Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

QoS-Efficient and Near-Optimal Imprecise Computation Task Deployment for Heterogeneous Multicore Systems

Linlin Zhao
Southeast University
Xinmei Li
Southeast University
Minyu Cui
Linnaeus University
Lei Mo
lmo@seu.edu.cn

Southeast University

Research Article

Keywords: Heterogeneous multicores, imprecise computation tasks, quality-ofservice, problem decomposition

Posted Date: November 9th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11416230/v1

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

Additional Declarations: No competing interests reported.

Real-Time Systems manuscript No.
(will be inserted by the editor)

QoS-Efficient and Near-Optimal Imprecise Computation Task Deployment for Heterogeneous Multicore Systems

Abstract Heterogeneous multicore platforms are widely used in many applications due to their customizability, where energy efficiency, real-time response, and system quality-of-service (QoS) are critical. Imprecise computation (IC) tasks provide an efficient architecture to balance these conflicting requirements. Existing methods seldom consider deploying IC tasks on heterogeneous platforms under complex scenarios and providing an efficient way to find proper solutions. This paper studies the joint-design problem of mapping and scheduling dependent IC tasks on heterogeneous multicores. The target problem is formulated as a mixed-integer nonlinear program (MINLP) that maximizes system QoS under energy supply, task deadline, precedence, and non-overlap constraints. To solve the problem efficiently, we propose a decomposition-based solution, including a master problem (MP) that handles task allocation and execution cycle adjustment, a scheduling problem (SP) that focuses on dependency-aware task scheduling, and a closed-loop feedback mechanism from SP to MP to improve task schedulability. Experimental results show that the proposed method achieves 95.25% of the optimal QoS obtained by a commercial solver while using only about 19.8% of its computation time, demonstrating a favorable trade-off between solution quality and computational efficiency.

Keywords Heterogeneous multicores · imprecise computation tasks · quality-of-service · problem decomposition

1 Introduction

Heterogeneous multicore platforms are widely used in current embedded and mobile computing systems [13, 35], e.g., automated guided vehicles (AGVs) and unmanned aerial vehicles (UAVs). Multiple cores can execute the tasks in parallel to improve task execution efficiency, and they provide heterogeneity for different tasks [16]. Applications are usually composed of dependent and real-time tasks, which can be modeled as directed acyclic graphs (DAGs) [18, 29]. By deploying these tasks on the platforms, the desired system performance can be achieved [27],

e.g., federated learning and UAV flight control. Because tasks usually have real-time requirements while platforms are often limited by system resources (e.g., process capacity and energy supply), task deployment must satisfy both timing and energy constraints [6]. Energy efficiency and task deadlines are important, but these are conflicting objectives, since enhancing real-time execution often requires consuming more energy [37]. Task allocation (on which processor each task is executed) and task scheduling (when each task starts and ends its execution) should be considered jointly to satisfy energy supply, task deadline, precedence, and non-overlap constraints.

Imprecise computation (IC) [17] is an efficient architecture to balance output quality, execution time, and energy consumption. Executing more task cycles improves output quality at the cost of additional task execution time and energy. Each task consists of a mandatory part and an optional part [19]. The mandatory part guarantees the baseline QoS, while the optional part further improves QoS when more resources are available [20]. For the IC tasks, we should further consider the optional cycle adjustment in the task deployment problem.

Existing methods mainly include energy- and QoS-aware task deployments. Energy-aware task deployment methods minimize energy consumption while preserving time and dependency constraints [8, 11, 15, 23, 24, 30, 32]; however, most of them consider fixed task workloads, and thus, the QoS improvements are limited. In contrast, QoS-aware task deployment methods consider IC tasks to enhance system QoS [2, 10, 21, 22, 25, 26, 31, 33, 36, 38], but platform heterogeneity, task dependency, time and/or energy constraints are not comprehensively considered. In addition, most IC studies use a linear QoS model and assume that QoS increases linearly with optional execution [10,22,31,33,36,38]. Although the linear model has a simple structure, it cannot cover a more general and realistic case, e.g., the QoS improvement is saturated when more optional cycles are executed. In addition, the nonlinear items in the task deployment problem bring difficulties in the solving process, making the computation time and solution quality hard to balance.

Compared with existing works, we consider maximizing the system QoS by mapping IC tasks on a heterogeneous multicore platform under real-time response, task dependency and non-overlapping, and energy-supply constraints. The original problem is a mixed-integer nonlinear program (MINLP), because binary allocation and ordering decision variables interact with continuous optional-execution variables and nonlinear QoS functions. To make the problem computationally tractable, we develop a problem decomposition solving framework. The master problem (MP) determines task-to-core allocation and optional-execution adjustment, while the scheduling problem (SP) considers task sequence and task start and end time, and further, the closed-loop feedback mechanism updates the results of the MP according to the SP solution. The main contributions of this paper can be summarized as follows.

- *Joint-Design Problem Formulation.* QoS-aware task deployment problem on heterogeneous multicore platforms is formulated as an MINLP, which jointly optimizes task-to-core allocation, optional-cycle adjustment, and task sequence and scheduling, under multiple constraints.
- *Decomposition-Based Solution.* The original problem is decomposed into an MP and an SP, and solved in sequence through problem transformation, dual de-

Table 1: Task Deployment Methods

Ref.	**Obj.**		**Plat.**		**Fun.**		**Cons.**			
	EA	QA	HO	HE	LF	NF	RT	ES	DE	NO
[15]	✓		✓				✓		✓	✓
[11]	✓		✓				✓	✓	✓	✓
[30]	✓			✓			✓			✓
[32]	✓			✓			✓			✓
[23]	✓			✓			✓			✓
[24]	✓			✓			✓			✓
[8]	✓			✓			✓		✓	✓
[2]		✓	✓		✓	✓	✓			✓
[22]		✓	✓		✓		✓		✓	✓
[26]		✓	✓		✓		✓	✓	✓	✓
[25]		✓	✓		✓		✓	✓		✓
[31]		✓		✓	✓		✓			✓
[36]		✓		✓	✓		✓	✓		✓
[38]		✓		✓	✓		✓	✓		✓
[33]		✓		✓	✓		✓	✓		✓
[10]		✓	✓		✓		✓	✓	✓	✓
[21]		✓		✓		✓	✓	✓		✓
Proposed		✓		✓		✓	✓	✓	✓	✓

EA: energy minimization; QA: QoS maximization; HO: homogeneous multicore; HE: heterogeneous multicore; LF: linear QoS function; NF: nonlinear QoS function; RT: real-time constraint; ES: energy constraint; DE: task dependency; NO: task non-overlapping.

composition, and a sorting algorithm based on DAG structure to reduce computational complexity.
– *Constraint-Aware Feedback Mechanism.* A feedback mechanism is introduced to adjust the MP result based on the constraint violation of the SP solution, in order to improve the solution quality and task schedulability of the whole problem.
– *Performance Evaluation.* Experimental results show that our method achieves 95.25% of the optimal QoS obtained by the Gurobi solver while using only 19.8% of its computation time. Compared with existing methods, the feedback mechanism improves QoS and task schedulability for complex DAG scenarios.

