \max\left(ts_j, te_j^{Pre^{\boldsymbol{s}^{H2}_{old}}}, t_j^{core}\right);$
        if $ts_j + t_j^{temp} > d'_j$ **then**
          break // deadline constraint not satisfied
        else
          $te_j \leftarrow ts_j + t_j^{temp};$

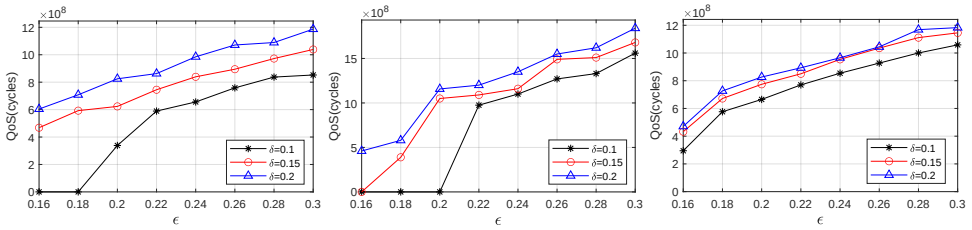
$M = 2 \Rightarrow \{\theta_1, \theta_6\}$; $M = 3 \Rightarrow \{\theta_1, \theta_3, \theta_6\}$; $M = 6 \Rightarrow \{\theta_1, \ldots, \theta_6\}$. In the following section, we first evaluate the impact of system parameters on task mapping results, e.g., time factor ϵ and energy factor δ. Then, we compare the system performances, e.g., system QoS, problem feasibility, and computation time, achieved by the proposed optimal (OPT) and heuristic (HEU) methods with the State-of-the-Art (SoA) methods, e.g., task mapping Without Task Migration (WTM) [8], task mapping Without DVFS (WDV) [26], and task mapping with Fixed Migration Number (FMN) [22]. All experiments are conducted on a server featuring a 32-core 3.7 *GHz* AMD® Ryzen™ Threadripper™ 3970X processor and 160 *GB* of RAM. We use MATLAB® 2021b together with Gurobi 11 for optimization.

Table 3. Parameters of experiment setup

Task Parameters	
$M_i, O_i \in [4 \times 10^7, 6 \times 10^8]$	$\lambda_{i,k} \in [0.4, 1]$
$d_{\min,i} = \min_{k \in \mathcal{M}, l \in \mathcal{L}} \left\{ \frac{M_i + O_i}{\lambda_{i,k} f_{k,l}} \right\}$	
$d_{\max} = \max_{k \in \mathcal{M}, l \in \mathcal{L}} \left\{ \sum_{i \in \text{CPT}} \frac{M_i + O_i}{\lambda_{i,k} f_{k,l}} \right\}$	
$d_i = (d_{\max} - d_{\min,i})\epsilon + d_{\min,i}$	$H = \max_{i \in \mathcal{N}} \{d_i\}$
Energy Constraint	
$E_{\min} = \min_{k \in \mathcal{M}, l \in \mathcal{L}} \left\{ \sum_{i \in \mathcal{N}} \frac{M_i + O_i}{\lambda_{i,k} f_{k,l}} (P^s_{k,l} + P^d_{k,l}) \right\}$	
$E_{\max} = \max_{k \in \mathcal{M}, l \in \mathcal{L}} \left\{ \sum_{i \in \mathcal{N}} \frac{M_i + O_i}{\lambda_{i,k} f_{k,l}} (P^s_{k,l} + P^d_{k,l}) \right\}$	
$E_s = (E_{\max} - E_{\min})\delta + E_{\min} + MHP_{on}$	

