Optimal and Energy Efficient Imprecise Computation Task Deployment on Heterogeneous Multicore Platforms Combining DVFS and DPM

HAOTONG ZHU, School of Automation, Southeast University, Nanjing, China
LEI MO, School of Automation, Southeast University, Nanjing, China
TAMIM AL-HASAN, School of Computer Science and Electronic Engineering, University of Essex, Colchester, United Kingdom of Great Britain and Northern Ireland
ANGELIKI KRITIKAKOU, IRISA, Rennes, France, Computer Science and Electrical Engineering, University of Rennes, Rennes, France, INRIA, Le Chesnay, France, and CNRS, Paris, France
XIAOJUN ZHAI, School of Computer Science and Electronic Engineering, University of Essex, Colchester, United Kingdom of Great Britain and Northern Ireland
QING GAO, School of Automation Science and Electrical Engineering, Beihang University, Beijing, China
SHIBO HE, College of Control Science and Engineering, Zhejiang University, Hangzhou, China

Heterogeneous multicore embedded systems can handle complex tasks such as data collection, processing, and wireless communication. However, these systems often face constraints due to limited computational resources, requiring efficient task deployment strategies to manage dependent and real-time tasks. Despite advancements, existing methods rarely address the deployment of Imprecise Computation (IC) tasks on heterogeneous multicore platforms that utilize both Dynamic Voltage/Frequency Scaling (DVFS) and Dynamic Power Management (DPM). This article introduces a novel QoS-aware task deployment methodology to enhance the Quality of Service (QoS) under resource limitations. The proposed approach simultaneously optimizes frequency assignment, processor state transitions, task allocation, scheduling, and task execution cycle adjustments to maximize system QoS while adhering to time, energy, and dependency constraints. The problem is initially formulated as a Mixed Integer Non-Linear Programming (MINLP) model, then transformed into a Mixed Integer Linear Programming (MILP) formulation to obtain optimal solutions. We developed a heuristic method with low computational complexity to address scalability challenges, employing problem decomposition and a greedy algorithm. The process operates in two stages: task allocation and frequency

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 Grants 62533019 and 92367204, and the Natural Science Foundation of Jiangsu Province under Grant BK20242028.
Authors' Contact Information: Haotong Zhu, School of Automation, Southeast University, Nanjing, Jiangsu, China; e-mail: 220232114@seu.edu.cn; Lei Mo (corresponding author), School of Automation, Southeast University, Nanjing, Jiangsu, China; e-mail: lmo@seu.edu.cn; Tamim Al-Hasan, School of Computer Science and Electronic Engineering, University of Essex, Colchester, England, United Kingdom of Great Britain and Northern Ireland; e-mail: tamim.almulla@essex.ac.uk; Angeliki Kritikakou, IRISA, Rennes, Brittany, France, Computer Science and Electrical Engineering, University of Rennes, Rennes, Brittany, France, INRIA, Le Chesnay, Île-de-France, France, and CNRS, Paris, Île-de-France, France; e-mail: angeliki.kritikakou@irisa.fr; Xiaojun Zhai, School of Computer Science and Electronic Engineering, University of Essex, Colchester, England, United Kingdom of Great Britain and Northern Ireland; e-mail: xzhai@essex.ac.uk; Qing Gao, School of Automation Science and Electrical Engineering, Beihang University, Beijing, Beijing, China; e-mail: gaoqing@buaa.edu.cn; Shibo He, College of Control Science and Engineering, Zhejiang University, Hangzhou, Zhejiang, China; e-mail: s18he@zju.edu.cn.



This work is licensed under a Creative Commons Attribution 4.0 International License.

© 2027 Copyright held by the owner/author(s).
ACM 1084-4309/2027/03-ART44
https://doi.org/10.1145/3821440

assignment, followed by task scheduling and cycle adjustment. The simulation results demonstrate that the proposed method achieves superior system performance compared to existing approaches, particularly in QoS (41.30%), task schedulability (58.67%), and execution efficiency.

CCS Concepts: • **Computer systems organization → Real-time systems**; • **Computing methodologies → Parallel computing methodologies;**

Additional Key Words and Phrases: Heterogeneous platform, task deployment, DVFS, DPM, IC task, QoS

ACM Reference Format:
Haotong Zhu, Lei Mo, Tamim Al-Hasan, Angeliki Kritikakou, Xiaojun Zhai, Qing Gao, and Shibo He. 2027. Optimal and Energy Efficient Imprecise Computation Task Deployment on Heterogeneous Multicore Platforms Combining DVFS and DPM. *ACM Trans. Des. Autom. Electron. Syst.* 32, 2, Article 44 (March 2027), 32 pages. https://doi.org/10.1145/3821440

1 Introduction

High-performance embedded systems have a wide range of applications [35], such as mobile devices and **Unmanned Aerial Vehicles** (**UAVs**) [19, 45]. By mapping (allocating and executing) the applications on the embedded systems, the desired performances, e.g., IoT federated learning and UAV flight control, can be achieved. The heterogeneous multicore platforms, such as ARM big.LITTLE processors [41] are energy-efficient platforms to satisfy the complex demand of applications, as the big cores and LITTLE cores have different characteristics, i.e., the big cores have higher computation performance. In comparison, LITTLE cores offer better energy efficiency. Energy consumption and real-time response are two critical metrics for task execution on multicore platforms. On one hand, multicore platforms are generally energy-restricted, especially for battery-powered devices, e.g., mobile devices and UAVs, and high energy consumption will reduce the lifetime of devices and cause critical failure and thermal issues [7]. On the other hand, the applications usually contain a series of real-time tasks and must be finished within the given deadlines [19], the tasks that violate real-time requirements will cause serious consequences, e.g., the attitude control of UAVs.

Dynamic Voltage and Frequency Scaling (**DVFS**) technology [13] can dynamically adjust processors' operating frequency and supply voltage to improve task execution efficiency. By enhancing the **Voltage/Frequency** (**V/F**) level of task execution, task execution time can be shortened while task execution energy increases. To balance task execution time and energy, a suitable V/F level should be selected using DVFS [25]. Note that DVFS is primarily used to adjust processor power consumption during task execution. The processors enter an idle or sleep state when the assigned tasks are finished. **Dynamic Power Management** (**DPM**) technology [36] can further optimize processor energy consumption in idle or sleep states. Therefore, combining DVFS and DPM can optimize processor states during the scheduling horizon to enhance energy efficiency. Additionally, the **Imprecise Computation** (**IC**) model [47] can be used to describe the tasks executed on multicore platforms under strict energy and time constraints, e.g., the control and optimization problems, as the accuracy of the control result and the quality of the problem solution depend on the number of algorithm iterations. In several application domains, an approximate – but still in time – result is preferable to an exact result obtained too late. An IC task can be logically decomposed into a mandatory subtask and an optional subtask, where the compulsory subtask must be fully executed before the deadline to obtain an acceptable approximate computation result (basic task QoS). In contrast, the optional subtask can remain unexecuted at the cost of reducing task QoS [5].

By mapping IC tasks onto heterogeneous multicore platforms using DVFS and DPM, we can better utilize system resources to improve task QoS under multiple constraints, such as **energy supply** (**ES**), real-time response, and task dependencies. Although some existing works consider

task deployment problems for multicore platforms with DVFS and/or DPM [1, 8, 14, 17, 22–24, 26, 28, 31, 34, 40, 42, 46]; most of them target homogeneous multicore platforms [8, 14, 17, 22–24, 28, 46]; moreover, they consider precise computation tasks [1, 8, 14, 17, 22–24, 31, 34, 40, 42, 46], i.e., the task execution cycle number is fixed. Thus, the improvement of task QoS is limited. Compared with existing works, we jointly optimized the allocation and execution of dependent and real-time IC tasks on heterogeneous multicore platforms through DVFS and DPM to enhance system QoS under energy and real-time constraints. Our main contributions can be summarized as follows:

(1) We address the task deployment problem for heterogeneous multicore platforms executing IC tasks with DVFS and DPM enabled. The problem involves optimizing **task allocation** (**TA**), scheduling, **frequency assignment** (**FA**), optional cycle adjustment, and processor state transitions to maximize system QoS while meeting constraints such as ES, real-time response, and task dependencies. This complex joint design is formulated as a mixed-integer non-linear programming problem. To obtain optimal solutions, we reformulate it as a mixed-integer linear programming model by introducing auxiliary variables and additional constraints to replace the nonlinear terms without degrading solution quality.
(2) To improve scalability, we propose a heuristic method leveraging decomposition, greedy policy, and iterative feedback. The original problem is split into two subproblems: (1) TA and FA, and (2) **task scheduling** (**TS**) and optional cycle adjustment. A low-complexity greedy algorithm solves the first problem, while the second problem iteratively adjusts V/F levels and optional cycles to refine the solution.
(3) Extensive experiments demonstrate that the proposed method surpasses existing approaches in task QoS (41.30%), schedulability (58.67%), and execution efficiency. Compared to the optimal method, the heuristic reduces computation time by 99.9% while incurring only a 28.18% QoS loss.

The remaining part of the article is organized as follows. Section 2 presents the relevant literature review and motivates the task deployment problem through an example. Section 3 introduces the system model, and Section 4 mathematically formulates the target problem. Section 5 presents the proposed heuristic. Section 6 shows the evaluation results, Section 7 discusses the extensions, and Section 8 concludes this article.

2 Related Work

Table 1 compares representative works on task deployment for multicore platforms, categorized by task types, platform characteristics, and optimization strategies. Tasks are classified as Independent (I) or Dependent (D), and Precise (P) or Imprecise (IP), based on whether execution cycles are fixed or adjustable. Platforms are categorized into Homogeneous (HO) or Heterogeneous (HE) based on the uniformity of core capabilities. Task deployment leverages techniques like DVFS and/or DPM for efficient energy management, handling FA, Processor **State Selection** (**SS**), TA, and TS. The deployment of tasks must satisfy constraints such as Real-Time (RT) deadlines and ES limits while achieving objectives like **Energy-Aware** (**EA**) or **QoS-Aware** (**QA**) optimization. The categorization in Table 1 highlights the limitations of existing methods, emphasizing the need for a comprehensive approach that addresses dependent and imprecise tasks, heterogeneous platforms, and integrated DVFS and DPM for enhanced performance, detailed as follows:

Task Deployment with DVFS: DVFS-based approaches optimize energy consumption, execution time, and QoS. Representative works include [24, 26, 28, 31, 40, 42]. The study [24] focuses on real-time tasks on heterogeneous multicore platforms, optimizing TA and clustered DVFS to minimize energy under time constraints. Similarly, [42] introduces an energy-efficient V/F scaling framework for heterogeneous platforms using model-free constrained reinforcement learning. However, these

Table 1. Task Deployment Methods

References	*Tasks*				*Platforms*		*Variables*				*Constraints*		*Objective*	
	D	ID	P	IP	HO	HE	FA	SS	TA	TS	RT	ES	QA	EA
[24]		✓	✓		✓		✓		✓		✓			✓
[42]		✓	✓			✓	✓		✓		✓			✓
[31]	✓		✓			✓	✓		✓	✓	✓			✓
[40]	✓		✓			✓	✓		✓	✓	✓			✓
[28]		✓		✓	✓		✓		✓		✓	✓	✓	
[26]	✓			✓		✓	✓		✓	✓	✓	✓	✓	
[22]		✓	✓		✓			✓	✓	✓	✓			✓
[23]		✓	✓		✓			✓	✓	✓	✓			✓
[46]		✓	✓		✓			✓	✓	✓	✓			✓
[8]		✓	✓		✓		✓	✓		✓	✓			✓
[14]		✓	✓		✓		✓	✓	✓	✓	✓			✓
[17]	✓		✓		✓		✓	✓	✓	✓	✓			✓
[1]		✓	✓			✓	✓	✓	✓	✓	✓			✓
[34]	✓		✓			✓	✓	✓	✓	✓	✓			✓
Proposed	✓			✓		✓	✓	✓	✓	✓	✓	✓	✓	

methods primarily address independent tasks, neglecting task dependencies. Optimizing dependent real-time tasks is considered in [31] and [40], which combine DVFS, task allocation, and scheduling to reduce energy consumption. Despite these advancements, the focus remains on precise tasks, limiting applicability to ICs. To address this, both [28] and [26] explore IC tasks to enhance QoS under strict constraints. Ref. [28] employs a decomposition-based method combining DVFS and an elastic task model to optimize QoS, execution time, and energy. Ref. [26], on the other hand, incorporates task dependencies and combines DVFS with task migration, allowing tasks to shift between cores with varying V/F levels and efficiencies, improving flexibility and resource utilization.

Task Deployment with DPM: DPM is extensively used to optimize energy consumption and minimize idle time on multicore platforms [22, 23, 46]. Time-out heuristic policies, as proposed in [22] and [23], transition processors into low-power states during inactivity. In [23], task allocation and scheduling maximize sleep state durations to improve energy efficiency, while [22] employs task migration to balance task sizes and time slots, optimizing idle and sleep intervals. Extending these deterministic approaches, [46] introduces a stochastic, learning-based DPM that uses probabilistic policies for dynamic state transitions, adapting to varying workloads and system conditions. However, these methods primarily address independent real-time tasks and homogeneous multicore platforms, limiting their applicability to heterogeneous, more complex scenarios.

Task Deployment with DVFS and DPM: Joint optimization of DVFS and DPM provides a more comprehensive approach to balancing energy consumption and performance under multiple constraints [1, 8, 14, 17, 34]. For example, [8] explores the scheduling of real-time dependent tasks on homogeneous multicore platforms, where idle interval lengths are optimized using DVFS and DPM techniques. In [14] and [17], task allocation and scheduling are integrated with DVFS and DPM to minimize overall energy consumption. Specifically, [14] addresses mixed task sets comprising independent hard and soft real-time tasks, while [17] focuses on dependent real-time tasks. Despite these advancements, both studies are limited to homogeneous multicore platforms. Heterogeneous multicore platforms are considered in [1] and [34], where DVFS and DPM jointly optimize dynamic and static energy consumption. Compared to [1], which primarily addresses TS and energy consumption, [34] incorporates task dependencies into the optimization process, enabling more

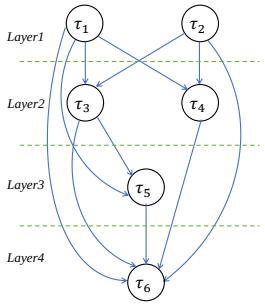


Fig. 1. Dependent task dag.

efficient resource utilization and execution in heterogeneous environments. These methods target precise computation tasks with fixed execution cycles. The given QoS is achieved when tasks finish within the time threshold.

Besides [28] and [26], there are also some works that focus on the IC-task deployment problem, e.g., [12, 27, 49], with the aim to maximize system QoS under multiple constraints. Compared with the works considering independent tasks [28], single-core platforms [12], homogeneous platforms [27], without DVFS [49] and DPM [26], this article focuses on the multi-dimensional tug-of-war between active power reduction (DVFS) and static power reduction (DPM) within an elastic execution framework (IC) for the heterogeneous multicore platforms. Note that the optional parts of IC tasks are adjustable, and the V/F levels and ranges of processors in heterogeneous platforms are different. Task allocation and scheduling, and frequency and state adjustments are coupled with each other, which are restricted by dependency, energy, and time constraints.

Toh ighlight the motivation, we showcase the example illustrated in Figures 1 and 2 to define the target problem and motivate the proposed method. We consider six dependent and real-time IC tasks $\{\tau_1, \ldots, \tau_6\}$. Tasks are allocated to a heterogeneous multicore platform with two processors, θ_1 and θ_2, that support DVFS and DPM, and the platform has an energy budget of $E_b = 1.17$ J. Based on the length of the task critical path, the scheduling horizon is set to $H = 1.23$ s, and the task relative deadlines are set to $D_1 = 0.33$ s, $D_2 = 0.29$ s, $D_3 = 0.20$ s, $D_4 = 0.38$ s, $D_5 = 0.27$ s, and $D_6 = 0.43$ s. We adopt the cumulative executed optional cycles as the system QoS. Figure 2 shows the task deployment results implemented by the methods with DVFS and DPM (OPT), DVFS (TDV), DPM (TDP), and fixed task-to-core allocation (FTA), where TDP considers the fixed processor operating frequency. The system QoS are 9.2422×10^8, 6.8704×10^8, 9.058×10^8, and 8.0589×10^8 (cycles), respectively, under the same time and energy constraints.