The rest of this paper is organized as follows. Section 2 reviews the related works. Section 3 introduces the system model and formulates the joint optimization problem. Section 4 presents the proposed decomposition-based solution framework. Section 5 shows the experimental evaluations. Section 6 concludes the paper.

2 RELATED WORK AND BACKGROUND

Table 1 compares some representative studies on multicore task deployment problems. The objective can be classified as energy-aware (EA) or QoS-aware (QA). The platform can be classified as homogeneous (HO) or heterogeneous (HE). The QoS is described by a linear (LF) or nonlinear (NF) function. The constraints include real-time requirements (RT), energy supply (ES), task dependency (DE), and task non-overlapping (NO).

2.1 Energy-Aware Task Deployment

Energy-aware task deployment mainly focuses on minimizing energy consumption under application and platform requirements. For homogeneous multicore platforms, Energy consumption and deadline-missing ratio are reduced in [15] for the application tasks described by a DAG. Considering the battery-powered embedded platforms, the computation and communication costs among the cores are optimized in [11] when handling the dependent and real-time tasks. Task dependency enforces task scheduling to follow precedence relations and avoid execution conflicts. The target problems are usually formulated by mathematical programming combined with allocation and scheduling rules.

For heterogeneous multicore platforms, task-to-core allocation couples with task execution time and energy consumption because different cores have different execution characteristics. Energy consumption for executing independent periodic tasks under deadline constraints is minimized in [30] through a task partition method. Task allocation, scheduling, and task data processing are jointly optimized in [32] for real-time tasks by integer linear programming (ILP) and greedy-based heuristics. ILP formulations are also used in [23, 24] to minimize the energy consumption of large-scale independent task sets. The above methods do not consider task dependency. In contrast, dependent tasks are considered in [8], and a task mapping method is developed for heterogeneous multicore systems. Nevertheless, the above methods consider fixed task workloads (precise computation tasks) and thus cannot perform optional-execution adjustment to improve system QoS.

2.2 QoS-Aware Task Deployment

QoS-aware task deployment improves system QoS through optional-execution adjustment under resource constraints. On homogeneous multicore platforms, [2] optimizes the rewards of independent real-time tasks under linear and nonlinear concave reward functions. [22] uses a linear QoS model and considers intra-task and inter-task dependencies when performing optional execution adjustment. [26] and [25] maximize the QoS of approximate real-time tasks through hybrid task scheduling methods. With a linear QoS model and dependent IC tasks, [10] jointly considers deadline, energy, precedence, and non-overlap constraints. Although the above methods consider linear/nonlinear QoS models or task dependency, they mainly focus on homogeneous platforms. For heterogeneous platforms, task-to-core allocation is coupled with other variables, e.g., optional-execution adjustment.

On heterogeneous multicore platforms, a task allocation and scheduling method is developed in [31] for independent IC tasks. QoS for independent imprecise or approximate computation tasks under time and energy constraints are optimized in [33,36,38], but they consider linear QoS models. In contrast, a nonlinear concave QoS function is considered in [21] to describe the diminishing gain from additional optional cycles and determine task-to-core allocation and optional-execution adjustment on heterogeneous platforms; however, the dependency among the tasks is not considered. The extensions from independent to dependent tasks are not straightforward, especially involving heterogeneity and nonlinear QoS functions, as complex coupling among the variables makes the problem difficult to solve.

The energy-aware and QoS-aware task deployment problems are well-known NP-hard problems, as they usually involve a set of binary/integer/continuous variables (e.g., task allocation and sequence, task start and end time); finding an optimal solution is time-consuming. Heuristics are popular methods used to solve complex optimization problems, e.g., [32, 38]. To reduce computation time, they usually adopt a multi-step optimization, i.e., to decouple the variables and to determine their values in sequence. Although they can find feasible solutions in a short amount of time, the solution qualities are hard to guarantee. Compared with existing optimal and heuristic methods, we propose a near-optimal method to handle our task deployment problem, balancing computation time and solution quality.

3 System Model and Problem Formulation

3.1 IC Task Model

Definition 3.1 *A task τ_i is defined as an IC task when it can be logically divided into a mandatory subtask and an optional subtask. The mandatory subtask executes M_i cycles to generate the baseline QoS level. The optional subtask executes o_i cycles to improve the QoS.*

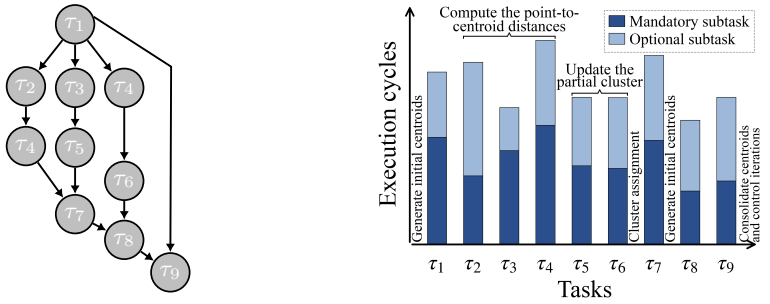
For the sake of problem formulation, a real-time application $\mathcal{A}$ is modeled as a five-element tuple $\{\mathcal{T}, \mathcal{E}, M, O, D\}$, where $\mathcal{T} = \{\tau_1, \ldots, \tau_N\}$ is the set of N tasks, each task τ_i is released at time 0 and sharing a common deadline H (also known as scheduling horizon); $\mathcal{E}$ is the set of directed edges e_{ij}, where $e_{ij} \in \mathcal{E}$ implies task τ_j cannot start until τ_i completes; each task τ_i is an IC task divided into a mandatory subtask with M_i cycles to achieve the baseline QoS, and an optional subtask with o_i cycles bounded by $0 \leq o_i \leq O_i$ to enhance QoS, where O_i is an upper bound. The total execution cycle number of τ_i is given by $M_i + o_i$. M_i and o_i are measured by the worst-case execution cycles (WCEC).