Table 4. Parameters of multicore platform

θ_1	$V\ (Voltage)$	0.93	0.96	1	1.04	1.08	1.1	1.15	1.2	1.23	$C^s = 1.478$, $C^d = 0.471$,
	$f\ (GHz)$	0.8	0.9	1	1.1	1.2	1.3	1.4	1.5	1.6	$\rho = 0.379$, $P_{on} = 0.08mW$
θ_2	$V\ (Voltage)$	0.94	1.00		1.06		1.13		1.19		$C^s = 1.406$, $C^d = 0.391$,
	$f\ (GHz)$	0.7	0.8		0.9		1.0		1.1		$\rho = 0.474$, $P_{on} = 0.08mW$
θ_3	$V\ (Voltage)$	0.91			0.95			1.03			$C^s = 1.235$, $C^d = 0.218$,
	$f\ (GHz)$	0.65			0.7			0.75			$\rho = 0.526$, $P_{on} = 0.08mW$
θ_4	$V\ (Voltage)$	0.92	0.98		1.05		1.12		1.18		$C^s = 1.163$, $C^d = 0.172$,
	$f\ (GHz)$	0.35	0.45		0.55		0.65		0.75		$\rho = 0.662$, $P_{on} = 0.08mW$
θ_5	$V\ (Voltage)$	0.91	0.96		1.03		1.10		1.19		$C^s = 1.177$, $C^d = 0.163$,
	$f\ (GHz)$	0.30	0.38		0.48		0.58		0.68		$\rho = 0.710$, $P_{on} = 0.08mW$
θ_6	$V\ (Voltage)$	0.9	0.94		1.01		1.09		1.2		$C^s = 1.191$, $C^d = 0.153$,
	$f\ (GHz)$	0.25	0.3		0.4		0.5		0.6		$\rho = 0.757$, $P_{on} = 0.08mW$



(a) $N = 4$. (b) $N = 5$. (c) $N = 6$.

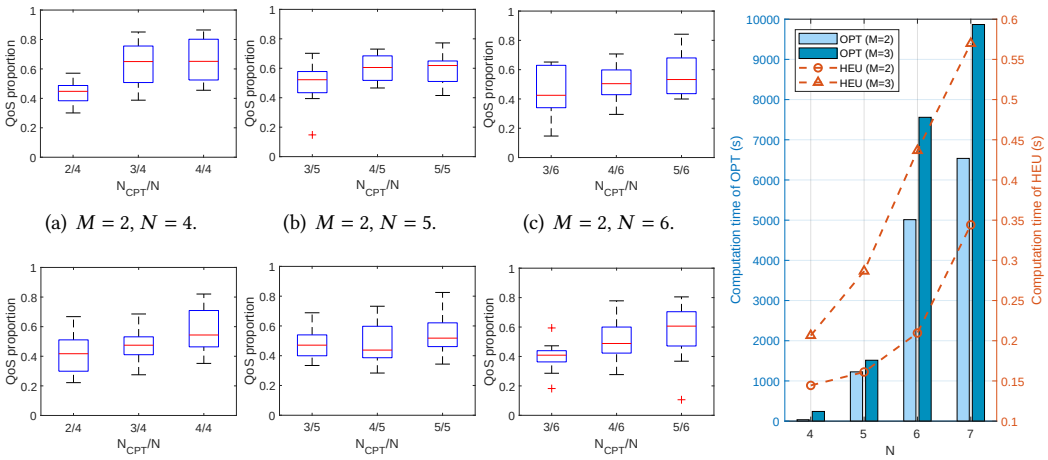
Fig. 8. The QoS achieved by OPT method with parameters ϵ and δ varying.

6.0.2 Simulation Results. Fig. 8 illustrates how the system QoS varies under changes in ϵ (time factor) and δ (energy factor) when $N \in \{4, 5, 6\}$. We assume a three-processor platform (see Table 4) with $\lambda_{i,k} = 0.8$, $T^m = 100\ ms$, and $E^m = 10\ mJ$. The deadlines d_i and energy supply E_s shrink or grow according to ϵ and δ. If the mapping fails to meet feasibility constraints, QoS is zero as $Q_b = 0$ and $\sum_{i \in \mathcal{N}} o_i = 0$. As both ϵ and δ increase, more optional cycles can be executed, boosting QoS. Conversely, when ϵ and/or δ are too small (e.g., $\epsilon \leq 0.16$ and $\delta = 0.1$), the mapping often fails. Once δ and ϵ exceed certain thresholds (e.g., $\delta = 0.2$ and $\epsilon = 0.28$–0.3 in Fig. 8(c)), raising them further provides only marginal QoS gains, because each task's optional cycles are capped by O_i.