Firstly, we can use DVFS to increase task execution frequency compared to the fixed-frequency method, allowing more optional cycles to be executed within the same period. For instance, with TDP, the execution time of task τ_1 is 0.33 s, which is within the time range D_1, and 3.1973×10^8 optional cycles can be executed with the fixed frequency f_{13}, as shown in Figure 2(c). Since OPT combines DVFS, it uses optimized (higher) frequency f_{14} to execute τ_1, and thus, more optional cycles 3.9841×10^8 can be executed, as shown in Figure 2(a).

Secondly, with DPM, when no task is executing, the processor state can be switched to reduce energy consumption. In Figure 2(b), during the period $[0.53, 1.23]$, the processor θ_1 keeps the idle state through TDV, while in Figure 2(a), the processor θ_1 enters the sleep state according to OPT, as it combines DPM, which can reduce 0.2378 J energy. By utilizing the additional energy resource, OPT can execute more optional subtasks than TDV within the same time and energy constraints.

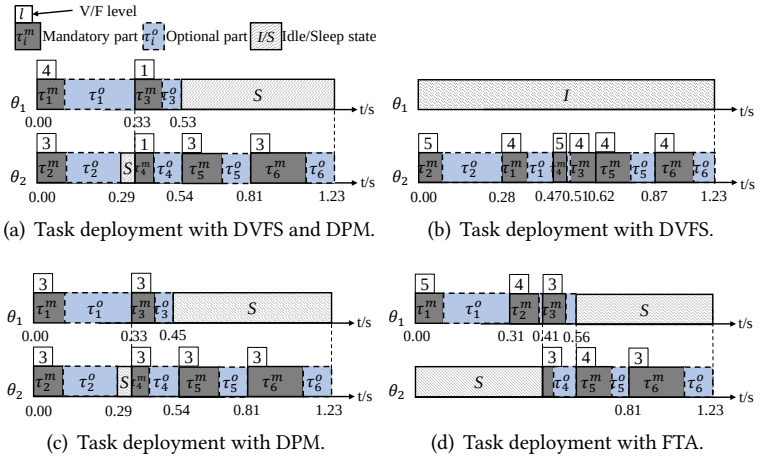


Fig. 2. Task deployment comparisons under different schemes.

Due to the heterogeneity of the multicore platform and dependency of the tasks, an appropriate task-to-core allocation can also enhance the task execution efficiency, e.g., reduce task execution time and energy. For example, task τ_2 is allocated to processor θ_1 and θ_2 by FTA (Figure 2(d)) and OPT (Figure 2(a)), respectively. Compared with θ_1, when τ_2 is allocated to θ_2, the execution efficiency index $\lambda_{22} = 0.7659 > \lambda_{21} = 0.6376$, indicating this TA requires less task execution energy and time. Moreover, τ_2 can be executed in parallel with τ_1 when τ_2 is allocated to θ_2. In contrast, τ_1 and τ_2 can be executed only in sequence when allocated to θ_1, thereby degrading task execution efficiency. In summary, by applying DVFS and DPM to the heterogeneous multicore platform, we can better utilize system resources to improve system QoS under multiple constraints.

3 System Model

3.1 Task Model

We consider a task set including N real-time, non-preemptive, and periodic IC tasks $\mathcal{T} = \{\tau_1, \ldots, \tau_N\}$. These tasks are represented by a **Directed Acyclic Graph** (**DAG**) $G(T, E)$, where vertices T denote tasks and edges E denote data dependencies. Task τ_i executes only after receiving data from all predecessors, and its results are available to successors upon completion. For the tasks with different periods, they can be involved in the hyper-period (i.e., the least common multiple of all task periods) to construct $G(T, E)$. Each task τ_i is characterized by a tuple $\{M_i, o_i, O_i, A_i, D_i, t_i^s, t_i^e\}$, where M_i and o_i represent the mandatory and optional cycles, respectively, and they are measured by the **Worst-Case Execution Cycles** (**WCEC**). The optional cycles are adjustable to an upper bound O_i ($0 \le o_i \le O_i$). A_i denotes the arrival time of task τ_i, which is smaller than its start time t_i^s, i.e., $t_i^s \ge A_i$. D_i is task relative deadline [28], with $t_i^e - t_i^s \le D_i$, and t_i^e is the end time of task τ_i. The scheduling horizon H ensures that each task τ_i is completed within $t_i^e \le H$, and is fixed.

The mandatory subtask ensures basic QoS, while optional subtasks improve QoS within the same deadline. The optional subtask is executed after the mandatory subtask. Higher execution of optional cycles results in better QoS, modeled using a QoS function. Linear and concave QoS functions are widely used in applications such as image processing, speech processing, control systems, and target tracking [2]. Here, we adopt a linear QoS function $f_i(o_i) = k_i o_i + R_i$, where k_i is a constant, and R_i is the basic QoS achieved after completing M_i cycles.

3.2 Energy Model

We consider a multicore platform with M heterogeneous processors $\{\theta_1, \ldots, \theta_M\}$, each supporting DVFS. For each processor θ_k, there are l_k discrete V/F levels, $\mathcal{VF}_k = \{(v_1, f_1), \ldots, (v_{l_k}, f_{l_k})\}$, sorted in ascending order. The relationship between voltage v_l and frequency f_l is approximately linear [13], meaning changes in frequency also alter voltage. To handle varying l_k across processors, we set $L = \max_{1 \le k \le M}\{l_k\}$ and extend matrices for processors with fewer V/F levels by repeating their maximum v and f values.

A binary variable $q_{ikl} = 1$ indicates that task τ_i is assigned to processor θ_k and executed at V/F level (v_{kl}, f_{kl}); otherwise, $q_{ikl} = 0$. For processor heterogeneity, we define an execution efficiency index $\lambda_{ik} \in (0, 1]$ [25], representing the efficiency of processor θ_k executing task τ_i. The computation time for τ_i with $M_i + o_i$ cycles on θ_k at V/F level (v_{kl}, f_{kl}) is $\frac{M_i+o_i}{f_{kl}\lambda_{ik}}$, which can be expressed as $\sum_{k=1}^{M}\sum_{l=1}^{L} q_{ikl}\frac{M_i+o_i}{f_{kl}\lambda_{ik}}$. In practice, λ_{ik} is measured empirically by executing the same task on each processor at each V/F level and recording execution cycles [10]. If $\lambda_{ik} = 1$ and all the processors have the same characteristics, e.g., the V/F level, our method can be applied to the homogeneous platforms.

Processor power consumption at V/F level (v_{kl}, f_{kl}) [28] is given by

$$P_{kl}^{c} = P_{kl}^{s} + P_{kl}^{d} = C_k^{s} v_{kl}^{\rho^k} + C_k^{d} v_{kl} f_{kl}^2, \tag{1}$$

where P_{kl}^{s} and P_{kl}^{d} represent static and dynamic power, and C_k^{s}, C_k^{d}, and ρ^k depend on processor type. A processor θ_k has three states: *Active*, *Idle*, and *Sleep* [8]. In the active state, dynamic power dominates ($P_{kl}^{active} = P_{kl}^{c}$), while idle and sleep states consume idle power P_k^{idle} and sleep power P_k^{sleep}, respectively, with $P_k^{sleep} < P_k^{idle}$ [1].

3.3 DPM Model

Based on the active power P_{kl}^{active}, the energy consumed by processor θ_k at V/F level (v_{kl}, f_{kl}) to execute task τ_i with $M_i + o_i$ cycles is $\frac{M_i+o_i}{f_{kl}\lambda_{ik}} P_{kl}^{active}$. During a time interval T when θ_k is idle or in sleep state, the corresponding energy consumption is TP_k^{idle} or TP_k^{sleep}, respectively. While the sleep state consumes less power than the idle state, switching states incurs additional time and energy overheads [1]. To simplify the analysis, we convert time overhead into energy [8]. When the idle period T exceeds a threshold, the energy consumption in the idle state exceeds the switching energy, making the sleep state more energy efficient. The break-even time T_k^{BET} represents the minimum interval T required for energy savings during state switching, which can be calculated as

$$T_k^{BET} = \max\left\{t_k^{sw}, \frac{E_k^{sw} - P_k^{sleep} t_k^{sw}}{P_k^{idle} - P_k^{sleep}}\right\}. \tag{2}$$

In Equation (2), t_k^{sw} and E_k^{sw} are the time and energy required for state switching. If θ_k is idle for the time interval T, the energy consumed in idle state is TP_k^{idle}, while switching to sleep state incurs $(T - t_k^{sw})P_k^{sleep} + E_k^{sw}$. If $T \ge T_k^{BET}$, switching to a sleep state is more energy efficient. Conversely, switching requires t_k^{sw} time; $T \ge t_k^{sw}$ ensures sufficient time for the state transition. Thus, $T \ge T_k^{BET}$ indicates that the sleep state is preferable. Efficient DPM relies on accurately calculating T_k^{BET}, which depends on task allocation and scheduling. Specifically, T_k^{BET} occurs between adjacent tasks assigned to θ_k. For tasks τ_i and τ_j assigned to θ_k, where τ_j starts immediately after τ_i, the gap

$T = t_j^s - t_i^e$ must satisfy $T \geq T_k^{BET}$. If this condition is met, θ_k enters sleep after completing τ_i and wakes up when τ_j begins. Compared to remaining idle, this approach reduces energy consumption.

We introduce a binary variable c_i for task τ_i to implement the DPM mechanism. If $c_i = 1$, θ_k enters sleep state between τ_i and its successor; if $c_i = 0$, it remains idle. Start and end times of tasks, t_i^s and t_i^e, are influenced by task allocation and scheduling, which are optimization variables in this study. For the scheduling horizon H, θ_k is initially in a sleep state before its first task and transitions back to sleep after completing its last task. To model this process, two virtual tasks, VTS_k and VTE_k, represent the entry and exit states for θ_k. These virtual tasks, with zero execution cycles, depend on assigned tasks to capture the full state-switching process. Extending the task graph $G(T, E)$ with VTS_k and VTE_k, we construct a new task graph $G'(T', E')$ that models state-switching dynamics.

4 Problem Formulation

The task deployment problem aims to maximize system QoS under multiple constraints from both tasks and multicore platforms, such as task dependencies, real-time response requirements, and ES constraints. Therefore, we need to determine the decisions of (1) TA, (2) FA, (3) TS, (4) optional subtask adjustment, and (5) state switching. To formulate the target problem, we introduce the following binary and continuous variables: (1) $q_{ikl} = 1$, if task τ_i is assigned to processor θ_k and executed with the V/F level (v_{kl}, f_{kl}), otherwise, $q_{ikl} = 0$; (2) $p_{ijk} = 1$, if τ_i and τ_j are assigned to θ_k and τ_i precedes τ_j, otherwise, $p_{ijk} = 0$; (3) $c_i = 1$, if the processor switches from the idle state to the sleep state after executing τ_i; otherwise, $c_i = 0$; 4) continuous variables t_i^s and o_i denote the start time and the optional cycles of τ_i, respectively. During the problem formulation, o_i is considered a continuous variable. When the task deployment problem is solved, we round the result down. The impact of one cycle is negligible, as the tasks typically execute hundreds of thousands of cycles [28, 44]. The main symbols used in the problem formulation are summarized in Table 2. For the sake of article presentation, let $\mathcal{N} \triangleq \{1, \ldots, N\}$, $\mathcal{M} \triangleq \{1, \ldots, M\}$, and $\mathcal{L} \triangleq \{1, \ldots, L\}$. Note that each processor θ_k has an entry task VTS_k and an exit task VTE_k. Hence, we have N original tasks and $2M$ virtual tasks. Let $\mathcal{N}' \triangleq \{1, \ldots, 2M + N\}$. Figure 3 shows the structure of our task deployment problem, and the constraints are as follows.

4.1 Task Allocation and Frequency Assignment Constraints

Without task migration [48] and with task-level DVFS [13, 25], since each task τ_i is assigned to one processor and τ_i starts and ends its execution using one V/F level, the variable q_{ikl} regarding TA and FA is bounded by

$$\sum_{k \in \mathcal{M}} \sum_{l \in \mathcal{L}} q_{ikl} = 1, \ \forall i \in \mathcal{N}'. \tag{3}$$

Note that the value of q_{ikl} is also related to the task-to-core allocation. We introduce an auxiliary (binary) variable x_{ik}, where $x_{ik} = 1$ represents that task τ_i is assigned to processor θ_k, otherwise, $x_{ik} = 0$. Based on q_{ikl}, we have $x_{ik} = \sum_{l \in \mathcal{L}} q_{ikl}$, $\forall i \in \mathcal{N}$.

4.2 Task Real-Time Constraints

Since the execution time of task τ_i cannot exceed the relative deadline D_i, and all the tasks should be finished within the scheduling period H [28], we have the following time constraints:

$$t_i^c = \sum_{k \in \mathcal{M}} \sum_{l \in \mathcal{L}} q_{ikl} \frac{M_i + o_i}{\lambda_{ik} f_{kl}} \leq D_i, \ \forall i \in \mathcal{N}', \tag{4}$$

$$t_i^e \leq H, \ \forall i \in \mathcal{N}', \tag{5}$$

Table 2. Main Notations and Optimization Variables

Parameters	
N	number of tasks
M	number of processors
τ_i	the i^{th} task
θ_k	the k^{th} processor
f_{kl}	the l^{th} frequency level of θ_k
l_k	number of frequency levels in θ_k
L	maximum l_k of processors
H	scheduling horizon
M_i	number of mandatory subtask cycles
O_i	number of optional subtask cycles
D_i	relative deadline of τ_i
λ_{ik}	execution efficiency of task τ_i on processor θ_k
s_{ij}	$= \begin{cases} 1 & \tau_j \text{ depends on } \tau_i \\ 0 & \text{otherwise} \end{cases}$
Binary Variables	
q_{ikl}	$= \begin{cases} 1 & \tau_i \text{ is executed on } \theta_k \text{ at V/F level } (v_{kl}, f_{kl}) \\ 0 & \text{otherwise} \end{cases}$
p_{ijk}	$= \begin{cases} 1 & \tau_i \text{ is the predecessor of } \tau_j \text{ on } \theta_k \\ 0 & \text{otherwise} \end{cases}$
c_i	$= \begin{cases} 1 & \text{processor switches from idle to sleep state after } \tau_i \\ 0 & \text{otherwise} \end{cases}$
Continuous Variables	
o_i	optional cycles of τ_i
t_i^s	start time of τ_i

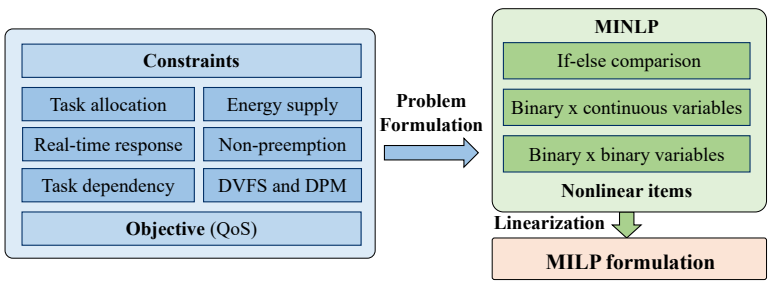


Fig. 3. Structure of task deployment problem.

where $t_i^e = t_i^s + t_i^c$, and t_i^c is the computation time of task τ_i. If $D_i = H$, Equation (4) can be replaced by Equation (5) since $D_i \leq H$ and $t_i^s \geq 0$.

4.3 Task Non-Preemption Constraints

Note that some tasks in the task set $\mathcal{T}$ are independent. When the independent tasks, e.g., τ_i and τ_j with $s_{ij} = 0$ are assigned to the same processor, such as θ_k, we need to determine task execution sequence, since one processor θ_k cannot process multiple tasks at the same time, i.e., tasks assigned on the same processor cannot overlap with each other [26]. Therefore, we have the following

constraints:

$$\sum_{k\in\mathcal{M}} p_{ijk}t_i^e \le t_j^s,\ \forall i \ne j \in \mathcal{N}'. \tag{6}$$

If $p_{ijk} = 1$, i.e., τ_i and τ_j are assigned to θ_k and τ_i precedes τ_j, we have $t_i^e \le t_j^s$ by Equation (6); otherwise, τ_i and τ_j are assigned to different processors, Equation (6) is always satisfied as $p_{ijk} = 0$ and $t_j^s \ge 0$. Since x_{ik} is influenced by p_{ijk}, we have $x_{ik} = \sum_{j\in\mathcal{N}'} p_{ijk}, \forall i \in \mathcal{N}$. This article does not explicitly consider priorities among the tasks [4], as it leads to preemptive execution.