Linear and *nonlinear* functions are widely used QoS representations in the literature [1], as they can capture the behavior of many application areas, such as image and speech processing, control engineering, and automatic target recognition. In this paper, we mainly consider a nonlinear QoS function.

Because the tasks are both dependent and real-time, task mapping must satisfy the following timing requirements:

- All tasks must complete within the scheduling horizon H;
- For each precedence edge $e_{ij} \in \mathcal{E}$, task τ_j can start only after τ_i finishes, i.e., $t_j^s \geq t_i^e$;
- The computation time of task τ_i is small than its relative deadline D_i but still within the scheduling horizon H, i.e., $D_i \leq H$.

For IC tasks, executing more optional cycles generally improves QoS but also consumes additional time and energy. Therefore, the proposed task mapping algorithm jointly optimizes task scheduling and allocation with precedence and QoS-energy trade-offs, which are critical for resource-constrained heterogeneous systems.



(a) DAG of K-means

(b) Imprecise computation tasks of K-means

Fig. 1: K-means application tasks.

For example, *K-means* applications can be modeled as IC tasks [3], since it is an iterative algorithm; the more iterations are executed, the higher the classification accuracy can be achieved. In addition, K-means can tolerate arithmetic inaccuracies while maintaining comparable clustering performance.

The task DAG of K-means is illustrated in Fig. 1, where the mandatory subtask (M_1 cycles) of task τ_1 generates initial centroids, while the optional subtask ($o_1 \leq O_1$) refines this initialization. Then, tasks τ_2–τ_6 iteratively compute the point-to-centroid distances and update the partial cluster statistics, in which mandatory subtasks provide the coarse but acceptable results and optional subtasks progressively improve centroid accuracy. Finally, task τ_7 performs cluster assignment, while tasks τ_8 and τ_9 consolidate centroids and control iterations, respectively; their mandatory subtasks ensure basic correctness, and their optional subtasks enhance clustering precision.

3.2 Multicore Platform Model

We consider a multicore platform with M heterogeneous processors $\{\theta_1, \ldots, \theta_M\}$. The energy consumption of each processor θ_k includes both static and dynamic components. On the one hand, the dynamic power $P_k^d = C_k^{\text{eff}} v_k^2 f_k$ of processor θ_k is determined by its clock frequency f_k and supply voltage v_k, where C_k^{eff} denotes the average effective switching capacitance of θ_k, capturing the power consumed by circuit activity during task execution.

On the other hand, the static power P_k^s of θ_k is counted over the whole scheduling horizon H, including the time when the processor executes tasks and the time when it is idle but still powered on. Since static power is mainly caused by leakage currents, we adopt the following leakage model [19]:

$$P_k^s = v_k^t K_1 e^{K_2 v_k^e} e^{K_3 v_k^{\text{bs}}} + |v_k^{\text{bs}}| I_j, \tag{1}$$

where K_1, K_2, and K_3 are the technology-dependent constants; v_k^t is the thermal voltage; v_k^e is the effective gate voltage; v_k^{bs} is the reverse-bias voltage treated as constant for leakage reduction; and I_j is the approximately constant junction leakage current.

Therefore, for processor θ_k, the total power consumption during task execution is the sum of static and dynamic components: $P_k^c = P_k^s + P_k^d$. To characterize task

Table 2: Main Symbols Used in Problem Formulation

Symbol	Description
Parameters	
M	Number of processors
N	Number of tasks
λ_{ik}	Execution efficiency of task τ_i on processor θ_k
v_k	Voltage of θ_k
f_k	Frequency of θ_k
D_i	Relative deadline of τ_i
H	Scheduling horizon
E_s	Energy supply of platform
M_i	Mandatory cycles of τ_i
O_i	Maximum optional cycles of τ_i
e_{ij}	Task dependency ($e_{ij} = 1$ if τ_i precedes τ_j)
P_k^d	Dynamic power of θ_k
P_k^s	Static power of θ_k
p_i, q_i	QoS coefficients of τ_i
Binary Variables	
s_{ik}	$= 1$ if τ_i is assigned to θ_k, else $= 0$
u_{ij}	$= 1$ if τ_i and τ_j are allocated to the same processor and τ_i is executed before τ_j, else $= 0$
Continuous Variables	
t_i^s	Start time of τ_i
t_i^e	End time of τ_i
o_i	Optional cycles of τ_i
Auxiliary Variables	
t_{ik}	Execution time of τ_i's optional subtask on θ_k

execution efficiency on heterogeneous processors, we introduce a parameter $\lambda_{ik} \in (0, 1]$ (also known as a *heterogeneity factor*) [28], which describes how efficiently processor θ_k executes task τ_i relative to an idealized baseline. When task τ_i with $M_i + o_i$ cycles is assigned to processor θ_k working on voltage v_k and frequency f_k, the corresponding execution time of τ_i is then given by $\frac{M_i+o_i}{\lambda_{ik} f_k}$.

3.3 Problem Formulation

The objective is to maximize system QoS by mapping a set of dependent tasks onto heterogeneous processors while accounting for real-time and energy supply constraints. In this context, we determine the following decisions:

- On which processor should the task be executed (task-to-processor allocation);
- How many optional cycles should be executed (task cycles adjustment);
- When should the task start (task start time determination).

To formulate the task mapping problem, we introduce the binary and continuous variables summarized in Table 2. Let $\mathcal{N} \triangleq \{1, \ldots, N\}$ and $\mathcal{M} \triangleq \{1, \ldots, M\}$ denote the sets of tasks and processors, respectively. The objective function and the constraints are explained as follows.

3.3.1 Objective Function

Our goal is to maximize QoS under multiple system constraints, where a general nonlinear QoS function [9] is adopted here to explicitly capture the non-decreasing relationship between QoS and optional cycles o_i.

$$g(o) = \sum\nolimits_{i\in\mathcal{N}} \left(p_i o_i - q_i\sqrt{o_i}\right), \tag{2}$$

with task QoS coefficients p_i and q_i.