Table 5 shows how QoS and task migration number change for different task migration time $T^m \in [0, 100]\ ms$ and energy $E^m \in [0, 10]\ mJ$, based on different applications [4, 19, 27]. Each entry has the form QoS proportion (migration number), where QoS proportion is defined as the rate between the achieved QoS under the given T^m and E^m, and the zero-overhead case, i.e.,

Table 5. QoS proportion and number of task migrations under different task migration overheads.

	E^m (mJ)					
T^m (ms)	0	0.1	0.5	1	5	10
0	1.00 (7)	0.99 (7)	0.99 (7)	0.98 (7)	0.98 (7)	0.97 (7)
1	0.99 (7)	0.98 (7)	0.97 (6)	0.97 (6)	0.97 (6)	0.96 (6)
10	0.99 (7)	0.98 (6)	0.97 (6)	0.97 (6)	0.96 (6)	0.96 (6)
50	0.98 (7)	0.97 (6)	0.97 (6)	0.97 (6)	0.94 (6)	0.94 (5)
100	0.97 (6)	0.97 (6)	0.97 (6)	0.95 (5)	0.94 (5)	0.93 (5)



(a) $M = 2, N = 4$. (b) $M = 2, N = 5$. (c) $M = 2, N = 6$.

(d) $M = 3, N = 4$. (e) $M = 3, N = 5$. (f) $M = 3, N = 6$. (g) Computation time.

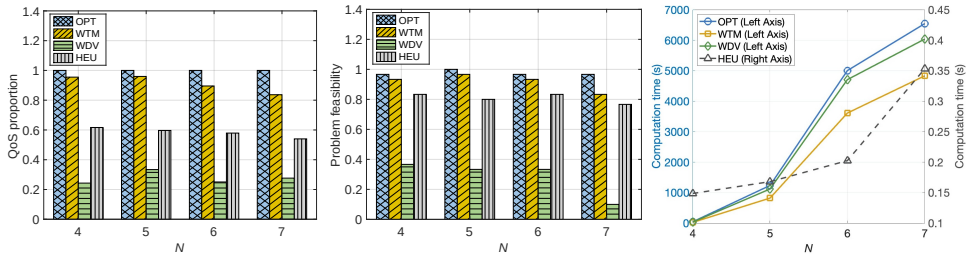
Fig. 9. QoS and computation time comparisons of the OPT and HEU methods.

QoS proportion = $\text{QoS}(T^m, E^m)/\text{QoS}(0, 0)$. The QoS proportion reflects the QoS degradation caused by task migration overhead, compared to the case without it. From Table 5, we can see that, as the values of T^m and E^m increase, both QoS and the number of migrations decrease, compared to the case without migration overhead. Nevertheless, the degradation is limited: even in the pessimistic case $(100\ ms, 10\ mJ)$, the QoS proportion remains 0.93, while the total number of migrations only decreases from at most 7 to 5. Our approach adaptively optimizes migration based on the overhead parameters, and migration offers flexibility by reallocating subtasks to more suitable processors.

In Fig. 9, we compare the QoS and computation times for the OPT and HEU methods with $N \in \{4, 5, 6\}$, $M \in \{2, 3\}$, $\epsilon = \delta = 0.2$, $E^m = 10\ mJ$, and $T^m = 100\ ms$. To quantify DAG parallelism, let N_{CPT} be the number of tasks on the critical path. A smaller N_{CPT}/N ratio implies greater parallelism.

Fig. 9(a)–Fig. 9(f) show the ratio $\text{QoS}_{\text{HEU}}/\text{QoS}_{\text{OPT}}$, averaged over 12 random DAGs for each parameter combination. On average, the HEU solution achieves about 57.7% of OPT QoS, although it can reach 86.5%. The gap narrows as N_{CPT}/N grows (i.e., fewer parallel tasks), because the heuristic has fewer choices for optional-subtask placement.

Fig. 9(g) contrasts the average runtime of the two methods for 20 random DAG instances at each (N, M) setting. The runtime grows with N and M for both methods, but the OPT approach becomes significantly more expensive (up to 9867.65 s) due to the large mixed-integer search space. The HEU method, by contrast, remains below a second ($[0.1444,\ 0.5698]$ s) in all tested cases, demonstrating the benefits of problem decomposition.