4.4 Task Dependency Constraints

For the dependent tasks, e.g., τ_i and τ_j with $s_{ij} = 1$, when they are assigned to the same processor or different processors, the execution sequences between them are fixed. Thus, we have the following constraints:

$$t_i^e \le t_j^s + (1 - s_{ij})H,\ \forall i \ne j \in \mathcal{N}'. \tag{7}$$

If task τ_i is the predecessor of τ_j, as $s_{ij} = 1$, Equation (7) is relaxed to $t_i^e \le t_j^s$, i.e., the start time of τ_j should not exceed the end time of τ_i, otherwise ($s_{ij} = 0$), Equation (7) is relaxed to $t_i^e \le t_j^s + H$ and it is always satisfied as $0 \le t_i^s < t_i^e \le H$.

4.5 DPM Constraints

Note that the break-even time T_k^{BET} is given by Equation (2), and $I(\tau_i)$ denotes the time interval length after task τ_i is completed. If τ_i is assigned to processor θ_k, and the time period $I(\tau_i)$ of τ_i is larger than T_k^{BET}, θ_k can enter sleep state, i.e., $c_i = 1$. Therefore, we compare $I(\tau_i)$ with $\sum_{k\in\mathcal{M}} x_{ik}T_k^{BET}$ to determine the value of c_i. The relationship between them is given by

$$c_i = \begin{cases} 0, & I(\tau_i) \le \sum\limits_{k\in\mathcal{M}} x_{ik}T_k^{BET} \\ 1, & I(\tau_i) > \sum\limits_{k\in\mathcal{M}} x_{ik}T_k^{BET} \end{cases},\ \forall i \in \mathcal{N}. \tag{8}$$

4.6 Energy Constraints

Taking the available energy E_b during the scheduling horizon H into account, the system operation can be divided into three states [44]:

(1) *Low*: the energy E_b is insufficient to execute all the mandatory subtasks $\{M_1, \ldots, M_N\}$;
(2) *High*: the energy E_b is sufficient to execute all the mandatory and optional parts $\{M_1+O_1, \ldots, M_N+O_N\}$;
(3) *Medium*: all the mandatory subtasks are ensured to finish, while not all the optional subtasks have enough energy to complete their executions.

In this article, we primarily focus on the system in a medium-energy state. The platform's energy consumption details are as follows.

The total energy consumption of the M processors within the period H is the sum of the energy consumption in the active, idle, and sleep states, plus the switching costs, denoted by E_a, E_i, E_s, and E_{ov}, respectively. As the total energy consumption cannot exceed the budget E_b, we have

$$E_a + E_i + E_s + E_{ov} \le E_b. \tag{9}$$

If task τ_i with $M_i + o_i$ cycles is assigned to processor θ_k, and executed with the V/F level (v_{kl}, f_{kl}), the energy consumption of θ_k in the active state is $\frac{M_i+o_i}{\lambda_{ik}f_{kl}}P_{kl}^{active}$. Taking TA and FA q_{ikl} into account,

the energy consumption E_a of the multicore platform in the active state is calculated as

$$E_a = \sum_{i\in\mathcal{N}} \sum_{k\in\mathcal{M}} \sum_{l\in\mathcal{L}} q_{ikl} \frac{M_i + o_i}{\lambda_{ik} f_{kl}} P_{kl}^{active}. \tag{10}$$

When the assigned task τ_i is finished, and the processor enters the idle (sleep) state, we have $c_i = 0$ ($c_i = 1$). If the nearest task after τ_i is τ_j, i.e., $p_{ijk} = 1$, the value of time interval is $I(\tau_i) = \sum_{j\in\mathcal{N}} \sum_{k\in\mathcal{M}} p_{ijk}(t_j^s - t_i^e)$. Taking the value of c_i into account, the energy consumption E_i of the multicore platform in the idle state is

$$E_i = \sum_{i\in\mathcal{N}} \sum_{k\in\mathcal{M}} (1 - c_i) I(\tau_i) x_{ik} P_k^{idle}. \tag{11}$$

It is relaxed to $E_i = \sum_{i\in\mathcal{N}} \sum_{k\in\mathcal{M}} I(\tau_i) x_{ik} P_k^{idle}$ if $c_i = 0$, else, we have $E_i = 0$ as $c_i = 1$. The execution cycles of entry tasks $\{VTS_1, \ldots, VTS_M\}$ are zero, i.e., the corresponding task computation time is also zero. It is in the sleep state before a processor θ_k executes the first assigned task, e.g., $\tau_i \in \mathcal{T}$. Therefore, we restrict the original rather than the virtual tasks to calculate the energy consumption E_i in an idle state.

The energy consumption E_s in the sleep state is determined by the time spent in that state. Considering that a processor is in the sleep state before the first task starts and after all the assigned tasks end, E_s is composed of three parts:

$$E_s = \sum_{k\in\mathcal{M}} \Bigg[\underbrace{\sum_{i\in\mathcal{VS}} \sum_{j\in\mathcal{N}} p_{ijk} t_j^s}_{(a)} + \underbrace{\sum_{i\in\mathcal{N}} \sum_{j\in\mathcal{VE}} p_{ijk} \left(H - t_i^e\right)}_{(b)} + \underbrace{\sum_{i\in\mathcal{N}} c_i I(\tau_i) x_{ik}}_{(c)} \Bigg] P_k^{sleep}. \tag{12}$$

Let $\mathcal{VS}$ and $\mathcal{VE}$ denote the index sets of entry and exit tasks, respectively. For processor θ_k:

(1) If the i^{th} entry task and the j^{th} ordinary task (i.e., τ_j) are the first and the second assigned tasks, respectively, and θ_k is in the sleep state before τ_j starts its execution, as $p_{ijk} = 1$, the period in the sleep state is t_j^s, and thus, we have Equation (12)-(a);
(2) If the i^{th} ordinary task (i.e., τ_i) and the j^{th} exit task are the penultimate and the last assigned tasks of θ_k, respectively, and θ_k is in the sleep state after τ_i is finished, the period of θ_k in the sleep state is $H - t_i^e$, and thus, we have Equation (12)-(b) as $p_{ijk} = 1$;
(3) If the ordinary task τ_i is assigned to processor θ_k, and θ_k is in the sleep state when τ_i is finished, we have Equation (12)-(c) as the period of θ_k in sleep state is $I(\tau_i)$ and $c_i = 1$.

Finally, taking M processors and sleep power P_k^{sleep} into account, we get Equation (12). If there is no ordinary task assigned to processor θ_k, Equation (12)-(a), Equation (12)-(b), and Equation (12)-(c) are relaxed to 0 as $p_{ijk} = 0$ and $x_{ik} = 0$.

When processor θ_k changes its state, e.g., from the active to the sleep states when the assigned task τ_i is finished, the energy overhead E_k^{sw} is introduced due to the use of DPM technology. Since in the switching process, we have $c_i = 1$ and $x_{ik} = 1$, the total energy overhead can be given by

$$E_{ov} = \sum_{i\in\mathcal{N}} \sum_{k\in\mathcal{M}} c_i x_{ik} E_k^{sw}. \tag{13}$$

4.7 Objective Function and Primal Problem

For each task τ_i, a linear function $f_i(o_i) = k_i o_i + R_i$ is introduced to describe the non-decreasing relationship between task QoS and optional cycles o_i, where k_i and R_i are constants [2]. Considering the number of tasks, the QoS function is defined as $Q = \sum_{i\in\mathcal{N}} o_i$ [28]. Based on the constraints and

the objective (maximizing system QoS) mentioned above, the task deployment problem (primal problem) can be formulated as follows:

$$\mathbf{PP}: \Phi = \min_{p,q,c,o,t^s} \left(-\sum_{i \in \mathcal{N}} o_i \right) \tag{14}$$
$$\text{s.t.} \begin{cases} (3)-(9), \\ q_{ikl},\ p_{ijk},\ c_i \in \{0,1\},\ 0 \le o_i \le O_i,\ 0 \le t_i^s \le H. \end{cases}$$

4.8 Problem Linearization

There are three types of nonlinear items in PP:

(1) Nonlinear items caused by the comparison, e.g., Equation (8);
(2) Nonlinear items caused by the products of continuous and binary variables, e.g., $o_i q_{ikl}$ in Equation (10);
(3) Nonlinear items caused by the products of binary variables, e.g., $c_i x_{ik}$ in Equation (13).

Finding an optimal solution for the MINLP-based PP is complex. To solve this problem efficiently, we simplify its structure and reformulate it as an MILP using the following linearization methods.

4.8.1 Nonlinear Item Caused by the If-Else Comparison. We introduce *Lemma* 4.1 to linearize the comparison given by Equation (8).

Lemma 4.1. *Assume that b is a binary variable, while $f(x)$ is a linear function and is bounded by $L \le f(x) \le U$ and $L < 0 < U$. A linear constraint $(1-b)L + b\zeta \le f(x) \le bU$ can be used to describe the comparison $f(x) \le 0 \Rightarrow b = 0$, and $f(x) > 0 \Rightarrow b = 1$, where ζ is a small enough positive constant.*

Proof. If $f(x) \le 0$, by the left part of $(1-b)L + b\zeta \le f(x) \le bU$ (i.e., $(1-b)L + b\zeta \le 0$), we have $b = 0$, and the constraint is transformed to $L \le f(x) \le 0$. If $f(x) > 0$, by the right part of $(1-b)L + b\zeta \le f(x) \le bU$ (i.e., $0 < bU$), we get $b = 1$, and the constraint is transformed to $\zeta \le f(x) \le U$. □

Since $-H \le I(\tau_i) - \sum_{k \in \mathcal{M}} x_{ik} T_k^{BET} \le H$, according to *Lemma* 4.1, the comparison between time interval $I(\tau_i)$ and break-even time T_k^{BET} can be expressed linearly as follows:

$$-(1-c_i)H + c_i\zeta \le I(\tau_i) - \sum_{k \in \mathcal{M}} x_{ik} T_k^{BET} \le c_i H,\ \forall i \in \mathcal{N}. \tag{15}$$

4.8.2 Nonlinear Items Caused by the Products of Binary and Continuous Variables. We introduce *Lemma* 4.2 to linearize nonlinear terms by multiplying integer and continuous variables.

Lemma 4.2. *Assume that we have two constants $s_1, s_2 > 0$ and two constraint spaces $P_1 = \{|t, b, x| \, t = bx, -s_1 \le x \le s_2, b \in \{0,1\}\}$, $P_2 = \{|t, b, x| - bs_1 \le t \le bs_2, -s_1(1-b) \le x - t \le s_2(1-b), b \in \{0,1\}\}$, then $P_1 \rightleftharpoons P_2$.*

Proof. 1) $P_1 \Rightarrow P_2$. Since $t = bx$ and $-s_1 \le x \le s_2$, we have $-bs_1 \le t \le bs_2$. By $-s_1 \le x \le s_2$ and $b \in \{0,1\}$, we get $(b-1)(x - s_2) \ge 0$ and $(b-1)(x + s_1) \le 0$. Then, $t - bs_2 - x + s_2 \ge 0$ and $t + bs_1 - x - s_1 \le 0$ hold. As $x - t \le (1-b)s_2$ and $-s_1(1-b) \le x - t$, we have $-s_1(1-b) \le x - t \le s_2(1-b)$. 2) $P_2 \Leftarrow P_1$. If $b = 0$, we have $t = 0$ and $-s_1 \le x \le s_2$, else (i.e., $b = 1$), we get $-s_1 \le t = bx = x \le s_2$. Therefore, $P_1 \rightleftharpoons P_2$. □

In this article, we need to handle the following several nonlinear terms caused by the products of binary and continuous variables:

$-\sum_{k\in\mathcal{M}} p_{ijk}t_i^e = \sum_{k\in\mathcal{M}} p_{ijk}\left(t_i^s + t_i^c\right) = \sum_{k\in\mathcal{M}} p_{ijk}\left(t_i^s + \sum_{k\in\mathcal{M}}\sum_{l\in\mathcal{L}} q_{ikl}\frac{M_i+o_i}{\lambda_{ik}f_{kl}}\right)$ in Equation (6);
$-\sum_{i\in\mathcal{N}}\sum_{k\in\mathcal{M}}\sum_{l\in\mathcal{L}} q_{ikl}\frac{M_i+o_i}{\lambda_{ik}f_{kl}}$ in Equation (10);
$-\sum_{i\in\mathcal{N}}\sum_{k\in\mathcal{M}}(1-c_i)I(\tau_i)x_{ik}P_k^{idle}$ in Equation (11).

For the nonlinear items in Equation (11), as time interval $I(\tau_i) = \sum_{j\in\mathcal{N}}\sum_{k\in\mathcal{M}} p_{ijk}(t_j^s - t_i^e)$, we can get

$$\begin{aligned}
&\sum_{i\in\mathcal{N}}\sum_{k\in\mathcal{M}}(1-c_i)I(\tau_i)x_{ik}P_k^{idle}\\
&=\sum_{i\in\mathcal{N}}\sum_{k\in\mathcal{M}}(1-c_i)\left[\sum_{j\in\mathcal{N}}\sum_{k\in\mathcal{M}} p_{ijk}(t_j^s - t_i^e)\right]x_{ik}P_k^{idle}\\
&=\sum_{i\in\mathcal{N}}\sum_{k\in\mathcal{M}}(1-c_i)\left[\sum_{j\in\mathcal{N}}\sum_{k\in\mathcal{M}} p_{ijk}(t_j^s - t_i^s - t_i^c)\right]x_{ik}P_k^{idle}\\
&=\sum_{i\in\mathcal{N}}\sum_{k\in\mathcal{M}}(1-c_i)\left[\sum_{j\in\mathcal{N}}\sum_{k\in\mathcal{M}} p_{ijk}\left(t_j^s - t_i^s - \sum_{k\in\mathcal{M}}\sum_{l\in\mathcal{L}} q_{ikl}\frac{M_i+o_i}{\lambda_{ik}f_{kl}}\right)\right]x_{ik}P_k^{idle}.
\end{aligned} \tag{16}$$

Note that in Equation (16) the processor index k of the outer layer, middle layer, and inner layer could be different. For the sake of presentation, we rewrite Equation (16) as follows:

$$\sum_{i\in\mathcal{N}}\sum_{k''\in\mathcal{M}}(1-c_i)\left[\sum_{j\in\mathcal{N}}\sum_{k'\in\mathcal{M}} p_{ijk'}\left(t_j^s - t_i^s - \sum_{k\in\mathcal{M}}\sum_{l\in\mathcal{L}} q_{ikl}\frac{M_i+o_i}{\lambda_{ik}f_{kl}}\right)\right]x_{ik''}P_{k''}^{idle}. \tag{17}$$

Similarly, we get $\sum_{k\in\mathcal{M}} p_{ijk}\left(t_i^s + \sum_{k\in\mathcal{M}}\sum_{l\in\mathcal{L}} q_{ikl}\frac{M_i+o_i}{\lambda_{ik}f_{kl}}\right) = \sum_{k'\in\mathcal{M}} p_{ijk'}\left(t_i^s + \sum_{k\in\mathcal{M}}\sum_{l\in\mathcal{L}} q_{ikl}\frac{M_i+o_i}{\lambda_{ik}f_{kl}}\right)$ in Equation (6).