3.3.2 Task Allocation Constraint

Without considering task migration [14], each task is assigned to exactly one processor for execution:

$$\sum\nolimits_{k\in\mathcal{M}} s_{ik} = 1, \quad \forall i \in \mathcal{N}. \tag{3}$$

3.3.3 Task Real-Time Constraints

Taking time $\frac{M_i+o_i}{\lambda_{ik} f_k}$ and task allocation decision s_{ik} into account, the execution time of task τ_i is given by $t_i^{\text{comp}} = \sum_{k\in\mathcal{M}} \left(s_{ik}\frac{M_i+o_i}{\lambda_{ik} f_k}\right)$. Since the execution time of each τ_i cannot exceed its relative deadline D_i, and the end time of τ_i must remain within the scheduling horizon H, we have the following time constraints:

$$t_i^{\text{comp}} \le D_i, \quad \forall i \in \mathcal{N}, \tag{4}$$

$$t_i^{e} \le H, \quad \forall i \in \mathcal{N}. \tag{5}$$

3.3.4 Energy Supply Constraint

Since dynamic power P_k^d is mainly consumed by task execution, while static power P_k^s is consumed across the scheduling horizon H to maintain the basic circuit operation; moreover, the total energy consumption across all processors during the scheduling horizon H must be smaller than the available energy supply E_s, we get

$$E^{\text{comp}} = \sum\nolimits_{k\in\mathcal{M}} \left[\sum\nolimits_{i\in\mathcal{N}} \left(s_{ik}\frac{M_i+o_i}{\lambda_{ik} f_k}\right) P_k^d + H P_k^s\right] \le E_s. \tag{6}$$

3.3.5 Task Dependency Constraint

If task τ_i is a predecessor of τ_j (i.e., $e_{ij} = 1$), τ_j starts only after τ_i completes:

$$t_j^s + (1 - e_{ij})H \ge t_i^s + t_i^{\text{comp}}, \quad \forall i \ne j \in \mathcal{N}, \tag{7}$$

where t_i^s is the start time of τ_i. Note that (7) only restricts dependent tasks τ_i and τ_j. If τ_i and τ_j are independent, i.e., $e_{ij} = 0$, (7) is always satisfied, as $0 \le t_i^s, t_j^s \le H$.

3.3.6 Task Non-Overlapping Constraint

For independent tasks τ_i and τ_j (i.e., $e_{ij} = 0$), when these tasks are assigned to the same processor (e.g., θ_k), we need to determine the task execution sequence, as one processor cannot handle multiple tasks simultaneously:

$$t_i^e \leq t_j^s + (2 - s_{ik} - s_{jk})H + (1 - u_{ij})H, \quad \forall i \neq j \in \mathcal{N}, \tag{8}$$

where $t_i^e = t_i^s + t_i^{\text{comp}}$ is the end time of τ_i. If τ_i and τ_j are assigned to θ_k, i.e., $s_{ik} = s_{jk} = 1$, (8) is *non-trivial* and restricts their execution sequences, e.g., if $u_{ij} = 1$, τ_i is executed before τ_j; otherwise, it holds automatically.

3.3.7 Primal Problem

Based on the objective function and the constraints mentioned above, the primal problem (PP) can be formulated as follows:

$$\mathbf{PP}: \min_{\boldsymbol{s},\boldsymbol{u},\boldsymbol{t^s},\boldsymbol{o}} \Phi = -g(o) \tag{9}$$
$$\text{s.t.} \begin{cases} (3) - (8), \\ s_{ik}, u_{ij} \in \{0,1\},\ 0 \leq o_i \leq O_i,\ 0 \leq t_i^s \leq H. \end{cases}$$

Remark 3.1 *For tractability, the execution cycle number o_i is treated as a continuous variable in* (9)*. When the problem is solved, we can round the result down. As tasks typically execute hundreds of thousands of cycles, the impact of a single cycle is negligible [16].*

Remark 3.2 *Note that nonlinear terms exist in the constraints through the products $s_{ik}o_i$ of binary variable s_{ik} and continuous variable o_i in* (4)–(8)*, and also in the objective function $g(o)$ with respect to o_i in* (2)*. Thus,* (9) *is a mixed-integer nonlinear programming problem, which is NP-hard [5].*

4 Dual Decomposition Algorithm Based on Topological Sorting

Because s_{ik} and u_{ij} are binary variables and are coupled with continuous variables (e.g., execution cycles and timing variables), (9) is a non-convex optimization problem. Directly solving it becomes computationally expensive as the number of tasks and processors increases. We therefore use a decomposition strategy: the assignment-related variables (e.g., s_{ik} and o_i) are handled in the master problem (MP), while the ordering and start-time variables (e.g., u_{ij} and t_i^s) are recovered in the scheduling problem (SP) through a dependency-aware scheduling procedure.

Compared with s_{ik}, relaxing u_{ij} from a binary variable to a continuous variable is more difficult. This is because u_{ij} represents the execution sequence between tasks τ_i and τ_j. In addition, u_{ij} is only applied for independent tasks τ_i and τ_j (i.e., $e_{ij} = 0$) when they are allocated to the same processor (i.e., $s_{ik} = s_{jk}$). If τ_i and τ_j are dependent (i.e., $e_{ij} = 1$), the execution sequence between them is fixed, no matter whether they are located on the same or different processors. On the other hand, although s_{ik} represents the task-to-core allocation, we can treat it as a continuous variable, and during the problem-solving process, if additional

constraints are taken into account to force it to select either 1 or 0, the final result of s_{ik} is still a binary value.

Based on this idea, we decompose the original problem (9) into two subproblems, where s_{ik} and u_{ij} are considered in the first and the second subproblems, respectively, as u_{ij} is influenced by s_{ik}. Each subproblem, because it contains fewer variables and constraints, is easier to solve than the single joint-design problem. Besides s_{ik} and u_{ij}, primal problem (9) also involves variables o_i (task optional cycles) and t_i^s (task start time). When u_{ij} is unknown, the value of t_i^s is unavailable. Therefore, s_{ik} and o_i can be considered simultaneously.