(a) System QoS. (b) Problem feasibility. (c) Computation time.

Fig. 10. QoS, feasibility, and computation time comparison of OPT, WTM, WDV, and HEU methods.

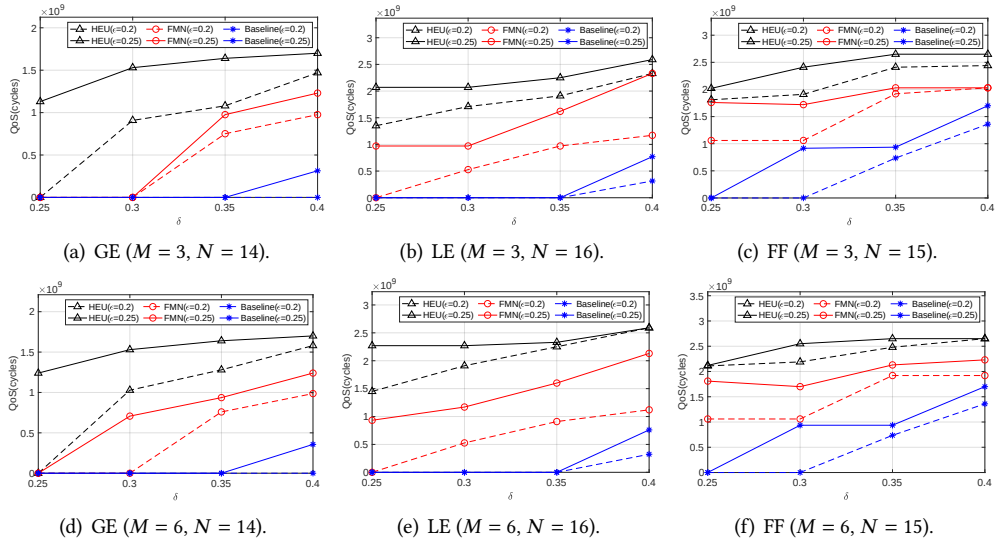
Table 6. QoS proportion of HEU method under different ω_1/ω_2 in real DAGs ($M = 3, 6$).

ω_1/ω_2	DAG	GE ($N = 14$)						LE ($N = 16$)						FF ($N = 15$)					
	ϵ	0.25		0.3		0.3		0.25		0.3		0.3		0.25		0.25		0.3	
	δ	0.3		0.3		0.4		0.3		0.3		0.35		0.25		0.35		0.35	
	M	3	6	3	6	3	6	3	6	3	6	3	6	3	6	3	6	3	6
0		×	×	×	×	0.36	0.38	×	×	×	×	×	×	×	×	×	×	×	×
0.5		×	×	0.36	0.38	0.45	0.47	×	×	**0.41**	0.41	**0.48**	**0.48**	×	×	×	×	×	×
0.8		**0.40**	0.38	0.36	0.38	**0.49**	**0.47**	0.38	**0.41**	**0.41**	**0.48**	0.45	**0.48**	×	×	**0.47**	0.47	**0.53**	**0.53**
1		0.22	**0.42**	**0.47**	**0.47**	**0.49**	0.47	**0.41**	**0.41**	**0.41**	0.41	0.41	0.41	**0.35**	**0.35**	**0.47**	**0.50**	0.47	0.47
1.5		×	0.22	0.27	0.30	0.31	0.33	0.34	0.36	0.38	0.38	0.38	0.38	0.29	**0.35**	0.35	0.36	0.46	0.47
2		×	×	0.27	0.27	0.31	0.33	×	0.35	0.34	0.36	0.34	0.36	×	0.29	0.29	0.36	0.29	0.35
4		×	×	×	0.27	0.29	0.29	×	×	×	×	0.34	0.36	×	0.29	0.29	0.29	0.29	0.29
$+\infty$		×	×	×	×	0.27	0.27	×	×	×	×	0.34	0.36	×	×	0.29	0.29	0.29	0.29