According to *Lemma* 4.2, we first introduce an auxiliary variable o_q_{ikl} to replace the nonlinear item $o_i q_{ikl}$ in $\sum_{i\in\mathcal{N}}\sum_{k\in\mathcal{M}}\sum_{l\in\mathcal{L}} q_{ikl}\frac{M_i+o_i}{\lambda_{ik}f_{kl}}$. Since $q_{ikl}\in\{0,1\}$ and $o_i\in[0,O_i]$, we add the following constraints to the problem (14):

$$\{0 \le o_i - o_q_{ikl} \le O_i(1-q_{ikl}),\ 0 \le o_q_{ikl} \le O_i q_{ikl},\ \forall i\in\mathcal{N},\ \forall k\in\mathcal{M},\ \forall l\in\mathcal{L}\}. \tag{18}$$

Then, we deal with the nonlinear items in $\sum_{k'\in\mathcal{M}} p_{ijk'}\left(t_i^s + \sum_{k\in\mathcal{M}}\sum_{l\in\mathcal{L}} q_{ikl}\frac{M_i+o_i}{\lambda_{ik}f_{kl}}\right)$. According to the linearization method in Section 4.8.3, to deal with the nonlinear item $p_{ijk'}q_{ikl}$ caused by the products of binary variables, we replace it with auxiliary variable $p_q_{ijkk'l}$. Since $p_q_{ijkk'l}$ is also a binary variable, we set $o_i p_q_{ijkk'l} = o_p_q_{ijkk'l}$. Similarly, we set $p_{ijk'}t_i^s = p_t_{ijk'}^{si}$. For $o_p_q_{ijkk'l}$ and $p_t_{ijk'}^{si}$, since $t_i^s\in[0,H]$, $p_{ijk'}\in\{0,1\}$ and $p_q_{ijkk'l}\in\{0,1\}$, we add the following constraints to the problem (14):

$$\begin{aligned}
\{0 \le o_i - o_p_q_{ijkk'l} \le O_i(1-p_q_{ijkk'l}),\ 0 \le o_p_q_{ijkk'l} \le O_i p_q_{ijkk'l},\\
\forall i,j\in\mathcal{N}',\ \forall k,k'\in\mathcal{M},\ \forall l\in\mathcal{L}\},
\end{aligned} \tag{19}$$

$$\{0 \le t_i^s - p_t_{ijk'}^{si} \le H(1-p_{ijk'}),\ 0 \le p_t_{ijk'}^{si} \le H p_{ijk'},\ \forall i,j\in\mathcal{N}',\ \forall k'\in\mathcal{M}\}, \tag{20}$$

Next, we deal with the nonlinear items in Equation (17). To this end, we first set $p_{ijk'}x_{ik''} = p_x_{ijk'k''}$, $p_{ijk'}q_{ikl}x_{ik''} = p_q_x_{ijkk'k''l}$, $c_i p_{ijk'}x_{ik''} = c_p_x_{ijk'k''}$, and $c_i p_{ijk'}q_{ikl}x_{ik''} = c_p_q_x_{ijkk'k''l}$. Then, we set $p_x_{ijk'k''}t_j^s = p_x_t_{ijk'k''}^{sj}$, $p_x_{ijk'k''}t_i^s = p_x_t_{ijk'k''}^{si}$, $c_p_q_x_{ijkk'k''l}t_j^s =$

$c_p_q_x_t^{sj}_{ijkk'k''l}$, $c_p_q_x_{ijkk'k''l}t_i^s = c_p_q_x_t^{si}_{ijkk'k''l}$, $p_q_x_{ijkk'k''l}o_i = o_p_q_x_{ijkk'k''l}$, and $c_p_q_x_{ijkk'k''l}o_i = c_o_p_q_x_{ijkk'k''l}$, and add the following constraints to the problem (14):

$$\{0 \le t_j^s - p_x_t^{sj}_{ijk'k''} \le H(1 - p_x_{ijk'k''}),\ 0 \le p_x_t^{sj}_{ijk'k''} \le H p_x_{ijk'k''},\ \forall i, j \in \mathcal{N},\ \forall k', k'' \in \mathcal{M}\}, \tag{21}$$

$$\{0 \le t_i^s - p_x_t^{si}_{ijk'k''} \le H(1 - p_x_{ijk'k''}),\ 0 \le p_x_t^{si}_{ijk'k''} \le H p_x_{ijk'k''},\ \forall i, j \in \mathcal{N},\ \forall k', k'' \in \mathcal{M}\}, \tag{22}$$

$$\{0 \le t_j^s - c_p_q_x_t^{sj}_{ijkk'k''l} \le H(1 - c_p_q_x_{ijkk'k''l}),\ 0 \le c_p_q_x_t^{sj}_{ijkk'k''l} \le H c_p_q_x_{ijkk'k''l},\ \forall i, j \in \mathcal{N},\ \forall k, k', k'' \in \mathcal{M},\ \forall l \in \mathcal{L}\}, \tag{23}$$

$$\{0 \le t_i^s - c_p_q_x_t^{si}_{ijkk'k''l} \le H(1 - c_p_q_x_{ijkk'k''l}),\ 0 \le c_p_q_x_t^{si}_{ijkk'k''l} \le H c_p_q_x_{ijkk'k''l},\ \forall i, j \in \mathcal{N},\ \forall k, k', k'' \in \mathcal{M},\ \forall l \in \mathcal{L}\}, \tag{24}$$

$$\{0 \le o_i - o_p_q_x_{ijkk'k''l} \le O_i(1 - p_q_x_{ijkk'k''l}),\ 0 \le o_p_q_x_{ijkk'k''l} \le O_i p_q_x_{ijkk'k''l},\ \forall i, j \in \mathcal{N},\ \forall k, k', k'' \in \mathcal{M},\ \forall l \in \mathcal{L}\}, \tag{25}$$

$$\{0 \le o_i - c_o_p_q_x_{ijkk'k''l} \le O_i(1 - c_p_q_x_{ijkk'k''l}),\ 0 \le c_o_p_q_x_{ijkk'k''l} \le O_i c_p_q_x_{ijkk'k''l},\ \forall i, j \in \mathcal{N},\ \forall k, k', k'' \in \mathcal{M},\ \forall l \in \mathcal{L}\}, \tag{26}$$

4.8.3 Nonlinear Items Caused by the Products of Binary Variables. We introduce *Lemma* 4.3 to deal with nonlinear items that have the form of integer variable multiplication.

Lemma 4.3. *Assume that x_1 and x_2 are binary variables, the nonlinear term x_1x_2 can be replaced by an auxiliary variable $y = x_1x_2$, and the additional linear constraints: (1) $y \le x_1$, (2) $y \le x_2$, and (3) $y \ge x_1 + x_2 - 1$.*

Proof. When $x_1 = x_2 = 1$, the constraints lead to $1 \le y \le 1$, i.e., $y = x_1x_2 = 1$. For the residual cases: (1) $x_1 = 0$, $x_2 = 1$; (2) $x_1 = 1$, $x_2 = 0$; (3) $x_1 = 0$, $x_2 = 0$, the constraints lead to $0 \le y \le 0$, we have $y = x_1x_2 = 0$. □

According to *Lemma* 4.3, to deal with the nonlinear item c_ix_{ik} in Equation (13), we set $c_x_{ik} = c_ix_{ik}$ and add the following constraints to the problem (14):

$$\{c_x_{ik} \le x_{ik},\ c_x_{ik} \le c_i,\ c_x_{ik} \ge x_{ik} + c_i - 1, \forall i \in \mathcal{N},\ \forall k \in \mathcal{M}\}. \tag{27}$$

Then, to handle $p_{ijk'}q_{ikl}$ in Equation (6), we introduce $p_q_{ijkk'l}$ and add the following constraints:

$$\{p_q_{ijkk'l} \le p_{ijk'},\ p_q_{ijkk'l} \le q_{ikl},\ p_q_{ijkk'l} \ge p_{ijk'} + q_{ikl} - 1, \forall i, j \in \mathcal{N}',\ \forall k, k' \in \mathcal{M},\ \forall l \in \mathcal{L}\}. \tag{28}$$

Finally, we set $p_{ijk'}x_{ik''} = p_x_{ijk'k''}$, $p_{ijk'}q_{ikl}x_{ik''} = p_q_x_{ijkk'k''l}$, $c_ip_{ijk'}x_{ik''} = c_p_x_{ijk'k''}$, and $c_ip_{ijk'}q_{ikl}x_{ik''} = c_p_q_x_{ijkk'k''l}$, and add the following constraints to deal with the nonlinear items in Equation (11).

$$\{p_x_{ijk'k''} \le x_{ik''},\ p_x_{ijk'k''} \le p_{ijk'}, p_x_{ijk'k''} \ge x_{ik''} + p_{ijk'} - 1,\ \forall i, j \in \mathcal{N},\ \forall k', k'' \in \mathcal{M}\}, \tag{29}$$

$$\{p_q_x_{ijkk'k''l} \le p_x_{ijk'k''},\ p_q_x_{ijkk'k''l} \le q_{ikl}, p_q_x_{ijkk'k''l} \ge p_x_{ijk'k''} + q_{ikl} - 1,\ \forall i, j \in \mathcal{N},\ \forall k, k', k'' \in \mathcal{M},\ \forall l \in \mathcal{L}\}, \tag{30}$$

Table 3. Linearization Variables and Constraints

Nonlinear Items	Auxiliary Variables	Additional Constraints
$c_i x_{ik}$	c_x_{ik}	(27)
$p_{ijk'} q_{ikl}$	$p_q_{ijkk'l}$	(28)
$p_{ijk'} x_{ik''}$	$p_x_{ijk'k''}$	(29)
$p_{ijk'} q_{ikl} x_{ik''}$	$p_q_x_{ijkk'k''l}$	(30)
$c_i p_{ijk'} x_{ik''}$	$c_p_x_{ijk'k''}$	(31)
$c_i p_{ijk'} q_{ikl} x_{ik''}$	$c_p_q_x_{ijkk'k''l}$	(32)
$o_i q_{ikl}$	o_q_{ikl}	(18)
$o_i p_q_{ijkk'l}$	$o_p_q_{ijkk'l}$	(19)
$p_q_x_{ijkk'k''l} o_i$	$o_p_q_x_{ijkk'k''l}$	(25)
$c_p_q_x_{ijkk'k''l} o_i$	$c_o_p_q_x_{ijkk'k''l}$	(26)
$p_{ijk'} t_i^s$	$p_t_{ijk'}^{si}$	(20)
$p_x_{ijk'k''} t_j^s$	$p_x_t_{ijk'k''}^{sj}$	(21)
$p_x_{ijk'k''} t_i^s$	$p_x_t_{ijk'k''}^{si}$	(22)
$c_p_q_x_{ijkk'k''l} t_j^s$	$c_p_q_x_t_{ijkk'k''l}^{sj}$	(23)
$c_p_q_x_{ijkk'k''l} t_i^s$	$c_p_q_x_t_{ijkk'k''l}^{si}$	(24)

$$\{c_p_x_{ijk'k''} \le p_x_{ijk'k''},\ c_p_x_{ijk'k''} \le c_i, c_p_x_{ijk'k''} \ge p_x_{ijk'k''} + c_i - 1,\ \forall i, j \in \mathcal{N},\ \forall k', k'' \in \mathcal{M}\}, \tag{31}$$

$$\{c_p_q_x_{ijkk'k''l} \le p_q_x_{ijkk'k''l},\ c_p_q_x_{ijkk'k''l} \le c_i, c_p_q_x_{ijkk'k''l} \ge p_q_x_{ijkk'k''l} + c_i - 1,\ \forall i, j \in \mathcal{N},\ \forall k, k', k'' \in \mathcal{M},\ \forall l \in \mathcal{L}\}, \tag{32}$$

4.8.4 MILP-Based Primal Problem. The auxiliary variables and additional constraints used for linearization are summarized in Table 3. Using the above linearization methods, problem (14) can be transformed to the following MILP problem (33).

$$\mathbf{PP2}: \Phi = \min_{p,q,c,o,t^s,\Omega} \left(- \sum_{i \in \mathcal{N}} o_i \right) \tag{33}$$
$$\text{s.t.} \begin{cases} (3) - (7),\ (9),\ (15) - (32), \\ q_{ikl}, p_{ijk}, c_i, c_x_{ik}, p_q_{ijkk'l}, p_x_{ijk'k''}, p_q_x_{ijkk'k''l}, c_p_x_{ijk'k''}, c_p_q_x_{ijkk'k''l} \in \{0,1\}, \\ 0 \le o_i, o_q_{ikl}, o_p_q_{ijkk'l}, o_p_q_x_{ijkk'k''l}, c_o_p_q_x_{ijkk'k''l} \le O_i, \\ 0 \le t_i^s, p_t_{ijk'}^{si}, p_x_t_{ijk'k''}^{sj}, p_x_t_{ijk'k''}^{si}, c_p_q_x_t_{ijkk'k''l}^{sj}, c_p_q_x_t_{ijkk'k''l}^{si} \le H, \end{cases}$$

where $\Omega = \{c_x_{ik},\ p_q_{ijkk'l},\ p_x_{ijk'k''},\ p_q_x_{ijkk'k''l},\ c_p_x_{ijk'k''},\ c_p_q_x_{ijkk'k''l},\ o_q_{ikl}, o_p_q_{ijkk'l},\ o_p_q_x_{ijkk'k''l},\ c_o_p_q_x_{ijkk'k''l},\ p_t_{ijk'}^{si},\ p_x_t_{ijk'k''}^{sj},\ p_x_t_{ijk'k''}^{si},\ c_p_q_x_t_{ijkk'k''l}^{sj}, c_p_q_x_t_{ijkk'k''l}^{si}\}$ is the set of auxiliary variables. Although DVFS switching and core communication overheads are not considered in Equation (33), our method can be extended as follows.

Remark 4.1. *DVFS switching overheads (i.e., switching energy and time) are mainly influenced by the V/F level before and after the transition [32]. Let f_s and f_e denote the frequency before and after the transition, respectively. The time and energy overheads can be given by $t^{switch}(f_s, f_e) = T_{trans} \times |f_e - f_s|$ and $E^{switch}(f_s, f_e) = P_s^{switch} \times t^{switch}(f_s, f_e)$, respectively, where T_{trans} is a time transition parameter,*

and P_s^{switch} is the power before the frequency transition. Based on $t^{switch}(f_s, f_e)$, $E^{switch}(f_s, f_e)$, task allocation, frequency selection, and task sequence variables q_{ikl} and p_{ijk}, we can get t_{ijk}^{switch}, where t_{ijk}^{switch} denotes the time overhead, when tasks τ_i and τ_j are allocated to the same processor, e.g., θ_k, and executed in sequence with different frequencies. Similarly, we can calculate the energy overhead E_{ijk}^{switch}. Thus, DVFS overheads can be added into time and energy constraints, e.g., (5), (6), (7), and (9).

Remark 4.2. *When dependent tasks are allocated to different processors, additional communication costs arise from data transfers between them. Let $TC = [tc_{mn}]_{M\times M}$ ($EC = [ec_{mn}]_{M\times M}$) denote the processor communication time (energy) matrix, where tc_{mn} (ec_{mn}) is the communication time (energy) required to transmit unit task data from processor θ_m to θ_n. If $m = n$, we have $tc_{mn} = 0$ ($ec_{mn} = 0$). According to the edge of $G(T, E)$, we can get a task data matrix $DT = [dt_{ij}]_{N\times N}$, where dt_{ij} is the size of task data that τ_i transmits to τ_j when τ_i is finished. If τ_i and τ_j are independent or $i = j$, we have $dt_{ij} = 0$. Assume that task τ_i is a predecessor of τ_j, i.e., $s_{ij} = 1$. If τ_i and τ_j are allocated to different processors, e.g., θ_m and θ_n, the data communication time and energy among τ_i and τ_j are $t_{ij}^{trans} = \sum_{m\in\mathcal{M}}\sum_{n\in\mathcal{M}} dt_{ij}x_{im}x_{jn}tc_{mn}$ and $E_{ij}^{trans} = \sum_{m\in\mathcal{M}}\sum_{n\in\mathcal{M}} dt_{ij}x_{im}x_{jn}ec_{mn}$. Hence, the constraints regarding time and energy can be updated accordingly.*

5 Heuristic Task Deployment Algorithm

Using the above linearization methods, the original MINLP problem is transformed into an MILP problem. This enables solving the problem optimally using methods like **Branch-and-Bound** (**B&B**) and **Benders Decomposition** (**BD**) [28] or solvers such as IBM ILOG CPLEX and Gurobi [8]. However, as the problem size grows, e.g., with increasing task number N or processor count M, the solution time for the MILP problem scales exponentially, limiting its applicability in complex scenarios. To address this, we propose a novel heuristic algorithm to improve scalability by efficiently solving the task deployment problem. The original problem (PP) is decomposed into two subproblems based on its structure, which are solved sequentially using low-complexity greedy algorithms. A feedback mechanism is introduced to enhance the quality of the final solution by refining decisions through adjustments to V/F levels and optional subtask cycles during iterations between subproblems.

We design the heuristic algorithm for PP instead of the linearized problem (PP1) for two main reasons:

(1) PP has fewer variables and constraints than PP1, simplifying algorithm design;
(2) Problem decomposition in PP decouples specific nonlinear terms, altering PP1's structure.

Considering processor heterogeneity, we first determine task-to-core allocation and frequency-to-task selection. Next, task execution order is optimized based on dependencies, TS, and optional cycle adjustments to enable DPM. Finally, FA and optional cycle adjustments are further refined to improve the results from earlier steps. The algorithm structure is shown in Figure 4.