Note that the selections of s_{ik} and o_i are restricted by allocation constraint (3), time constraint (4), and energy constraint (6). Recalling the objective function in problem (9), the first subproblem (MP) can be formulated as (10). On the other hand, under the given s_{ik} and o_i, we determine u_{ij} and t_i^s in the second subproblem (SP), along with residual constraints, such as the deadline constraint (5), task dependency constraint (7), and task non-overlapping constraint (8). Therefore, the second subproblem can be formulated as (11).

$$\mathbf{MP}: \min_{\boldsymbol{s},\boldsymbol{o}} \ \Phi = \sum\nolimits_{i\in\mathcal{N}} \left(-p_i o_i + q_i\sqrt{o_i}\right) \tag{10}$$
$$\text{s.t.} \begin{cases} (3),\ (4),\ (6), \\ s_{ik} \in \{0,1\},\ 0 \le o_i \le O_i. \end{cases}$$

$$\mathbf{SP}: \min_{\boldsymbol{t}^{\boldsymbol{s}},\boldsymbol{u}} \ \Phi = \sum\nolimits_{i\in\mathcal{N}} \left(-p_i o_i + q_i\sqrt{o_i}\right) \tag{11}$$
$$\text{s.t.} \begin{cases} (5),\ (7),\ (8), \\ u_{ij} \in \{0,1\},\ 0 \le t_i^s \le H. \end{cases}$$

After relaxing the assignment (binary) variable s_{ik} in (10), the resulting continuous MP can be handled in an iterative dual-decomposition manner. In each iteration, the temporary results (i.e., the values of s_{ik} and o_i) are transmitted to (11). Once the SP is solved, its scheduling outcome is fed back into (10) for the next round multiplier update. This iterative design enables a controllable trade-off between solution quality and computational complexity through the prescribed convergence condition. The structure of our method is shown in Fig. 2.

4.1 Problem Transformation and Dual Decomposition for MP

To handle the nonlinear item $s_{ik}o_i$ in (4) and (6), we can introduce auxiliary variables and add additional constraints into (10), so as to transform it from MINLP to MILP [7]. However, it also increases the problem size. To circumvent this difficulty, we replace the nonlinear item $s_{ik}o_i$ with a new variable t_{ik}, representing the execution time of optional subtask τ_i on processor θ_k, which is bounded by $0 \le t_{ik} \le \frac{O_i}{\lambda_{ik} f_k}$.

With t_{ik}, the time constraint (4) and energy constraint (6) can be updated as follows:

$$\sum\nolimits_{k\in\mathcal{M}} \left(s_{ik}\frac{M_i}{\lambda_{ik} f_k} + t_{ik}\right) \le D_i, \quad \forall i \in \mathcal{N}, \tag{12}$$

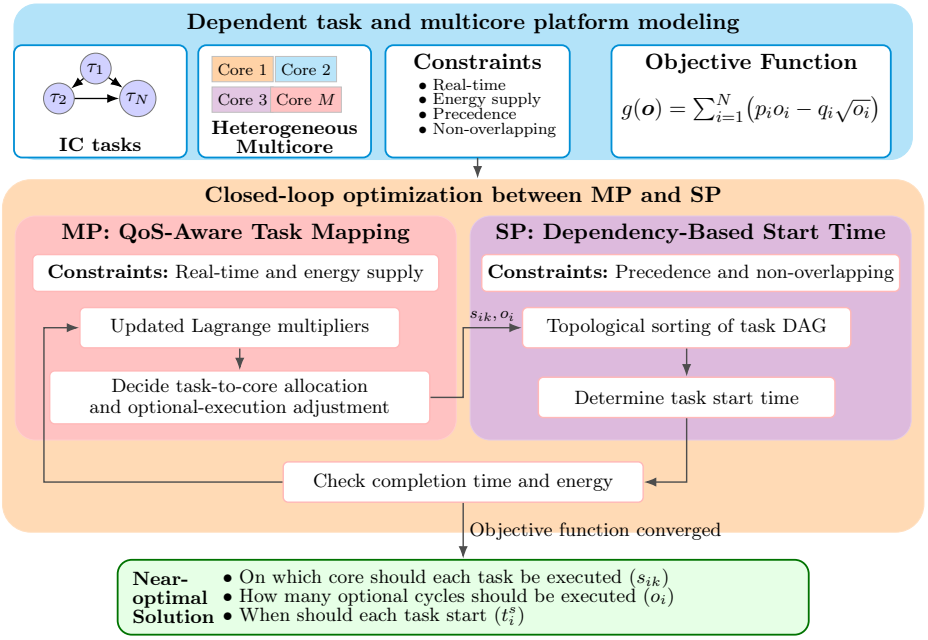


Fig. 2: Joint-design and problem decomposition architectures.

$$\sum\nolimits_{k\in\mathcal{M}}\left[\sum\nolimits_{i\in\mathcal{N}}\left(s_{ik}\frac{M_i}{\lambda_{ik}f_k}+t_{ik}\right)P_k^d+HP_k^s\right]\le E_s. \tag{13}$$

Note that we have

$$\sum\nolimits_{i\in\mathcal{N}}\left(-p_io_i+q_i\sqrt{o_i}\right)=\sum\nolimits_{i\in\mathcal{N}}\sum\nolimits_{k\in\mathcal{M}}\left(-p_it_{ik}+q_i\sqrt{t_{ik}}\right), \tag{14}$$

under the given task mapping decision t_{ik}. Therefore, (10) can then be reformulated as the following master problem with the affine time and energy constraints:

$$\begin{aligned}\textbf{MP1}: \min_{\boldsymbol{s},\boldsymbol{t}}\ & \Phi=\sum\nolimits_{i\in\mathcal{N}}\sum\nolimits_{k\in\mathcal{M}}\left(-p_it_{ik}+q_i\sqrt{t_{ik}}\right) \\ \text{s.t.}\ & \begin{cases}(3),\ (12),\ (13),\\ s_{ik}\in\{0,1\},\ 0\le t_{ik}\le\frac{O_i}{\lambda_{ik}f_k}.\end{cases}\end{aligned} \tag{15}$$

Lemma 4.1 *MP* (10) *and MP1* (15) *are equivalent.*

In the following section, we explain how to solve problem (15) efficiently. The proposed method contains three steps:

- Relax the binary variable s_{ik} to a continuous variable within the range $[0, 1]$;
- Decouple variables s_{ik} and t_{ik} through the Lagrange multipliers, yielding a task-allocation subproblem and an optional-execution subproblem;
- Update the multipliers to recover s_{ik} and t_{ik}, and transmit the results to the SP.