In Fig. 10, we compare QoS, feasibility, and computation time among 4 methods: OPT (proposed optimal), WTM (optimal but without task migration [8]), WDV (optimal but without DVFS [26]), and HEU (proposed heuristic). We fix $\epsilon = \delta = 0.2$, $M = 2$, $N \in [4, 7]$, $\lambda_{i,k} = 0.8$, $T^m = 100\ ms$, and $E^m = 10\ mJ$. For each N, we run 30 experiments using randomly generated DAGs. In Fig. 10(a), we normalize QoS_{OPT} as 1, then measure $\text{QoS}_{\text{WTM}}/\text{QoS}_{\text{OPT}}$, $\text{QoS}_{\text{WDV}}/\text{QoS}_{\text{OPT}}$, and $\text{QoS}_{\text{HEU}}/\text{QoS}_{\text{OPT}}$. Because OPT exploits both migration and DVFS, it outperforms WTM and WDV in QoS (up to 411.5%) and achieves greater feasibility in Fig. 10(b). This improvement reflects how extra flexibility (via DVFS and migration) helps meet time and energy constraints. Moreover, DVFS has a higher efficiency on QoS and feasibility improvement than task migration in our case.

In Table 6, we show how the bi-objective parameters ω_1 and ω_2 affect QoS for three real applications—Gaussian Elimination (GE, $N = 14$), Laplace Equation (LE, $N = 16$), and Fourier Transform (FF, $N = 15$) [22], where processor numbers $M = 3, 6$. The first step of the HEU method solves $PP1$ [see (41)] to minimize $\omega_1 \frac{t_{tot}}{H} + \omega_2 \frac{E_{tot}}{E_s}$. We set various ratios $\omega_1/\omega_2 \in \{0, 0.5, 0.8, 1, 1.5, 2, 4, +\infty\}$. Here, $\omega_1/\omega_2 = 0$ means we minimize only energy, while $\omega_1/\omega_2 = +\infty$ means we minimize only time. The QoS proportion is given by $\sum_{i \in \mathcal{N}} o_i \Big/ \sum_{i \in \mathcal{N}} O_i$, and the symbol "×" is used to denote infeasible mappings.

The table shows that under tight time and energy budgets, focusing on a single objective (pure time or pure energy minimization) often leads to failure. A balanced bi-objective approach is more likely to meet both constraints, enabling more optional cycles to be completed. For instance, ω_1/ω_2 around 1 (e.g., 0.5, 0.8, 1, or 1.5) often achieves higher QoS. This suggests that balancing time and energy in $PP1$ is beneficial for overall feasibility and performance; if $PP1$ fails, $PP2$ becomes less effective. Consequently, using a genuinely bi-objective approach better satisfies real-time and



(a) GE ($M = 3, N = 14$). (b) LE ($M = 3, N = 16$). (c) FF ($M = 3, N = 15$).

(d) GE ($M = 6, N = 14$). (e) LE ($M = 6, N = 16$). (f) FF ($M = 6, N = 15$).

Fig. 11. QoS comparison of the HEU, FMN, and baseline methods ($M = 3, 6$).

energy requirements, thereby achieving a higher QoS. In addition, the QoS proportion with $M = 6$ is higher than $M = 3$, as more processors lead to stronger task schedulability under limited resources and obtain higher QoS.

In Fig. 11, we compare system QoS achieved by the HEU, FMN [22], and Baseline method (heuristic without task migration) under different real applications (GE, LE, and FF) and system parameters (δ and ϵ). FMN splits a task τ_i into two parts, i.e., mandatory and optional subtasks, and assigns them to heterogeneous processors to execute. In addition, we set $M = \{3, 6\}$, $\lambda_{i,k} = 0.8$, $T^m = 100\ ms$, and $E^m = 10\ mJ$. The energy factor $\delta \in [0.25, 0.4]$ while the time factor $\epsilon \in [0.2, 0.35]$. Fig. 11 shows that HEU has a higher QoS than FMN (168.7% when $M = 3$ and 175.3% when $M = 6$ on average) and Baseline (625.0% when $M = 3$ and 661.9% when $M = 6$ on average), and the QoS gaps between HEU, FMN, and Baseline increase with processor number M. This is because HEU optimizes the number of task migrations and increases the flexibility and adaptability of task migration when more processors are available. Under limited system resources, e.g., $\delta = 0.3$ in Fig. 11(a), the FMN and baseline fail to map the tasks, resulting in zero QoS. However, the task mapping is successful by using HEU. By leveraging optimized task splitting and migration, we can improve task QoS and schedulability.