5.1 Task Allocation and Frequency Assignment

Since the aim of the task deployment problem (14) is to maximize system QoS, i.e., $\sum_{i\in\mathcal{N}} o_i$, under energy constraints (9) and time constraints (4) and (5), we set $o_i = O_i$ in this step. Note that the adjustments of FA q_{ikl} and optional cycle o_i will influence time and energy constraints. If these constraints are hard to satisfy by changing q_{ikl}, we can further reduce o_i. Moreover, DPM is not considered in this step, i.e., c_i, as task execution sequence and TS are unknown at the current step, influencing the length of the time interval between the adjacent tasks.

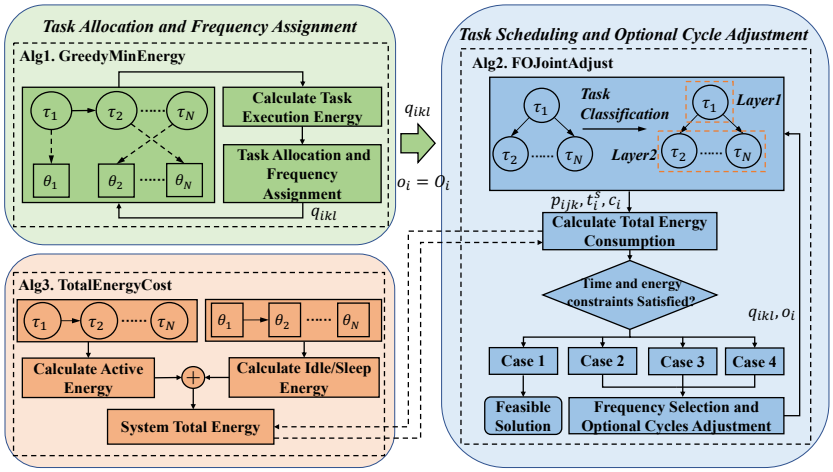


Fig. 4. Structure of the proposed heuristic method.

To minimize task execution energy E_a by DVFS, the TA and FA problem is given by

$$\mathbf{SP1}: \min_{q} \left(\sum_{i\in\mathcal{N}'} \sum_{k\in\mathcal{M}} \sum_{l\in\mathcal{L}} q_{ikl} \frac{M_i + O_i}{\lambda_{ik} f_{kl}} P_{kl}^{active} \right) \tag{34}$$
$$\text{s.t.} \begin{cases} (3), \\ q_{ikl} \in \{0, 1\}. \end{cases}$$

Note that SP1 is an ILP. To solve this problem effectively, we design a heuristic method based on the greedy algorithm, summarized in Algorithm 1. For the sake of presentation, we introduce a TA index $t2p[i]$ and a FA index $f2t[i]$ for each task τ_i, where $t2p[i] = k$ and $f2t[i] = l$ represents task τ_i is allocated to processor θ_k, and executed with the V/F level (v_{kl}, f_{kl}), respectively. In Algorithm 1, we determine TA and FA at the same time. Specifically, for each task τ_i with $M_i + O_i$ cycles, we allocate it to processor θ_k and use V/F level (v_{kl}, f_{kl}) to execute it. Then, we can obtain the corresponding task execution energy $\frac{M_i+O_i}{\lambda_{ik} f_{kl}} P_{kl}^{active}$. When task τ_i is assigned to different processors and executed with different V/F levels, we can find the smallest task execution energy $\min_{k\in\mathcal{M}, l\in\mathcal{L}} \{\frac{M_i+O_i}{\lambda_{ik} f_{kl}} P_{kl}^{active}\}$. By applying the above method for each task in the set $\{\tau_1, \ldots, \tau_N\}$, we can get the solution to SP1.

In Algorithm 1, we begin by initializing the indexes $f2t[i] = 0, t2p[i] = 0$, and $o_i = O_i$ for all tasks. Subsequently, we set $eMin$, which represents the minimum task execution energy consumption for each task, and initialize it as $eMin = \infty$. Since the processors are heterogeneous, the supply voltage v_{kl} and working frequency f_{kl} are influenced by task-to-core allocation; therefore, the values of $f2t[i]$ and $t2p[i]$ must be considered simultaneously. When task τ_i is assigned to processor θ_k and executed with the V/F level (v_{kl}, f_{kl}), the task execution energy is calculated as $e = \frac{M_i+o_i}{\lambda_{ik} f_{kl}} P_{kl}^{active}$. If this energy value e is less than the current minimum energy consumption $eMin$, we record this TA and FA decision for τ_i and update $t2p[i] = k$, $f2t[i] = l$, and $eMin = e$. Otherwise, the values of $t2p[i]$, $f2t[i]$, and $eMin$ remain unchanged. This process is applied iteratively for each task in the set $\{\tau_1, \ldots, \tau_N\}$, and the final decision q_{ikl} is determined based on the updated values of $t2p[i]$ and $f2t[i]$.

5.2 Task Scheduling and Optional Cycle Adjustment

Based on the solution to SP1, the execution time and energy of each task τ_i are known, i.e., $\sum_{i\in\mathcal{N}'} \sum_{k\in\mathcal{M}} \sum_{l\in\mathcal{L}} q_{ikl} \frac{M_i+O_i}{\lambda_{ik} f_{kl}}$ and $\sum_{i\in\mathcal{N}'} \sum_{k\in\mathcal{M}} \sum_{l\in\mathcal{L}} q_{ikl} \frac{M_i+O_i}{\lambda_{ik} f_{kl}} P_{kl}^{active}$. Substituting q_{ikl} into PP, this

ALGORITHM 1: GreedyMinEnergy

Input: Parameters in Table 2
Output: q_{ikl}

```
1  Initialization: t2p[i] = 0, f2t[i] = 0, and o_i = O_i;
2  for ∀i ∈ 𝒩 do
3      eMin = ∞;
4      for ∀k ∈ ℳ do
5          for ∀l ∈ ℒ do
6              e = ((M_i + o_i)/(λ_ik f_kl)) P_kl^active;
7              if e < eMin then
8                  Set eMin = e, t2p[i] = k, and f2t[i] = l;
9              else
10                 continue;
11             end
12         end
13     end
14     Calculate q_ikl according to t2p[i] and f2t[i];
15 end
```

problem is relaxed to SP2.

$$\mathbf{SP2}: \Phi = \min_{p,c,o,t^s} \left(- \sum_{i \in \mathcal{N}} o_i \right) \tag{35}$$
$$\text{s.t.} \begin{cases} (4) - (9), \\ p_{ijk},\ c_i \in \{0, 1\}, \\ 0 \le o_i \le O_i,\ 0 \le t_i^s \le H. \end{cases}$$

We divide the tasks into several layers based on their dependencies, as shown in Figure 1. Let $La(\tau_i)$ denote the layer of task τ_i. For entry tasks, e.g., τ_1 and τ_2, we set $La(\tau_1) = La(\tau_2) = 0$. For other tasks, the layer is defined as the maximum number of edges from the task to an entry task, i.e., $La(\tau_i) = \max_{e_{ji} \in E}\{La(\tau_j)\} + 1$, where e_{ji} represents the edge from τ_j to τ_i, and E is the edge set of $G(T, E)$. For instance, $La(\tau_3) = La(\tau_4) = 1$, $La(\tau_5) = 2$, and $La(\tau_6) = 3$.

Tasks in adjacent layers are dependent, e.g., τ_1 and τ_3, while tasks in the same layer are independent, e.g., τ_1 and τ_2. Scheduling tasks in the sequence $\{\dots, La(\tau_m) = i, La(\tau_n) = i + 1, \dots\}$ satisfies non-preemption (6) and dependency constraints (7). To better satisfy time constraints (4) and (5), we sort tasks within each layer in descending order of their relative deadlines. This rearranges $\mathcal{T}$ into a new sequence $TaskSeq$ (Line 3). Using $TaskSeq$ and q_{ikl}, we update the start t_i^s and end t_i^e times of each task τ_i. For adjacent tasks on the same processor, e.g., τ_i and τ_j, the DPM variable c_i is determined by comparing $t_j^s - t_i^e$ with T_k^{BET} using Equation (8).

When setting $o_i = O_i$, variables q_{ikl}, p_{ijk}, c_i, and t_i^s are calculated. However, directly following $TaskSeq$ may render the problem infeasible due to difficulty satisfying constraints (4), (5), and (9). In contrast, constraints such as TA and FA (3), dependency (7), and non-preemption (6) can be satisfied by Algorithm 1 and the proposed classification method. Based on constraints (4), (5), and (9), we consider the following four cases:

— (4), (5), and (9) are all satisfied, i.e., the solution, including q_{ikl}, p_{ijk}, c_i, t_i^s, and $o_i = O_i$, is feasible;
— (4) and/or (5) are not satisfied, but (9) is;
— (9) is not satisfied, but (4) and (5) are;
— (4), (5), and (9) are all not satisfied.

ALGORITHM 2: FOJointAdjust

Input: q_{ikl}, c_i, t_i^s, t_i^e, o_i, $TaskSeq$, and parameters in Table 2
Output: q_{ikl}, c_i, t_i^s, t_i^e, o_i, and $flag$

```
Initialization: flag = 0, flag_i = 0, and tp_o ∈ (0, 1);
Calculate f2t[i] and t2p[i] according to q_ikl;
Sort τ_i in order of TaskSeq;
for ∀i ∈ N do
    if flag==0 then
        while o_i ≥ M_i + tp_o O_i & flag_i == 0 do
            o_i = o_i − tp_o O_i;
            while f2t[i] < L do
                f2t[i] = f2t[i] + 1;
                if TotalEnergyCost ≤ E_b then
                    Calculate the start and end time of all tasks;
                    if constraints (4) and (5) are satisfied then
                        Set flag_i = 1 and flag = 1;
                        break;
                    else if constraint (5) is satisfied & t_i^e − t_i^s ≤ D_i then
                        Set flag_i = 1 and flag = 0;
                        break;
                    else
                        Set flag_i = 0 and flag = 0;
                        continue;
                    end
                else
                    f2t[i] = f2t[i] − 1;
                    break;
                end
            end
        end
    else
        break;
    end
end
Calculate q_ikl according to f2t[i], t2p[i];
```

For cases 2–4, we lower o_i to reduce task execution time and energy, making constraints (4), (5), and (9) easier to satisfy. However, as the objective is to maximize QoS (i.e., $\sum_{i\in\mathcal{N}} o_i$), o_i reduction should be minimal. Adjusting q_{ikl}, such as increasing the V/F level, can reduce execution time but increase energy consumption. To balance these, we simultaneously optimize o_i and q_{ikl} in Algorithm 2. Algorithm 2 adjusts o_i and q_{ikl} iteratively in a feedback control manner. The key ideas are:

(1) Increasing the V/F level reduces execution time but raises energy consumption;
(2) Optional cycles o_i are adjusted sequentially for all tasks in $TaskSeq$, starting with $o_i = O_i$. Updated o_i values yield new execution times and energies. By comparing $t_i^e - t_i^s \le D_i$ and $\max_{i\in\mathcal{N}}\{t_i^e\} \le H$, time constraints (4) and (5) are verified. Similarly, energy consumption is recalculated and compared with the energy budget E_b to verify (9).

If time and energy constraints (4), (5), and (9) are satisfied, the adjustment of o_i is finished, else, we continue to adjust the V/F selection by q_{ikl}. Thus, we have the following three cases:

ALGORITHM 3: TotalEnergyCost

Input: q_{ikl}, t_i^s, t_i^e, and parameters in Table 2
Output: c_i and E_c

Initialization: $Ec = 0$ and calculate T_k^{BET} by (2);
Calculate $t2p[i]$ and $f2t[i]$ according to q_{ikl};
for $\forall k \in \mathcal{M}$ **do**
  Build task set $\mathcal{N}_k \triangleq \{i | t2p[i] = k,\ \forall i \in \mathcal{N}\}$;
  Sort tasks in the set $\mathcal{N}_k$ in ascending order of start time;
  for $\forall i \in \{i | i \in \mathcal{N}_k \wedge t_i^s < \max_{i \in \mathcal{N}_k}\{t_i^s\}\}$ **do**
    Calculate $I(\tau_i)$;
    if $I(\tau_i) \geq T_k^{BET}$ **then**
      $c_i = 1$;
    else
      $c_i = 0$;
    end
    $Ec = Ec + (1 - c_i) I(\tau_i) P_{t2p[i]}^{idle}$;
    $Ec = Ec + c_i I(\tau_i) P_{t2p[i]}^{sleep}$;
    $Ec = Ec + c_i E_{t2p[i]}^{sw}$;
  end
  $Ec = Ec + \min_{i \in \mathcal{N}_k}\{t_i^s\} P_k^{sleep}$;
  $Ec = Ec + \left[H - \max_{i \in \mathcal{N}_k}\{t_i^e\}\right] P_k^{sleep}$;
end
for $\forall i \in \mathcal{N}$ **do**
  $Ec = Ec + (t_i^e - t_i^s) P_{t2p[i]f2t[i]}^{active}$;
end

— If the energy constraint (9) is satisfied while the time constraints (4) and/or (5) are not satisfied, we enhance q_{ikl} to decrease the task execution time of τ_i, with the cost of consuming more energy to execute the task.
— If the energy constraint (9) is not satisfied while the time constraints (4) and (5) are satisfied, we adjust o_i in the next round. This is because q_{ikl} is obtained by solving SP1, where the aim is to reduce task execution energy consumption $\sum_{i \in \mathcal{N}'} \sum_{k \in \mathcal{M}} \sum_{l \in \mathcal{L}} q_{ikl} \frac{M_i + O_i}{\lambda_{ik} f_{kl}} P_{kl}^{active}$, it is hard to lower V/F level further; however, to reduce task execution energy, we have to lower the V/F level.
— If the energy and time constraints are unsatisfied, we adjust o_i in the next round. Assume that the current round's index is m. We further reduce o_i at the next round $m + 1$. This process ends when $o_i = 0$. We apply this method to each task in $TaskSeq$ to complete the adjustment of o_i and q_{ikl}.

The details of o_i and q_{ikl} adjustments are summarized in Algorithm 2. In Algorithm 2, we begin by initializing the TS index $flag$ to zero, indicating that TS is infeasible, and update it to 1 when scheduling becomes feasible. Similarly, the task adjustment index $flag_i$ for task τ_i is initialized to zero, representing incomplete adjustments for o_i and q_{ikl}. The adjustment of o_i at round m is defined as $(1 - m \times tp_o) O_i$, where $tp_o \in (0, 1)$ is a tuned parameter. While a smaller tp_o makes time and energy constraints easier to satisfy, it increases the algorithm's complexity due to the higher number of adjustment rounds.

Tasks in the set $\mathcal{T}$ are sorted into the sequence $TaskSeq$ using the proposed task classification method. At the m-th round, for each task τ_i in $TaskSeq$, the optional cycle number o_i is updated to $(1 - m \times tp_o) O_i$, reducing by $tp_o \times O_i$ in each round. Using the updated o_i, the execution time

$\sum_{k\in\mathcal{M}}\sum_{l\in\mathcal{L}} q_{ikl}\frac{M_i+(1-m\times tp_o)O_i}{\lambda_{ik}f_{kl}}$ and end time t_i^e of τ_i are recalculated. Algorithm 3 is then introduced to compute the total energy consumption of the multicore platform, Ec. Based on the given o_i and Ec, the algorithm verifies whether the time constraints (4) and (5), and the energy constraint (9), are satisfied.

If the energy constraint (9) is satisfied but the time constraints (4) and/or (5) are violated, the frequency index of τ_i is increased from $f2t[i]$ to $f2t[i]+1$ to reduce execution time, albeit at the cost of higher energy consumption. This adjustment continues until the energy constraint (9) is no longer satisfied. If the total energy cost satisfies $TotalEnergyCost \le E_b$ for $f2t[i]$ but exceeds the budget when $f2t[i]$ is increased further, the maximum feasible FA for τ_i is recorded. If the maximum frequency level $f2t[i]=l_k$ still does not satisfy the energy constraint, we set $f2t[i]=l_k$.

Contrarily, if the time constraints (4) and (5) are satisfied but the energy constraint (9) is not, the optional cycle number o_i is reduced from o_i to $o_i - tp_o \times O_i$ to decrease energy consumption. This adjustment continues until the constraints (4) and (5) are satisfied, or o_i is reduced to zero. Reducing o_i satisfies the time constraints but violates the energy constraint; the most significant feasible optional cycle number is recorded. If $o_i=0$ still does not meet the constraints, o_i is set to zero.