4.1.1 Problem Relaxation

We first relax the binary variable s_{ik} to a continuous variable within the range $[0, 1]$. Therefore, (15) can be transformed into the following problem:

$$\mathbf{MP2}: \min_{\boldsymbol{s},\boldsymbol{t}} \ \Phi = \sum\nolimits_{i\in\mathcal{N}} \sum\nolimits_{k\in\mathcal{M}} \left(-p_i t_{ik} + q_i\sqrt{t_{ik}}\right) \tag{16}$$
$$\text{s.t.} \ \begin{cases} (3),\ (12),\ (13), \\ 0 \le s_{ik} \le 1,\ 0 \le t_{ik} \le \frac{O_i}{\lambda_{ik} f_k}. \end{cases}$$

Let Φ_1^* and Φ_2^* denote the optimal objective function values of (15) and (16), respectively. Compared with (15), the feasible region of (16) is enlarged because the range of the variable s_{ik} is enlarged. Since (15) and (16) are both the minimization problems, Φ_1^* is actually an upper bound of Φ_2^*, i.e., $\Phi_1^* \ge \Phi_2^*$. Note that (16) is the continuous relaxation of (15) and can be handled efficiently by dual-based iterative methods, such as the subgradient method [4]. If the solution obtained from (16) also satisfies the binary assignment structure required by (15), then we get $\Phi_1^* = \Phi_2^*$. To promote such a solution, the constraint $s_{ik} \in \{0, 1\}$ is embedded into the decision recovery step of (16). Next, we explain how to solve problem (15) based on the solution to problem (16).

4.1.2 Dual Problem Construction and Refinement

Instead of solving problem (16) directly, we construct a Lagrangian dual relaxation. To construct the dual problem, we introduce Lagrange multipliers $\boldsymbol{\alpha}$, $\boldsymbol{\beta} \ge 0$, $\boldsymbol{\gamma} \ge 0$, and $\boldsymbol{\mu} \ge 0$ with proper dimensions to constraints (3), (12), (13), and set $0 \le t_{ik} \le \frac{O_i}{\lambda_{ik} f_k}$. Hence, the corresponding Lagrangian is

$$\begin{aligned} \mathcal{L}(\boldsymbol{s}, \boldsymbol{t}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \boldsymbol{\mu}) = & \sum\nolimits_{i\in\mathcal{N}} \sum\nolimits_{k\in\mathcal{M}} \left(-p_i t_{ik} + q_i\sqrt{t_{ik}}\right) \\ & + \sum\nolimits_{i\in\mathcal{N}} \left(\sum\nolimits_{k\in\mathcal{M}} s_{ik} - 1\right) \alpha_i \\ & + \sum\nolimits_{i\in\mathcal{N}} \sum\nolimits_{k\in\mathcal{M}} \left(t_{ik} - s_{ik}\frac{O_i}{\lambda_{ik} f_k}\right) \beta_{ik} \\ & + \sum\nolimits_{i\in\mathcal{N}} \left[\sum\nolimits_{k\in\mathcal{M}} \left(s_{ik}\frac{M_i}{\lambda_{ik} f_k} + t_{ik}\right) - D_i\right] \gamma_{ik} \\ & + \left\{\sum\nolimits_{k\in\mathcal{M}} \left[\sum\nolimits_{i\in\mathcal{N}} \left(s_{ik}\frac{M_i}{\lambda_{ik} f_k} + t_{ik}\right) P_k^d + H P_k^s\right] - E_s\right\} \mu. \end{aligned} \tag{17}$$

The dual function $\mathcal{R}(\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \boldsymbol{\mu})$ is defined as the minimum value of the Lagrangian $\mathcal{L}(\boldsymbol{s}, \boldsymbol{t}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \boldsymbol{\mu})$ over the variables $\boldsymbol{s}$ and $\boldsymbol{t}$ of the original problem [9]. Therefore, under given Lagrange multipliers $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, $\boldsymbol{\gamma}$ and $\boldsymbol{\mu}$, the corresponding dual function is

$$\mathcal{R}(\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \boldsymbol{\mu}) = \min_{\boldsymbol{s},\boldsymbol{t}} \mathcal{L}(\boldsymbol{s}, \boldsymbol{t}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \boldsymbol{\mu}). \tag{18}$$

Based on the dual function $\mathcal{R}(\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \boldsymbol{\mu})$, the Dual Problem (DP) associated with problem (16) can be formulated as

$$\mathbf{DP}: \max_{\boldsymbol{\alpha},\boldsymbol{\beta},\boldsymbol{\gamma},\mu} \ \mathcal{R}(\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \mu) \tag{19}$$

$$\text{s.t. } \boldsymbol{\beta} \succeq 0,\ \boldsymbol{\gamma} \succeq 0,\ \mu \geq 0.$$

Note that (3), (12), (13) are linear functions regarding variables s_{ik} and t_{ik}, and $\sum_{i\in\mathcal{N}}\sum_{k\in\mathcal{M}}(-p_i t_{ik} + q_i\sqrt{t_{ik}})$ is a convex function [9]. Hence, Slater's condition is satisfied [4], and we have the following lemma.

Lemma 4.2 *Strong duality exists between the primal problem* (16) *and its dual problem* (19).

Due to the strong duality, the primal problem and its dual problem have the same optimal objective function value [4], i.e., solving problems (16) and (19) are equivalent.

Note that DP (19) has separable objective terms. To solve the problem of maximizing $\mathcal{R}(\boldsymbol{\alpha},\boldsymbol{\beta},\boldsymbol{\gamma},\boldsymbol{\mu})$, we first solve the problem of minimizing $\mathcal{L}(\boldsymbol{s},\boldsymbol{t},\boldsymbol{\alpha},\boldsymbol{\beta},\boldsymbol{\gamma},\boldsymbol{\mu})$ over $\boldsymbol{s}$ and $\boldsymbol{t}$. As variables $\boldsymbol{s}$ and $\boldsymbol{t}$ are coupled with each linearly in $\mathcal{L}(\boldsymbol{s},\boldsymbol{t},\boldsymbol{\alpha},\boldsymbol{\beta},\boldsymbol{\gamma},\boldsymbol{\mu})$, the Lagrangian (17) can be decomposed as