7 Conclusion

This paper investigates the problem of mapping approximate computing tasks on heterogeneous multicore platforms under dependency, energy, and real-time constraints. By jointly incorporating task migration and DVFS, we improve both energy efficiency and task schedulability during allocation and execution. Specifically, we first formulate a MINLP model, then linearize it into a MILP problem to enable an optimal solution search via existing solvers. To address the complexity of solving the full MILP directly, we propose a two-step heuristic that decomposes the original problem into sub-problems for mandatory and optional subtasks, each solved using a greedy-based approach. This decomposition significantly reduces computation time, while still satisfying the critical constraints.

Simulation results show that the proposed optimal method delivers higher QoS performance at approximately 238.9% on average and up to 411.5%, and achieves greater feasibility (around 326.3% on average and up to 966.7%) compared to other state-of-the-art approaches. Meanwhile, the heuristic method yields a QoS of about 57.7% of the optimal method while running in negligible time, making it well-suited for larger problem sizes or real-time scenarios where rapid decisions are paramount.

In sum, this work demonstrates that task migration and DVFS can be synergistically combined to improve system QoS and feasibility for approximate computing tasks. Future directions may involve exploring shared resource contention (e.g., memory bus) and inter-core interference in multicore systems. This will enable a more accurate, dynamic model where task execution times and migration overheads are dependent on the real-time system state rather than static values.

8 Acknowledgement

This research was partly supported by the National Key Research and Development Program of China under Grant 2022YFF0902800, the National Natural Science Foundation of China under Grant 62533019, and the Natural Science Foundation of Jiangsu Province under Grant BK20242028.

References

[1] Sohaib Iftikhar Abbasi, Shaharyar Kamal, Munkhjargal Gochoo, Ahmad Jalal, and Kibum Kim. 2021. Affinity-based task scheduling on heterogeneous multicore systems using CBS and QBICTM. *Applied Sciences* 11, 12 (2021), 5740.

[2] Daniel Y. Abramovitch, Sean Andersson, Kam K. Leang, William Nagel, and Shalom Ruben. 2023. A tutorial on real-time computing issues for control systems. In *American Control Conference*. 3751–3768.

[3] Amjad Ali, Asad Masood Khattak, Shahid Iqbal, Omar Alfandi, Bashir Hayat, Muhammad Hameed Siddiqi, and Adil Khan. 2023. Overhead based cluster scheduling of mixed criticality systems on multicore platform. *IEEE Access* 11 (2023), 142341–142359.

[4] Gabriel Marchesan Almeida, Sameer Varyani, Rémi Busseuil, Gilles Sassatelli, Pascal Benoit, Lionel Torres, Everton Alceu Carara, and Fernando Gehm Moraes. 2010. Evaluating the impact of task migration in multi-processor systems-on-chip. In *ACM Symposium on Integrated Circuits and System Design*. 73–78.

[5] Arm. 2025. *Arm DynamIQ technology overview*. Technical Report. https://www.arm.com/technologies/dynamiq

[6] Ashikahmed Bhuiyan, Di Liu, Aamir Khan, Abusayeed Saifullah, Nan Guan, and Zhishan Guo. 2020. Energy-efficient parallel real-time scheduling on clustered multi-core. *IEEE Transactions on Parallel and Distributed Systems* 31, 9 (2020), 2097–2111.

[7] Anastasiia Butko, Florent Bruguier, Abdoulaye Gamatié, Gilles Sassatelli, David Novo, Lionel Torres, and Michel Robert. 2016. Full-system simulation of big.LITTLE multicore architecture for performance and energy exploration. In *IEEE International Symposium on Embedded Multicore/Many-core Systems-on-Chip*. 201–208.