When both time constraints (4) and (5), and the energy constraint (9) are violated, a similar approach is used to decrease o_i iteratively. Adjustments to o_i and q_{ikl} for τ_i influence its own execution time, but not the deadline constraints of other tasks τ_j ($j\neq i$). During this process, the deadline constraints $t_i^e - t_i^s \le D_i$, $t_i^e \le H$, and the energy constraint (9) are verified. However, tasks scheduled after τ_i in $TaskSeq$, such as τ_j, may temporarily violate their deadline constraints $t_j^e - t_j^s \le D_j$ until adjustments for o_j and q_{jkl} are made. Once the adjustments for o_i and q_{ikl} are complete, the task adjustment index for τ_i is updated to $flag_i = 1$.

This process is applied iteratively for all tasks in $TaskSeq$, adjusting the optional cycle numbers o_i and V/F selections q_{ikl}. When all constraints (4), (5), and (9) are satisfied, the TS index $flag$ is set to 1, indicating feasible scheduling. If constraints remain unsatisfied, $flag$ remains 0, and the scheduling is deemed infeasible.

In Algorithm 2, to check whether the energy constraint (9) is satisfied or not, we introduce Algorithm 3 to calculate the total energy consumption of the multicore platform, which is composed of task execution energy and energy consumed when the assigned tasks are finished. On one hand, the execution energy of task τ_i can be calculated as $(t_i^e - t_i^s)P^{active}_{t2p[i]f2t[i]}$, where $t_i^e - t_i^s$ is the task execution time and $P^{active}_{t2p[i]f2t[i]}$ is the power of the processor, when τ_i is assigned to θ_k, i.e., $t2p[i]=k$ and is executed with the V/F level (v_{kl}, f_{kl}), i.e., $f2t[i]=l$. On the other hand, when the assigned tasks are finished, the target processor will be in the idle or sleep state, based on the DPM decision c_i, which influences the processor's energy consumption. Assume that τ_i and τ_j are two adjacent tasks assigned to processor θ_k. By comparing the interval length $I(\tau_i) = t_j^s - t_i^e$ with the break-even time T_k^{BET}, we can determine the state of θ_k during the time interval $I(\tau_i)$, i.e., we can determine the value of c_i. The details of Algorithm 3 are summarized as follows:

— At the beginning of Algorithm 3, we initialize the total energy consumption $Ec=0$, and calculate break-even time T_k^{BET} according to (2) (Line 1). For processor θ_k, as TA and FA q_{ikl} and task sequence p_{ijk} are fixed, the tasks assigned to θ_k are known. Let $\mathcal{N}_k$ denote the index set of tasks assigned to θ_k, i.e., $\mathcal{N}_k \triangleq \{i | t2p[i]=k,\ \forall i\in\mathcal{N}\}$ (Line 4). By sorting all the tasks on processor θ_k in ascending order of task start time t_i^s, we can get the adjacent relationship between the tasks in the set $\mathcal{N}_k$ (Line 5), and calculate the time interval between these tasks (Line 7). Note that each processor θ_k is in a sleep state before executing the first assigned task and after all the assigned tasks are finished. The processor state during these two periods is not influenced by the DPM

constraint (8). Therefore, according to constraint (8), we can only determine the state of processor θ_k in the intervals after the first task ends and before the last task starts. When determining the DPM decision c_i for task τ_i, where τ_i is assigned to processor θ_k, i.e., $i \in \mathcal{N}_k$, the task index i should satisfy $t_i^s < \max_{j\in\mathcal{N}_k}\{t_j^s\}$, where $\max_{j\in\mathcal{N}_k}\{t_j^s\}$ is the start time of the last task assigned to processor θ_k. Assume that τ_i and τ_j are assigned to processor θ_k (i.e., $i \neq j \in \mathcal{N}_k$), and they are not the last task of θ_k. In addition, τ_j is the nearest task after τ_i (i.e., no task exists between τ_i and τ_j). The time interval between τ_i and τ_j is given by $I(\tau_i) = t_j^s - t_i^e$. By comparing $I(\tau_i)$ with T_k^{BET}, we can determine the DPM decision c_i for τ_i (Lines 8–12).

— If $c_i = 0$, the processor θ_k is in an idle state during the time interval $I(\tau_i)$, and the corresponding energy consumption can be calculated as $(1 - c_i)(t_i^e - t_i^s)P_k^{idle}$ (Line 13), where P_k^{idle} is the idle power of the processor θ_k. The idle energy consumption equals zero when $c_i = 1$, as the processor θ_k is in a sleep state during the time interval $I(\tau_i)$.

— If $c_i = 1$, the processor θ_k is in a sleep state during the time interval $I(\tau_i)$. The corresponding energy consumption includes processor sleep and switching energy (from sleep to idle states). Similarly, the sleep energy of processor θ_k in the time interval $I(\tau_i)$ can be calculated as $c_i(t_i^e - t_i^s)P_k^{sleep}$ (Line 14), which equals zero when $c_i = 0$, i.e., the processor θ_k is in an idle state during $I(\tau_i)$, and P_k^{sleep} is the idle power of the processor θ_k. In addition, the switching energy of processor θ_k from idle state to sleep state can be calculated as $c_i E_k^{sw}$, and it equals zero when $c_i = 0$ (Line 15).

— Note that processor θ_k is in a sleep state before the first assigned task starts and after all the assigned tasks end. We denote them as the first and the last sleep intervals. The energy of processor θ_k consumed in the first sleep interval is given by $(\min_{i\in\mathcal{N}_k}\{t_i^s\})P_k^{sleep}$, where $\min_{i\in\mathcal{N}_k}\{t_i^s\}$ is the length of the first sleep interval (Line 17). Similarly, the energy of processor θ_k consumed in the last sleep interval can be calculate as $(H - \max_{i\in\mathcal{N}_k}\{t_i^e\})P_k^{sleep}$, where $(H - \max_{\forall i\in\mathcal{N}_k}\{t_i^e\})$ is the length of the last sleep interval (Line 18).

— By applying the above methods for each processor θ_k in the set $\{\theta_1, \ldots, \theta_M\}$, we can calculate the idle and sleep energy consumption of the multicore system under the given frequency-to-task assignment and task-to-core allocation decisions.

— The execution energy of task τ_i can be calculated as $(t_i^e - t_i^s)P_{t2p[i]f2t[i]}^{active}$, where $t_i^e - t_i^s$ is the task execution time, and $P_{t2p[i]f2t[i]}^{active}$ is the power of the processor, when τ_i is assigned to processor θ_k, i.e., $t2p[i] = k$, and is executed with the V/F level (v_{kl}, f_{kl}), i.e., $f2t[i] = l$. Then, we can calculate the execution energy for all tasks in the set $\mathcal{T}$ and obtain the active energy consumption of the multicore system (Line 21).

From the above statement, we can see that the proposed heuristic mainly considers the QoS function in the second step, and it uses a greedy-based algorithm to find proper values for the optimization variables, e.g., t_i^s, p_{ijk}, and c_i, under multiple constraints. Unlike the optimal method, the MILP formulation, e.g., the linear QoS function, has limited influence on the proposed heuristic; our heuristic can be easily extended to the non-linear QoS function scenario.

The feedback-driven cycle adjustment (Algorithm 2) fundamentally relies on the principle that reducing optional cycles (o_i) saves execution time and energy at the cost of QoS. This core principle holds for any monotonically non-decreasing QoS function, including concave functions. If a nonlinear QoS model were adopted, the overall heuristic framework would remain completely valid. The only necessary algorithmic adaptation would be to replace the uniform cycle reduction step (tp_o) with a gradient-aware evaluation, allowing the algorithm to prioritize reducing optional cycles for tasks that incur the smallest marginal QoS loss (i.e., the lowest $\Delta\text{QoS}/\Delta o_i$).

5.3 Algorithm Complexity

The proposed algorithm divides the primal problem PP (14) into two subproblems, SP1 (34) and SP2 (35), and solves them sequentially through Algorithm 1 and Algorithm 2. To address the energy constraint (9) in SP2 (35), we introduce Algorithm 3, which is incorporated into Algorithm 2. Below, we detail the computational complexities of these algorithms.

Time complexity of Algorithm 1 depends on the number of binary variables in SP1 (34) [43], represented by $\boldsymbol{q} = [q_{ikl}]_{N\times M\times L}$. When task $\tau_i \in \mathcal{N}$ is assigned to processor $\theta_k \in \mathcal{M}$ and executed at V/F level $(v_{kl}, f_{kl}) \in \mathcal{VF}_k$, the corresponding task execution energy is given by $\sum_{k\in\mathcal{M}}\sum_{l\in\mathcal{L}} q_{ikl}\frac{M_i+o_i}{\lambda_{ik}f_{kl}}P_{kl}^{active}$. To minimize this energy, we determine a suitable value for q_{ikl}. Since the dimension of $\boldsymbol{q}$ is $N \times M \times L$, the overall complexity of Algorithm 1 is $O(NML)$.

Algorithm 2 adjusts the optional cycles o_i and V/F levels q_{ikl} for N tasks. During this process, for each task τ_i, the optional cycle number o_i is decreased iteratively from o_i to $o_i - tp_o \times O_i$, and the task execution frequency q_{ikl} is increased. In the worst case, $L-1$ levels of task execution frequency q_{ikl} are increased, and adjusting o_i requires $\frac{1}{tp_o}$ iterations. For each iteration, the start and end times of all tasks are recalculated with a complexity of $O(N)$, and the total energy consumption of the multicore system is computed with a complexity of $O(N\log N + 2N)$. Thus, the complexity of adjusting one task is $O\left(\frac{1}{tp_o}(L-1)(N\log N + 3N)\right)$.

Additionally, to handle task dependencies in SP2 (35), a task-layer classification method is used to determine the task sequence *TaskSeq*. The complexity of this classification is $O(N\log(N))$ [26]. In the worst case, adjustments to o_i and q_{ikl} are made for all N tasks in *TaskSeq* to satisfy the time and energy constraints. The complexity of Algorithm 2 is $O\left(\frac{1}{tp_o}(L-1)(N\log N + 3N)N + N\log(N)\right)$.

Finally, Algorithm 3 primarily calculates the total energy consumption of the multicore system. First, all tasks assigned to the same processor are sorted by their start times to form M task sequences. In the worst case, N tasks are allocated to one processor, resulting in a sorting complexity of $O(N\log N)$ [30]. Based on these M task sequences, the DPM decision c_i is determined by comparing the time interval $I(\tau_i)$ with the break-even time T_k^{BET}, followed by calculations for sleep and idle energy consumption. Given that $\boldsymbol{c} = [c_i]_{1\times N}$, the complexity for this step is $O(N)$. Additionally, we calculate the execution energy for N tasks to obtain the active energy consumption, which has complexity $O(N)$. Therefore, the total complexity of Algorithm 3 is $O(N\log N + 2N)$.

6 Simulation Results

In this section, a wide range of experiments are performed to evaluate the performance and effectiveness of our methods.

6.1 Simulation Setup

We consider the heterogeneous multicore platform that supports both DVFS and DPM technology as a case study. The power parameters of the processors are adopted from [1]. The execution efficiency factor λ_{ik} is set within the range $[0.4, 1]$ [25], which indicates the heterogeneity of a task executed in different processors. In Table 4, α, β, and δ are the tuned parameters. To make task deployment problem feasible and practical, $E_{\min}$ denotes the minimum energy required to execute all the mandatory subtasks, and $E_{\max}$ denotes the maximum energy required to execute all the mandatory and optional subtasks. Similarly, let CPT denote the index set of tasks on the critical path, then $H_{\min}$ denotes the minimum time required to execute all the mandatory subtasks on the critical path, and $H_{\max}$ denotes the maximum time required to execute all the mandatory and optional subtasks on the critical path. In addition, $D_i^{\min}$ denotes the minimum time required to execute the mandatory subtask of task τ_i, and $D_i^{\max}$ denotes the maximum time required to execute

Table 4. Experimental Setup

Energy Constraints	
$E_b = (1-\alpha)E_{\min} + \alpha E_{\max}$	
$E_{\min} = \min_{k\in\mathcal{M},l\in\mathcal{L}} \left\{\sum_{i\in\mathcal{N}} \frac{M_i}{\lambda_{ik} f_{kl}} P^{active}_{kl}\right\}$	$E_{\max} = \max_{k\in\mathcal{M},l\in\mathcal{L}} \left\{\sum_{i\in\mathcal{N}} \frac{M_i+O_i}{\lambda_{ik} f_{kl}} P^{active}_{kl}\right\}$
Real-time Constraints	
$H = (1-\beta)H_{\min} + \beta H_{\max}$	
$H_{\min} = \min_{k\in\mathcal{M},l\in\mathcal{L}} \left\{\sum_{i\in CPT} \frac{M_i}{\lambda_{ik} f_{kl}}\right\}$	$H_{\max} = \max_{k\in\mathcal{M},l\in\mathcal{L}} \left\{\sum_{i\in CPT} \frac{M_i+O_i}{\lambda_{ik} f_{kl}}\right\}$
$D_i = (1-\delta)D_i^{\min} + \delta D_i^{\max}$	
$D_i^{\min} = \min_{k\in\mathcal{M},l\in\mathcal{L}} \left\{\frac{M_i}{\lambda_{ik} f_{kl}}\right\}$	$D_i^{\max} = \max_{k\in\mathcal{M},l\in\mathcal{L}} \left\{\frac{M_i+O_i}{\lambda_{ik} f_{kl}}\right\}$

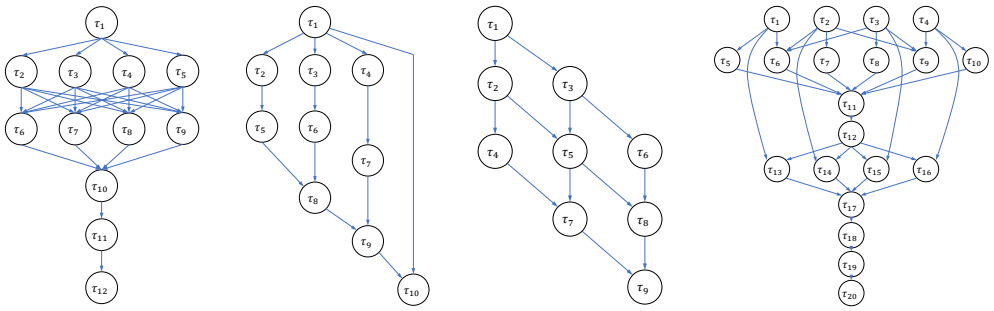
the mandatory and optional subtasks of τ_i. Therefore, we can change time and energy constraints by varying the parameters α, β, and δ.

Although the parameters of different processors and tasks yield different task deployment results, the problem structure remains the same. Therefore, the optimization problem and the proposed heuristic method are still applicable. The simulations are carried out on a computer with the Intel Core i5 processor operating at 1.80 GHz and 8 GB RAM, and the algorithms are implemented through MATLAB R2022a, YALMIP 2023, and Gurobi 10.0. In the following section, we first compare the system QoS and computation time of the proposed optimal (OPT) and heuristic (HEU) methods and other task deployment methods: task deployment methods with DVFS (TDV) [26], DPM (TDP) [23], and fixed task-to-core allocation with DVFS and DPM (FTA) [8]. Then, we compare the performances of the HEU, TDV, TDP, and FTA methods, e.g., system QoS and task schedulability, under different system parameters, e.g., energy parameter α, time parameters β and δ, and task number N. Note that DVFS switching and core communication overheads are not involved in Equation (33).

6.2 Results

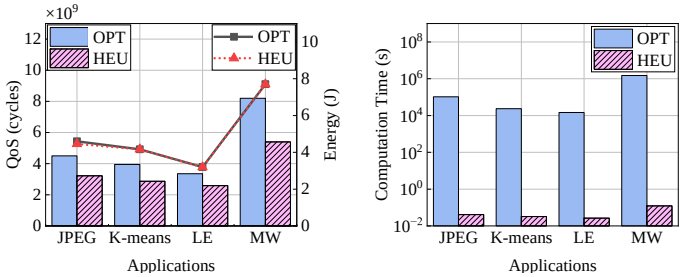
Figure 6 compares the system QoS and the computation time of OPT and HEU methods in different real applications, e.g., JPEG, K-means, **Laplace Equation** (**LE**), and **Montage Workflows** (**MW**), and the energy and time parameters are set to $\alpha = 0.45$, $\beta = 0.45$, and $\delta = 0.45$. The task numbers of JPEG, K-means, LE, and MW applications are $N = 12, 10, 9, 20$, respectively [3, 9, 11, 38], and the corresponding task DAGs are shown in Figure 5. The QoS of some tasks in JPEG, e.g., encode and decode, is related to task execution cycles [47]. In addition, K-means, LE, and MW are iteration-based methods [18, 21]. Therefore, the above applications can be modeled as IC tasks.