$$\mathcal{L}(\boldsymbol{s},\boldsymbol{t},\boldsymbol{\alpha},\boldsymbol{\beta},\boldsymbol{\gamma},\boldsymbol{\mu}) = \mathcal{L}_1(\boldsymbol{s},\boldsymbol{\alpha},\boldsymbol{\beta},\boldsymbol{\gamma},\boldsymbol{\mu}) + \mathcal{L}_2(\boldsymbol{t},\boldsymbol{\beta},\boldsymbol{\gamma},\boldsymbol{\mu}) + \mathcal{L}_3(\boldsymbol{\gamma},\boldsymbol{\mu}), \tag{20}$$

where

$$\mathcal{L}_1(\boldsymbol{s},\boldsymbol{\alpha},\boldsymbol{\beta},\boldsymbol{\gamma},\boldsymbol{\mu}) = \sum\nolimits_{i\in\mathcal{N}}\sum\nolimits_{k\in\mathcal{M}}\left[\alpha_i - \frac{O_i\beta_{ik} - M_i\left(\gamma_{ik} + P_k^d\mu\right)}{\lambda_{ik}f_k}\right]s_{ik} - \alpha_i, \tag{21}$$

$$\mathcal{L}_2(\boldsymbol{t},\boldsymbol{\beta},\boldsymbol{\gamma},\boldsymbol{\mu}) = \sum\nolimits_{i\in\mathcal{N}}\sum\nolimits_{k\in\mathcal{M}}\left[\left(\beta_{ik} + \gamma_{ik} + P_k^d\mu - p_i\right)t_{ik} + q_i\sqrt{t_{ik}}\right], \tag{22}$$

$$\mathcal{L}_3(\boldsymbol{\gamma},\boldsymbol{\mu}) = \sum\nolimits_{k\in\mathcal{M}}\left[\sum\nolimits_{i\in\mathcal{N}}(-D_i\gamma_{ik}) - E_s\mu + HP_k^s\mu\right]. \tag{23}$$

Note that $\mathcal{L}_1(\boldsymbol{s},\boldsymbol{\alpha},\boldsymbol{\beta},\boldsymbol{\gamma},\boldsymbol{\mu})$ is linear with respect to $\boldsymbol{s}$, $\mathcal{L}_2(\boldsymbol{t},\boldsymbol{\beta},\boldsymbol{\gamma},\boldsymbol{\mu})$ is separable with respect to $\boldsymbol{t}$, and $\mathcal{L}_3(\boldsymbol{\gamma},\boldsymbol{\mu})$ is fixed under the given multipliers $\boldsymbol{\gamma}$ and $\boldsymbol{\mu}$. Therefore, the dual function (18) in DP (19) can be written as

$$\mathcal{R}(\boldsymbol{\alpha},\boldsymbol{\beta},\boldsymbol{\gamma},\boldsymbol{\mu}) = \min_{\boldsymbol{s}}\mathcal{L}_1(\boldsymbol{s},\boldsymbol{\alpha},\boldsymbol{\beta},\boldsymbol{\gamma},\boldsymbol{\mu}) + \min_{\boldsymbol{t}}\mathcal{L}_2(\boldsymbol{t},\boldsymbol{\beta},\boldsymbol{\gamma},\boldsymbol{\mu}) + \mathcal{L}_3(\boldsymbol{\gamma},\boldsymbol{\mu}), \tag{24}$$

which means the two subproblems $\min_{\boldsymbol{s}}\mathcal{L}_1(\boldsymbol{s},\boldsymbol{\alpha},\boldsymbol{\beta},\boldsymbol{\gamma},\boldsymbol{\mu})$ and $\min_{\boldsymbol{t}}\mathcal{L}_2(\boldsymbol{t},\boldsymbol{\beta},\boldsymbol{\gamma},\boldsymbol{\mu})$ in (24) can be solved separately, and the details are as follows.

– *Minimizing $\mathcal{L}_1$ through $\boldsymbol{s}$:* Define $\mathcal{H}_{ik}(\boldsymbol{\beta},\boldsymbol{\gamma},\boldsymbol{\mu}) = \frac{O_i\beta_{ik} - M_i(\gamma_{ik} + P_k^d\mu)}{\lambda_{ik}f_k}$ and substitute $\mathcal{H}_{ik}(\boldsymbol{\beta},\boldsymbol{\gamma},\boldsymbol{\mu})$ into $\mathcal{L}_1(\boldsymbol{s},\boldsymbol{\alpha},\boldsymbol{\beta},\boldsymbol{\gamma},\boldsymbol{\mu})$, we can obtain:

$$\min_{\boldsymbol{s}}\mathcal{L}_1(\boldsymbol{s},\boldsymbol{\alpha},\boldsymbol{\beta},\boldsymbol{\gamma},\boldsymbol{\mu}) = \min_{\boldsymbol{s}}\sum\nolimits_{i\in\mathcal{N}}\sum\nolimits_{k\in\mathcal{M}}\left[\alpha_i - \mathcal{H}_{ik}(\boldsymbol{\beta},\boldsymbol{\gamma},\boldsymbol{\mu})\right]s_{ik} - \alpha_i. \tag{25}$$

Since $[\alpha_i - \mathcal{H}_{ik}(\boldsymbol{\beta},\boldsymbol{\gamma},\boldsymbol{\mu})]s_{ik}$ is a linear function regarding the variable s_{ik}, and the dimension of $\boldsymbol{s}$ is $N \times M$, (25) can be decomposed into $N \times M$ independent problems and each with the form: $\min_{s_{ik}}[\alpha_i - \mathcal{H}_{ik}(\boldsymbol{\beta},\boldsymbol{\gamma},\boldsymbol{\mu})]s_{ik}$. Comparing α_i with $\mathcal{H}_{ik}(\boldsymbol{\beta},\boldsymbol{\gamma},\boldsymbol{\mu})$, we can select a proper value of s_{ik} to achieve the minimization. Therefore, the task allocation decision s_{ik} can be determined as follows:

$$s_{ik} = \begin{cases} 1 & \text{if } \alpha_i < \mathcal{H}_{ik}(\boldsymbol{\beta},\boldsymbol{\gamma},\boldsymbol{\mu}) \\ [0,1] & \text{if } \alpha_i = \mathcal{H}_{ik}(\boldsymbol{\beta},\boldsymbol{\gamma},\boldsymbol{\mu}) \\ 0 & \text{if } \alpha_i > \mathcal{H}_{ik}(\boldsymbol{\beta},\boldsymbol{\gamma},\boldsymbol{\mu}). \end{cases} \tag{26}$$

From (26), we can obtain

$$\min_{s_{ik}}[\alpha_i - \mathcal{H}_{ik}(\boldsymbol{\beta},\boldsymbol{\gamma},\boldsymbol{\mu})]s_{ik} = [\alpha_i - \mathcal{H}_{ik}(\boldsymbol{\beta},\boldsymbol{\gamma},\boldsymbol{\mu})]^-, \tag{27}$$

where $[x]^- \triangleq \min\{x, 0\}$. With (27), subproblem (25) can be decomposed into N independent minimization problems, and each problem has an objective function: $\sum_{k\in\mathcal{M}}[\alpha_i - \mathcal{H}_{ik}(\boldsymbol{\beta},\boldsymbol{\gamma},\boldsymbol{\mu})]^-$. To transform s_{ik} from a continuous value to a binary value, we introduce the following lemma.