[8] Shounak Chakraborty, Sangeet Saha, Magnus Själander, and Klaus Mcdonald-Maier. 2021. Prepare: Power-aware approximate real-time task scheduling for energy-adaptive QoS maximization. *ACM Transactions on Embedded Computing Systems* 20, 5s (2021), 25 pages.

[9] Gang Chen, Kai Huang, and Alois Knoll. 2014. Energy optimization for real-time multiprocessor system-on-chip with optimal DVFS and DPM combination. *ACM Transactions on Embedded Computing Systems* 13 (2014), 1–21.

[10] Mei-Ling Chiang, Shu-Wei Tu, Wei-Lun Su, and Chen-Wei Lin. 2018. Enhancing inter-node process migration for load balancing on Linux-based NUMA multicore systems. In *IEEE Annual Computer Software and Applications Conference*. 394–399.

[11] Hoon Sung Chwa, Jaebaek Seo, Hyuck Yoo, Jinkyu Lee, and Insik Shin. 2013. Energy and feasibility optimal global scheduling framework on big.LITTLE platforms. In *IEEE International Real-Time Scheduling Open Problems Seminar*. 770–777.

[12] Minyu Cui, Angeliki Kritikakou, Lei Mo, and Emmanuel Casseau. 2021. Fault-tolerant mapping of real-time parallel applications under multiple DVFS schemes. In *IEEE Real-Time and Embedded Technology and Applications Symposium*. 387–399.

[13] Juan Fang, Jiaxing Zhang, Shuaibing Lu, and Hui Zhao. 2020. Exploration on task scheduling strategy for CPU-GPU heterogeneous computing system. In *IEEE Computer Society Annual Symposium on VLSI*. 306–311.

[14] FreeRTOS 2025. *Symmetric Multiprocessing (SMP) with FreeRTOS*. FreeRTOS. https://freertos.org/Documentation/02-Kernel/02-Kernel-features/13-Symmetric-multiprocessing-introduction#smp-specific-apis

[15] Sharma Gajendra and Prashant Poudel. 2022. Current trends in heterogeneous systems: A review. *Trends in Computer Science and Information Technology* 7 (2022), 086–090.

[16] Aymen Gammoudi, Adel BenZina, Mohamed Khalgui, and Daniel Chillet. 2020. Energy-efficient scheduling of real-time tasks in reconfigurable homogeneous multicore platforms. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 50, 12 (2020), 5092–5105.

[17] Jingyu He, Yao Xiao, Corina Bogdan, Shahin Nazarian, and Paul Bogdan. 2021. A design methodology for energy-aware processing in unmanned aerial vehicles. *ACM Transactions on Design Automation of Electronic Systems* 27 (2021), 1–20.

[18] Huang Huang, Gang Quan, Jeffrey Fan, and Meikang Qiu. 2011. Throughput maximization for periodic real-time systems under the maximal temperature constraint. In *ACM/IEEE Design Automation Conference*. 363–368.

[19] Ramkumar Jayaseelan and Tulika Mitra. 2008. Temperature aware task sequencing and voltage scaling. In *IEEE/ACM International Conference on Computer-Aided Design*. 618–623.

[20] Bartłomiej Kocot, Pawel Czarnul, and Jerzy Proficz. 2023. Energy-aware scheduling for high-performance computing systems: A survey. *Energies* 16 (2023), 890.

[21] Dawei Li and Jie Wu. 2015. Minimizing energy consumption for frame-based tasks on heterogeneous multiprocessor platforms. *IEEE Transactions on Parallel and Distributed Systems* 26, 3 (2015), 810–823.

[22] Xinmei Li, Lei Mo, Angeliki Kritikakou, and Olivier Sentieys. 2023. Approximation-aware task deployment on heterogeneous multicore platforms with DVFS. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 42, 7 (2023), 2108–2121.

[23] MediaTek. 2015. *MediaTek CorePilot 3.0: Tri-cluster CPU architecture and heterogeneous computing scheduling*. Technical Report. https://www.mediatek.com/hubfs/MediaTek%20Assets/Pdfs/White_Papers/MediaTek_CorePilot_3.0.pdf

[24] Lei Mo, Angeliki Kritikakou, and Olivier Sentieys. 2019. Approximation-aware task deployment on asymmetric multicore processors. In *Design, Automation & Test in Europe Conference & Exhibition*. 1513–1518.