The results show that the computation time of the HEU method is much lower than that of the OPT method, as the HEU method usually takes less than 0.5 seconds. However, the system QoS of the HEU method is 28.18% (on average) lower than the OPT method. This is because the original task deployment problem (14) is an MINLP problem, and to transform it into an MILP problem (33) to find the optimal solution, the auxiliary variables ζ and the additional constraints (15)–(32) are added into problem (14). The computation time of the OPT method increases significantly with the task number N, as more variables and constraints are involved in problem (33), thereby increasing the computational complexity. However, the HEU method decomposes the primal problem (14) into



(a) JPEG (b) K-means (c) LE (d) MW

Fig. 5. Task dags of jpeg, k-means, le, and mw applications.



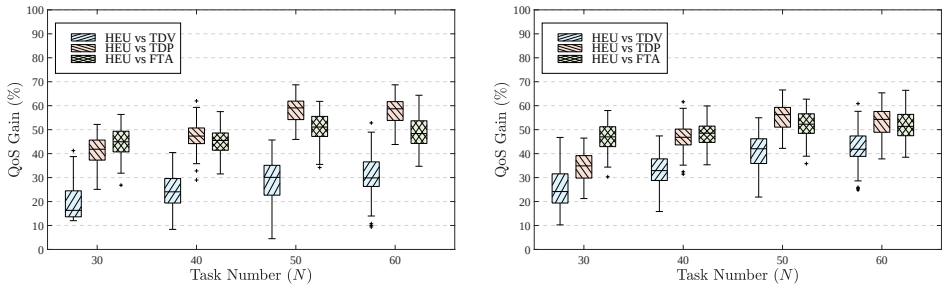
(a) System QoS. (b) Computation time.

Fig. 6. QoS and time comparisons of opt and heu with jpeg, k-means, le, and mw applications.

two subproblems, SP1 and SP2, with fewer variables and constraints, and solves these subproblems in sequence with low complexity greedy-based algorithms, i.e., Algorithm 1–Algorithm 3. This result aligns with our complexity analysis in Section 5. The proposed HEU method is suitable for large-scale systems due to its scalability.

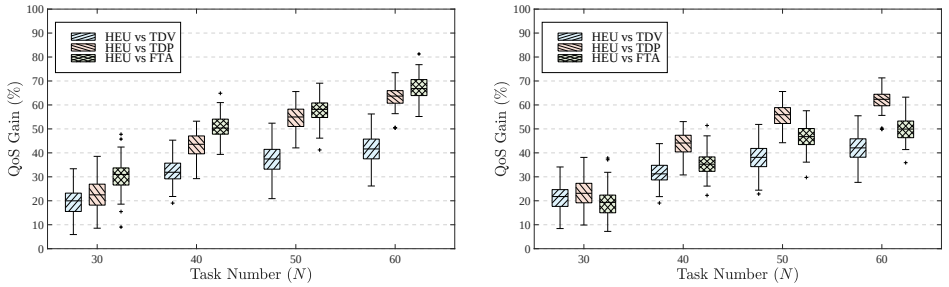
Figure 6(a) also shows that the energy consumption of OPT and TDV is almost the same. No matter whether DVFS+DPM or DVFS is applied to maximize QoS under time and energy constraints, the ES is E_b. Therefore, all available energy is consumed to execute as many optional subtasks as possible, thereby enhancing system QoS. If the task deployment problem is EA, i.e., the problem aims to minimize the total energy consumption, the energy consumption of DVFS+DPM could be lower than DVFS.

Figure 7 compares the system QoS achieved by the HEU, TDV, TDP, and FTA methods under different task numbers $N \in [30, 40, 50, 60]$, where the QoS gain is defined as $[QoS(HEU) - QoS(method)]/QoS(HEU)$, and *method* represents the TDV, TDP, or FTA. Under the given comparison and parameters, e.g., $N = 30$ and $\eta = 1$, we perform 20 times of experiments, and each time, we randomly generate a DAG with $N = 30$ tasks through TGFF [16], using the task parameters from [20]. The processor number is set to $M = 4$, and the energy and time parameters are set to $\alpha = 0.45$, $\beta = 0.45$, and $\delta = 0.45$. To show the influences of DVFS and DPM on system QoS, we introduce a parameter $\eta \in (0, 1]$ for processor θ_k, where the power of θ_k in the active state is $P^c_{kl} = P^s_{kl} + \eta P^d_{kl}$; when the value of η is changed, the weights of dynamic power P^d_{kl} and static power P^s_{kl} are changed as well.



(a) $\eta = 1$. (b) $\eta = 0.1$.

Fig. 7. QoS comparisons between heu, tdv, tdp, and fta under different n and η parameters.

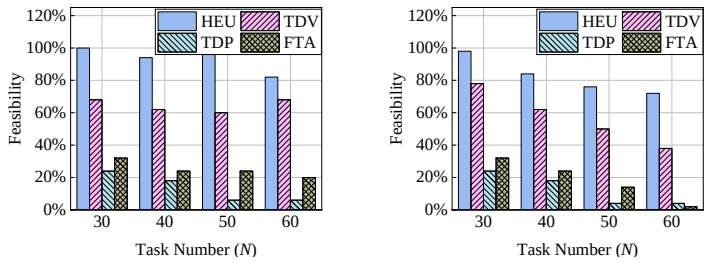


(a) $0.4 \leq \lambda_{ik} \leq 1$. (b) $0.8 \leq \lambda_{ik} \leq 1$.

Fig. 8. QoS comparisons between heu, tdv, tdp, and fta under different N and λ_{ik} parameters.

Figure 7 shows that HEU achieves a higher system QoS than TDV, TDP, and FTA (30.3%, 45.4%, and 48.7% on average). This is because the proposed HEU simultaneously optimizes DVFS, DPM, and task-to-core allocation. Although this joint design increases the number of optimization variables, it also enlarges the feasible region of the task deployment problem, thereby achieving a better task deployment solution. In Figure 7(a), we set $\eta = 1$, and the QoS gain between HEU and TDV is 25.1%. However, when we set $\eta = 0.1$ in Figure 7(b), the QoS gain between HEU and TDV changes to 35.4%. As the value of η decreases, HEU achieves a higher QoS gain over TDV. DVFS adjusts the voltage and frequency of task execution to reduce processor energy consumption in the active state, while DPM changes the processor state when no task is executing to reduce energy consumption in the idle and sleep states. When parameter η is small, the weight of dynamic power P_{kl}^d decreases, compared with static power P_{kl}^s. In addition, the power of the processor θ_k in active state is mainly influenced by P_{kl}^d. Therefore, the energy consumption of processor θ_k in the idle state increases, compared with that in the active state. By combining DVFS and DPM, the proposed HEU can save more energy than TDV, which achieves a higher system QoS gain.

Figure 8 evaluates the influence of execution efficiency factor λ_{ik} on QoS gain. The larger the range of λ_{ik}, the more obvious the heterogeneity of the platform is. The results show that HEU achieves a higher system QoS than TDV, TDP, and FTA (33.1%, 45.6%, and 44.7% on average). When the range of λ_{ik} varies from $[0.4, 1]$ to $[0.8, 1]$, the QoS gain between HEU and FTA reduces from 51.6% to 37.7%. This is because a smaller range of λ_{ik} indicates less processor heterogeneity, thereby reducing the benefit of optimizing task-to-core allocation.



(a) $\alpha = 0.25$, $\beta = 0.45$, $\delta = 0.45$. (b) $\alpha = 0.45$, $\beta = 0.25$, $\delta = 0.25$.

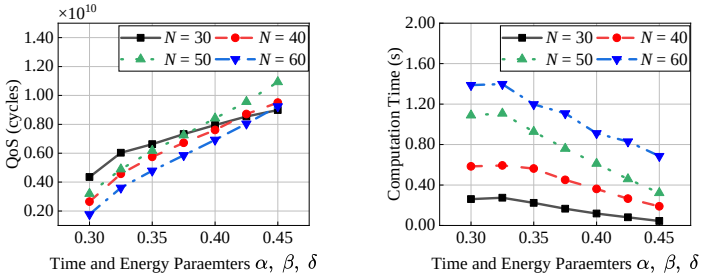
Fig. 9. Feasibility comparisons of heu, tdv, tdp, and fta under different N, α, β, and δ parameters.

Figure 9 compares the problem feasibility of HEU, TDV, TDP, and FTA methods under different task numbers $N \in [30, 40, 50, 60]$, where the processor number $M = 4$. To evaluate problem feasibility, we consider the system with restricted energy (time) constraints. Therefore, we set the tuned parameters $\alpha = 0.25$, $\beta = 0.45$, and $\delta = 0.45$ ($\alpha = 0.45$, $\beta = 0.25$, and $\delta = 0.25$). For each HEU, TDV, TDP, and FTA method, given the parameters α, β, and δ, we randomly generate 50 DAGs with N tasks for evaluation. The problem feasibility is given by $t_{success}/50$, where $t_{success}$ is the number of problems with feasible solutions.

Figure 9 shows that HEU achieves higher problem feasibility than TDV, TDP, and FTA under the same energy parameter α and time parameters β and δ. Specifically, the feasibility of HEU is 28.5%, 79.5%, and 68.0% (25.5%, 70.0%, and 64.5%) higher than that of TDV, TDP, and FTA under the same energy (time) parameters, respectively. This is because HEU combines the benefits of DVFS with DPM; it can save more energy compared with TDV and TDP, as they only consider DVFS or DPM, and thus, the additional energy can be used to enhance the supply voltage and working frequency of the processors, to make the time and energy constraints easier to satisfy. Compared with FTA, HEU optimizes task-to-core allocation and accounts for processor heterogeneity. Because processors have different task execution efficiencies (λ_{ik}), HEU can better allocate system resources to balance task execution time and energy.

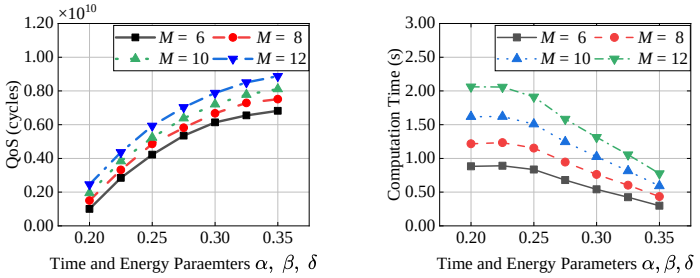
Figure 10(a) compares the QoS achieved by HEU under different task numbers $N \in [30, 40, 50, 60]$ and energy and time parameters $\alpha = \beta = \delta \in [0.3, 0.325, 0.35, 0.375, 0.4, 0.425, 0.45]$, and the processor number is set to $M = 4$. The result shows that the system QoS increases with task number N, and time and energy parameters α, β, and δ, as the higher the values of N, α, β, and δ, the more optional subtasks can be executed, due to more system resources. Figure 10(b) shows that the computation time of HEU increases with task number N. However, as the parameters α, β, and δ increase, the computation time of HEU decreases. On one hand, as task number N increases, the dimensions of optimization variables of PP (14) (e.g., q_{ijk}, p_{ijk}, c_i, o_i, and t_i^s) increase as well, which adds to the complexity of the problem. However, the increment is limited, and the computation time is usually within 1.5 s. On the other hand, when HEU uses Algorithm 2 to perform the adjustment of task τ_i, it first decreases optional cycles o_i, and then increases task execution frequency q_{ikl} until the energy constraint (9) and the time constraints (4), (5) are satisfied at the same time. Therefore, the smaller the parameters α, β, and δ, the harder the energy constraint (9) and time constraints (4), (5) are satisfied, and thus, increases the round number of task adjustment in Algorithm 2, i.e., increases the computation time of HEU.

Figure 11 compares the system QoS and computation time of HEU under different α, β, δ, and M parameters, where task number $N = 40$. With the processor number M increasing, more options to perform task-to-core allocation, i.e., more opportunities to exploit processor heterogeneity and



(a) System QoS. (b) Computation time.

Fig. 10. QoS and time comparisons of heu under different N, α, β, and δ parameters.



(a) System QoS. (b) Computation time.

Fig. 11. QoS and time comparisons of heu under different M, α, β, and δ parameters.

parallel execution, thereby improving QoS. In addition, increasing the value of M expands the solution space for TA, TS, FA, and processor state transitions, thereby increasing the time required to find a solution.

Finally, we use an ARM-based gem5 SE simulation platform to demonstrate the practical applicability of our HEU method. Specifically, we select the LE application as a case study and execute these tasks through gem5 under several processor/cache configurations (e.g., processor type and cache size) to profile these tasks. We can obtain parameters such as execution efficiency factor λ_{ik} and execution cycles M_i and O_i. Since task QoS is modeled as the cumulative optional cycles, i.e., $\sum_{i\in\mathcal{N}} o_i$, we execute the task with $o_i^{(\rho)} = \rho O_i$ optional cycles, where $\rho \in \{0.1, 0.2, \ldots, 1.0\}$, and record the corresponding task QoS. We use the above parameters in the design and the evaluation of HEU.

We first formulate the task deployment problem and solve it using the proposed HEU method in MATLAB. The obtained task deployment result is set as the input of gem5. When gem5 follows this result to perform task allocation and scheduling, we get the actual task execution time. On this basis, we can obtain the corresponding task QoS, real-time response, and energy consumption. By comparing these results with time and energy constraints, we can evaluate the quality of the solution.

Here, we mainly compare the achieved QoS and the consumed time and energy among HEU, FTA, and TDV through gem5. To this end, we introduce the normalized QoS $\sum_{i\in\mathcal{N}} o_i / \sum_{i\in\mathcal{N}} O_i$, and constraints E/E_b and T/H, to evaluate the performances of HEU, FTA, and TDV, where E and T are the actual energy consumption and makespan measured by gem5, respectively. A task deployment strategy is considered feasible only when both constraints $E/E_b \leq 1$ and $T/H \leq 1$ are

satisfied. For HEU, the QoS, energy, and time metrics are 0.72, 0.89, and 0.92, respectively; for FTA, the corresponding metrics are 0.54, 0.93, and 0.95, respectively; and for TDV, they are 0.65, 0.94, and 0.91, respectively. Although the gem5-based simulations account for practical issues such as cache interference [15, 37], the measured execution cycles remain within the WCEC range. The results show that HEU achieves higher QoS than FTA and TDV while satisfying both time and energy constraints.

7 Discussion and Extension

When DVFS is used to adjust task execution time and energy, it may lead to thermal- and reliability-related issues [29, 39]. On one hand, the thermal behavior of the multicore platforms, e.g., the heat transfer between the processor and the heat dissipation units, is a complex dynamic process influenced by various physical factors. This dynamic heat transfer process can usually be approximated using the RC model [6]:

$$\begin{aligned} 0 = &\sum_{n\in\mathcal{N}_m} [T(t,m) - T(t,n)] \cdot G_N(m,n) + C(m) \cdot \frac{dT(t,m)}{dt} \\ &+ [T(t,m) - T(t,h)] \cdot G_H(m) - \sum_{j\in\mathcal{N}} \alpha(t,j,m) \cdot P(j,m), \end{aligned} \tag{36}$$

where $T(t,m)$ ($T(t,n)$) represents the temperature of processor θ_m (θ_n) at time t, and $T(t,h)$ is the temperature of the heat element h attached to processor θ_m at time t. $G_H(m)$ represents the conduction between the processor and the heat element. $G_N(m,n)$ denotes the conduction among processors. $C(m)$ is the thermal capacitance of θ_m, N_m is the set of processors thermally adjacent to θ_m. In addition, $P(j,m)$ is the power consumption of processor θ_m when executing task τ_j, and is related to TA and FA variable q_{jml}, and $\alpha(t,j,m)$ is a binary variable indicating whether task τ_j is running on θ_m at time t.