[25] Takashi Nakada, Hiroyuki Yanagihashi, Kunimaro Imai, Hiroshi Ueki, Takashi Tsuchiya, Masanori Hayashikoshi, and Hiroshi Nakamura. 2020. An energy-efficient task scheduling for near real-time systems on heterogeneous multicore processors. *IEICE Transactions on Information and Systems* E103.D (2020), 329–338.

[26] Francesco Percassi and Sangeet Saha. 2022. QoS-aware approximate real-time tasks execution on multiprocessor systems using PDDL+ planning. In *International Conference on Automated Planning and Scheduling*. 1–6.

[27] Michele Pittau, Andrea Alimonda, Salvatore Carta, and Andrea Acquaviva. 2007. Impact of task migration on streaming multimedia for embedded multiprocessors: A quantitative evaluation. In *IEEE/ACM/IFIP Workshop on Embedded Systems for Real-Time Multimedia*. 59–64.

[28] Behnaz Pourmohseni, Fedor Smirnov, Stefan Wildermann, and Jürgen Teich. 2020. Real-time task migration for dynamic resource management in many-core systems. In *Workshop on Next Generation Real-Time Embedded Systems*. 5:1–5:14.

[29] Abhishek Roy, Hakan Aydin, and Dakai Zhu. 2021. Energy-aware primary/backup scheduling of periodic real-time tasks on heterogeneous multicore systems. *Sustainable Computing: Informatics and Systems* 29 (2021), 100474.

[30] Sangeet Saha, Shounak Chakraborty, Xiaojun Zhai, Shoaib Ehsan, and Klaus D. McDonald-Maier. 2022. ACCURATE: Accuracy maximization for real-time multicore systems with energy-efficient way-sharing caches. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41, 12 (2022), 5246–5260.

[31] Bagher Salami, Mohammadreza Baharani, and Hamid Noori. 2014. Proactive task migration with a self-adjusting migration threshold for dynamic thermal management of multi-core processors. *Journal of Supercomputing* 68, 3 (2014), 1068–1087.

[32] Tongquan Wei, Junlong Zhou, Kun Cao, Peijin Cong, Mingsong Chen, Gongxuan Zhang, Xiaobo Sharon Hu, and Jianming Yan. 2018. Cost-constrained QoS optimization for approximate computation real-time tasks in heterogeneous MPSoCs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37, 9 (2018), 1733–1746.

[33] Shengyan Wen, Xiaohang Wang, Amit Kumar Singh, Yingtao Jiang, and Mei Yang. 2022. Performance optimization of many-core systems by exploiting task migration and dark core allocation. *IEEE Trans. Comput.* 71, 1 (2022), 92–106.

[34] Saehanseul Yi, Tae-Wook Kim, Jong-Chan Kim, and Nikil Dutt. 2023. EASYR: Energy-efficient adaptive system reconfiguration for dynamic deadlines in autonomous driving on multicore processors. *ACM Transactions on Embedded Computing Systems* 22 (2023), 1–29.

[35] Heng Yu, Yajun Ha, and Bharadwaj Veeravalli. 2012. Quality-driven dynamic scheduling for real-time adaptive applications on multiprocessor systems. *IEEE Trans. Comput.* 62, 10 (2012), 2026–2040.

[36] Heng Yu, Yajun Ha, Bharadwaj Veeravalli, Fupeng Chen, and Hesham El-Sayed. 2021. DVFS-based quality maximization for adaptive applications with diminishing return. *IEEE Trans. Comput.* 70, 5 (2021), 803–816.

[37] Junlong Zhou, Jianming Yan, Tongquan Wei, Mingsong Chen, and Xiaobo Sharon Hu. 2017. Energy-adaptive scheduling of imprecise computation tasks for QoS optimization in real-time MPSoC systems. In *Design, Automation & Test in Europe Conference & Exhibition*. 1402–1407.