The challenge of adding thermal-related issues to task deployment is how to handle the differential item $\frac{dT(t,m)}{dt}$. For problem-solving, one popular method for handling this item is to consider the steady-state temperature. Moreover, the continuous-time expression in Equation (36) is converted into a task-level discrete description [6]. Specifically, $T(t,m)$, $T(t,h)$, and $\alpha(t,j,m)$ are replaced by $T_{i,m}$ (the temperature of θ_k when executing τ_i), $T_{i,h}$ (the temperature of heat element h when executing τ_i), and $\beta(i,j,m)$ (τ_i is assigned to θ_k and the executions of τ_i and τ_j have time overlap), respectively. With the setting $\frac{dT(t,m)}{dt} = 0$, Equation (36) can be rewritten as follows:

$$T_{i,m} = T_{i,h} + \frac{1}{G_H(m)} \left[\sum_{j\in\mathcal{N}} \beta(i,j,m) \cdot P(j,m) \right] + \frac{1}{G_H(m)} \sum_{n\in\mathcal{N}_m} G_N(m,n) \cdot \left[T_{i,n} - T_{i,m} \right]. \tag{37}$$

Based on Equation (37), we can add the following temperature constraint to the task deployment problem.

$$T_{i,m} \le T_{th}, \ \forall i \in \mathcal{N}, \ \forall m \in \mathcal{M}, \tag{38}$$

where T_{th} is the temperature threshold. From the above statement, we can see that binary variables $\beta(i,j,m)$ and q_{jml} in $P(j,m)$ are coupled nonlinearly in Equation (37). We can use *Lemma* 4.3 to handle these nonlinear items.

On the other hand, the reliability issue, e.g., transient fault, is related to the processor state during the task execution process. Specially, when task τ_i with C_i cycles is assigned to processor θ_k and executed with V/F level (v_{kl}, f_{kl}), the task reliability [33] is

$$r_{ikl} = e^{-\eta \times 10^{\frac{d(f_{\max} - f_{kl})}{f_{\max} - f_{\min}}} \times \frac{C_i}{f_{kl}}}, \tag{39}$$

where η is the maximum error-tolerant rate, d is the sensitivity of the failure rate corresponding to the frequency scaling, and $f_{\max}$ and $f_{\min}$ are the highest and lowest frequencies of the processor, respectively.

Taking TA and FA variable q_{ikl} into account, the reliability of τ_i is given by $r_i = \sum_{k\in\mathcal{M}} \sum_{l\in\mathcal{L}} q_{ikl} r_{ikl}$. If the reliability of task τ_i is lower than the reliability threshold R_{th}, we perform task duplication to enhance task reliability. To this end, we need to introduce an additional (binary) variable h_i to indicate the existence of task τ_i.

If the reliability of the original task τ_i is larger than the threshold R_{th}, i.e., $r_i \geq R_{th}$, there is no need to duplicate task τ_i, and task τ_{i+N} (duplication of τ_i) does not exist, i.e., $h_{i+N} = 0$; otherwise, task τ_i is duplicated, and task τ_{i+N} exists, i.e., $h_{i+N} = 1$. The following constraint can be used to describe the above cases:

$$h_{i+N} = \begin{cases} 0, & r_i \geq R_{th} \\ 1, & r_i < R_{th} \end{cases}, \ \forall i \in \mathcal{N}. \tag{40}$$

Note that with task duplication, the reliability of task τ_i is changed to $r_i' = 1 - (1 - h_i r_i)(1 - h_{i+N} r_{i+N})$. To satisfy the reliability constraint, we set:

$$r_i' \geq R_{th}, \ \forall i \in \mathcal{N}. \tag{41}$$

To deal with the logical comparison in Equation (40), we can use *Lemma* 4.1 to transform constraint (40) into a linear function, so as to incorporate the reliability issue into the task deployment problem. Similarly, we can use *Lemma* 4.3 to handle the binary-based nonlinear items like $h_i q_{ikl}$.

From the above statement, we can see that when thermal- and reliability-related issues are involved in the task deployment problem, we need to introduce additional variables (e.g., $\beta(i, j, m)$ and h_i) and constraints (e.g., temperature constraint (38) and reliability constraint (41)) into the problem. We also need to address the nonlinearities arising from thermal and reliability constraints, which greatly complicate our original task deployment problem. We will consider these issues in future work.

8 Conclusion

This article considers the IC task model and optimizes task allocation and scheduling, frequency allocation, and working status simultaneously for the heterogeneous multicore platforms that support DVFS and DPM to maximize system QoS under energy and real-time constraints. Firstly, the task deployment problem is formulated as an MINLP and converted to an MILP using the linearization method to obtain the optimal solution. Secondly, to improve the scalability of the proposed method, we divide the task deployment problem into two subproblems: (1) TA and FA, and (2) TS and optional cycle adjustment, and solve these subproblems in sequence. Our method has a shorter computation time because it decomposes the problem and combines polling and greedy methods. Finally, the simulation results show that the proposed IC task deployment method for the heterogeneous multicore platform, which combines DVFS and DPM, achieves higher system QoS than existing methods (41.30% on average). Moreover, the computation time of the heuristic method is negligible compared to the optimal method, with an acceptable performance loss (28.18% on average). In future work, we will extend our method to scenarios considering thermal- and reliability-related issues.

References

[1] Muhammad Ali Awan, Patrick Meumeu Yomsi, Geoffrey Nelissen, and Stefan M. Petters. 2016. Energy-aware task mapping onto heterogeneous platforms using DVFS and sleep states. *Real-Time Syst.* 52 (2016), 450–485.

[2] Hakan Aydin, Rami Melhem, Daniel Mosse, and Pedro Mejia-Alvarez. 2001. Optimal reward-based scheduling for periodic real-time tasks. *IEEE Trans. Comput.* 50, 2 (2001), 111–130.

[3] Mohammed Redha Bouzidi, Mourad Daoudi, Benameur Ziani, Kamel Boukhalfa, Chaker Abdelaziz Kerrache, and Nasreddine Lagraa. 2021. FAMOBACH: A fast and survivable workflow scheduling approach based MOHEFT using backtacking and checkpointing. *Comput. Commun.* 171 (2021), 16–27.
[4] Daniel Casini, Paolo Pazzaglia, Alessandro Biondi, and Marco Di Natale. 2022. Optimized partitioning and priority assignment of real-time applications on heterogeneous platforms with hardware acceleration. *J. Syst. Archit.* 124 (2022), 102416.
[5] 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 Trans. Embed. Comput. Syst.* 20, 5s (2021), 1–25.
[6] Thidapat Chantem, Robert P. Dick, and X. Sharon Hu. 2008. Temperature-aware scheduling and assignment for hard real-time applications on MPSoCs. In *Proc. IEEE/ACM DATE*. 288–293.
[7] Thidapat Chantem, X. Sharon Hu, and Robert P. Dick. 2011. Temperature-aware scheduling and assignment for hard real-time applications on MPSoCs. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* 19, 10 (2011), 1884–1897.
[8] Gang Chen, Kai Huang, and Alois Knoll. 2014. Energy optimization for real-time multiprocessor system-on-chip with optimal DVFS and DPM combination. *ACM Trans. on Embedded Comput. Syst.* 13, 3 (2014), 1–21.
[9] Long Chen, Xiaoping Li, Yucheng Guo, and Rubén Ruiz. 2021. Hybrid resource provisioning for cloud workflows with malleable and rigid tasks. *IEEE Trans. Cloud Comput.* 9, 3 (2021), 1089–1102.
[10] Hoon Sung Chwa, Jaebaek Seo, H. Yoo, Jinkyu Lee, and I. Shin. 2015. Energy and feasibility optimal global scheduling framework on big.LITTLE platforms. In *Proc. IEEE Real-Time Scheduling Open Problems Seminar*. 1–11.
[11] Moïse W. Convolbo and Jerry Chou. 2016. Cost-aware DAG scheduling algorithms for minimizing execution cost on cloud resources. *J. Supercomput.* 72, 3 (2016), 985–1012.
[12] Luis Alejandro Cortés, Petru Eles, and Zebo Peng. 2006. Quasi-static assignment of voltages and optional cycles in imprecise-computation systems with energy considerations. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* 14, 10 (2006), 1117–1129.
[13] Minyu Cui, Angeliki Kritikakou, Lei Mo, and Emmanuel Casseau. 2021. Fault-tolerant mapping of real-time parallel applications under multiple DVFS schemes. In *Proc. IEEE RTAS*. 387–399.
[14] Mayuri Digalwar, Biju K. Raveendran, and Sudeept Mohan. 2017. LAMCS: A leakage aware DVFS based mixed task set scheduler for multi-core processors. *Sustain. Comput.: Inform. and Syst.* 15 (2017), 63–81.
[15] Alessandro Druetto, Enrico Bini, Andrea Grosso, Stefano Puri, Silvio Bacci, Marco Di Natale, and Francesco Paladino. 2023. Task and memory mapping of large size embedded applications over NUMA architecture. In *Proc. Int. Conf. Real-Time Netw. Syst.* 166–176.
[16] Amirhossein Esmaili, Mahdi Nazemi, and Massoud Pedram. 2019. Energy-aware scheduling of task graphs with imprecise computations and end-to-end deadlines. *ACM Trans. Des. Autom. Electron. Syst.* 25, 1 (2019), 1–21.
[17] Amirhossein Esmaili, Mahdi Nazemi, and Massoud Pedram. 2019. Modeling processor idle times in MPSoC platforms to enable integrated DPM, DVFS, and task scheduling subject to a hard deadline. In *Proc. IEEE/ACM ASP-DAC*. 532–537.
[18] Marc Fehling and Wolfgang Bangerth. 2023. Algorithms for parallel generic hp-adaptive finite element software. *ACM Trans. Math. Softw.* 49, 3 (2023), 1–26.
[19] Jingyu He, Yao Xiao, Corina Bogdan, Shahin Nazarian, and Paul Bogdan. 2021. A design methodology for energy-aware processing in unmanned aerial vehicles. *ACM Trans. Des. Autom. Electron. Syst.* 27, 4 (2021), 1–20.
[20] Ramkumar Jayaseelan and Tulika Mitra. 2008. Temperature aware task sequencing and voltage scaling. In *Proc. IEEE/ACM ICCAD*. 618–623.
[21] Qingye Jiang, Young Choon Lee, Manuel Arenaz, Luke M. Leslie, and Albert Y. Zomaya. 2014. Optimizing scientific workflows in the cloud: A montage example. In *Proc. IEEE/ACM UCC*. 517–522.
[22] Hamayun Khan, Irfan Ud din, Arshad Ali, and Sami Alshmrany. 2022. Energy-efficient scheduling based on task migration policy using DPM for homogeneous MPSoCs. *Comput., Mater. and Continua* 74, 1 (2022), 965–981.
[23] Youngmin Kim, KiSeong Lee, and ChanGun Lee. 2017. Energy efficient real-time scheduling using DPM on mobile sensors with a uniform multi-cores. *Sensors* 17, 12 (2017), 1–22.
[24] Fanxin Kong, Wang Yi, and Qingxu Deng. 2011. Energy-efficient scheduling of real-time tasks on cluster-based multicores. In *Proc. IEEE/ACM DATE*. 1–6.
[25] Dawei Li and Jie Wu. 2015. Minimizing energy consumption for frame-based tasks on heterogeneous multiprocessor platforms. *IEEE Trans. Parallel Distrib. Syst.* 26, 3 (2015), 810–823.
[26] Xinmei Li, Lei Mo, Angeliki Kritikakou, and Olivier Sentieys. 2022. Approximation-aware task deployment on heterogeneous multicore platforms with DVFS. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.* 42, 7 (2022), 2108–2121.
[27] Isabel Mendez-Diaz, Javier Orozco, Rodrigo Santos, and Paula Zabala. 2017. Energy-aware scheduling mandatory/optional tasks in multicore real-time systems. *Int. Trans. Oper. Res.* 24, 12 (2017), 173–198.

[28] Lei Mo, Angeliki Kritikakou, and Olivier Sentieys. 2018. Energy-quality-time optimized task mapping on DVFS-enabled multicores. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.* 37, 11 (2018), 2428–2439.
[29] Sanjay Moulik and Yanshul Sharma. 2024. FRESH: Fault-tolerant real-time scheduler for heterogeneous multiprocessor platforms. *Future Gener. Comput. Syst.* 161 (2024), 214–225.
[30] David R. Musser. 1997. Introspective sorting and selection algorithms. *Softw. Pract. Exper.* 27, 8 (1997), 983–993.
[31] Gereon Onnebrink, Florian Walbroel, Jonathan Klimt, Rainer Leupers, Gerd Ascheid, Luis Gabriel Murillo, Stefan Schürmans, Xiaotao Chen, and YwhPyng Harn. 2017. DVFS-enabled power-performance trade-off in MPSoC SW application mapping. In *Proc. SAMOS*. 196–202.
[32] Sangyoung Park, Jaehyun Park, Donghwa Shin, Yanzhi Wang, Qing Xie, Massoud Pedram, and Naehyuck Chang. 2013. Accurate modeling of the delay and energy overhead of dynamic voltage and frequency scaling in modern microprocessors. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.* 32, 5 (2013), 695–708.
[33] Jiwu Peng, Kenli Li, Jianguo Chen, and Keqin Li. 2022. Reliability/performance-aware scheduling for parallel applications with energy constraints on heterogeneous computing systems. *IEEE Trans. Sustain. Comput.* 7, 3 (2022), 681–695.
[34] Weiming Qiu, Yonghao Chen, Dihu Chen, Tao Su, and Simei Yang. 2022. Run-time hierarchical management of mapping, per-cluster DVFS and per-core DPM for energy optimization. *Electronics* 11, 7 (2022), 1094–1112.
[35] Nayereh Rasouli, Cristian Klein, and Erik Elmroth. 2025. Resource management for mission-critical applications in edge computing: Systematic review on recent research and open issues. *ACM Comput. Surv.* 58, 3 (2025), 1–37.
[36] Federico Reghenzani, Ashikahmed Bhuiyan, William Fornaciari, and Zhishan Guo. 2021. A multi-level DPM approach for real-time DAG tasks in heterogeneous processors. In *Proc. IEEE RTSS*. 14–26.
[37] Ikram Senoussaoui, Giuseppe Lipari, Houssam-Eddine Zahaf, and Mohammed Kamal Benhaoua. 2024. Memory-processor co-scheduling of AECR-DAG real-time tasks on partitioned multicore platforms with scratchpads. *J. Syst. Archit.* 150 (2024), 103117.
[38] Edwin Sha, Li Wang, Qingfeng Zhuge, Jun Zhang, and Jing Liu. 2015. Power efficiency for hardware/software partitioning with time and area constraints on MPSoC. *Int. J. Parallel. Program.* 43, 3 (2015), 381–402.
[39] Yanshul Sharma and Sanjay Moulik. 2022. RT-SEAT: A hybrid approach based real-time scheduler for energy and temperature efficient heterogeneous multicore platforms. *Results Eng.* 16 (2022), 100708.
[40] K. Siddesha and G. V. Jayaramaiah. 2022. Energy-aware task scheduling approach using DVFS and particle swarm optimization for heterogeneous multicore processors. In *Proc. ERCICA*. 943–955.
[41] Mason Thammawichai and Eric C. Kerrigan. 2018. Energy-efficient real-time scheduling for two-type heterogeneous multiprocessors. *Real-Time Syst.* 54, 1 (2018), 132–165.
[42] Fakhruddin Muhammad Mahbub ul Islam, Man Lin, Laurence T. Yang, and KimKwang Raymond Choo. 2018. Task aware hybrid DVFS for multi-core real-time systems using machine learning. *Inf. Sciences* 433 (2018), 315–332.
[43] Andrew Vince. 2002. A framework for the greedy algorithm. *Discrete Appl. Math.* 121, 1-3 (2002), 247–260.
[44] Tongquan Wei, Junlong Zhou, Kun Cao, Peijin Cong, Mingsong Chen, Gongxuan Zhang, Xiaobo Sharon Hu, and Jianming Yan. 2017. Cost-constrained QoS optimization for approximate computation real-time tasks in heterogeneous MPSoCs. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.* 37, 9 (2017), 1733–1746.
[45] Jingjing Yao and Nirwan Ansari. 2021. Enhancing federated learning in fog-aided IoT by CPU frequency and wireless power control. *IEEE Internet Things J.* 8, 5 (2021), 3438–3445.
[46] Rong Ye and Qiang Xu. 2014. Learning-based power management for multicore processors via idle period manipulation. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.* 33, 7 (2014), 1043–1055.
[47] 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.
[48] Gang Zeng, Yutaka Matsubara, Hiroyuki Tomiyama, and Hiroaki Takada. 2016. Energy-aware task migration for multiprocessor real-time systems. *Future Gener. Comput. Syst.* 56 (2016), 220–228.
[49] 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 *Proc. DATE*. 1402–1407.

Received 19 January 2026; revised 24 April 2026; accepted 29 May 2026