Lemma 4.3 *Since each task τ_i is executed on only one processor, to minimize the function $\sum_{k\in\mathcal{M}}[\alpha_i - \mathcal{H}_{ik}(\boldsymbol{\beta},\boldsymbol{\gamma},\boldsymbol{\mu})]^-$, the optimal task allocation decision is determined by:*

$$s_{ik}^* = \begin{cases} 1 & \text{if } k = \arg\min\limits_k\{\alpha_i - \mathcal{H}_{ik}(\boldsymbol{\beta},\boldsymbol{\gamma},\boldsymbol{\mu})\} \\ 0 & \text{else} \end{cases} \tag{28}$$

If $\alpha_i - \mathcal{H}_{ik}(\boldsymbol{\beta},\boldsymbol{\gamma},\boldsymbol{\mu})$ has multiple same minimum values, we randomly select one from these minimum values, as each task can be assigned to only one processor.

– *Minimizing $\mathcal{L}_2$ through $\boldsymbol{t}$:* Under the given multipliers $\boldsymbol{\beta}$, $\boldsymbol{\gamma}$, and $\boldsymbol{\mu}$, to find an optimal task execution time $\boldsymbol{t}^*$ to minimize $\mathcal{L}_2(\boldsymbol{t},\boldsymbol{\beta},\boldsymbol{\gamma},\boldsymbol{\mu})$, we can solve the following differential equations: $\partial\mathcal{L}_2(\boldsymbol{t},\boldsymbol{\beta},\boldsymbol{\gamma},\boldsymbol{\mu})/\partial t_{ik} = 0 \Rightarrow \boldsymbol{t}$. Then, based on $\boldsymbol{t}$ and $t_{ik} = s_{ik}o_i$ (task execution time t_{ik} is influenced by task allocation decision s_{ik}), the optimal execution time of optional task t_{ik}^* can be determined as follows:

$$t_{ik}^* = \begin{cases} t_{ik} & \text{if } s_{ik}^* = 1 \\ 0 & \text{else} \end{cases} \tag{29}$$

Thus, we have $\min_{\boldsymbol{t}} \mathcal{L}_2(\boldsymbol{t},\boldsymbol{\beta},\boldsymbol{\gamma},\boldsymbol{\mu}) = \mathcal{L}_2(\boldsymbol{t}^*,\boldsymbol{\beta},\boldsymbol{\gamma},\boldsymbol{\mu})$, where $\boldsymbol{t}^*$ and $\mathcal{L}_2(\boldsymbol{t}^*,\boldsymbol{\beta},\boldsymbol{\gamma},\boldsymbol{\mu})$ are actually the functions with respect to multipliers $\boldsymbol{\beta}$, $\boldsymbol{\gamma}$, and $\boldsymbol{\mu}$. Substituting (28) and (29) into the dual function (18), the refined dual function is

$$\mathcal{R}(\boldsymbol{\alpha},\boldsymbol{\beta},\boldsymbol{\gamma},\boldsymbol{\mu}) = \mathcal{L}_1(\boldsymbol{s}^*,\boldsymbol{\alpha},\boldsymbol{\beta},\boldsymbol{\gamma},\boldsymbol{\mu}) + \mathcal{L}_2(\boldsymbol{t}^*,\boldsymbol{\beta},\boldsymbol{\gamma},\boldsymbol{\mu}) + \mathcal{L}_3(\boldsymbol{\gamma},\boldsymbol{\mu}). \tag{30}$$

Finally, based on the above two steps, we can replace the objective function in the DP with the refined dual function (30), and solve the refined dual problem through the Lagrangian multipliers $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, $\boldsymbol{\gamma}$, and $\boldsymbol{\mu}$.

4.1.3 Lagrangian Multipliers Update

To solve the refined dual problem, an important step is to find the subgradient of the refined dual function (30). Let $\nabla\alpha_i$, $\nabla\beta_{ik}$, $\nabla\gamma_{ik}$, and $\nabla\mu$ denote the subgradients of refined dual function $\mathcal{R}(\boldsymbol{\alpha},\boldsymbol{\beta},\boldsymbol{\gamma},\boldsymbol{\mu})$ with respect to the variables α_i, β_{ik}, γ_{ik}, and μ, respectively. Based on the definition of subgradient [4], we have

$$\begin{aligned}
\nabla\alpha_i &= \sum\nolimits_{k\in\mathcal{M}} s_{ik}^* - 1, \\
\nabla\beta_{ik} &= t_{ik}^* - s_{ik}^*\frac{O_i}{\lambda_{ik}f_k}, \\
\nabla\gamma_{ik} &= \sum\nolimits_{i\in\mathcal{N}}\left[\sum\nolimits_{k\in\mathcal{M}}\left(s_{ik}^*\frac{M_i}{\lambda_{ik}f_k} + t_{ik}^*\right) - D_i\right], \\
\nabla\mu &= \sum\nolimits_{k\in\mathcal{M}}\left[\sum\nolimits_{i\in\mathcal{N}}\left(s_{ik}^*\frac{M_i}{\lambda_{ik}f_k} + t_{ik}^*\right)P_k^d + HP_k^s\right] - E_s,
\end{aligned} \tag{31}$$

Algorithm 1 Dual Decomposition-Based Solution for MP2

Input: Parameters in MP2;
Output: $\boldsymbol{s}^*$ and $\boldsymbol{t}^*$;
1: Initialize Lagrangian multipliers $(\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \mu)$, set $n = 0$, $\delta_n = \delta_0$, and ϵ;
2: **while** $n < N_{\max}$ **do**
3: **for** $i = 1$ **to** N **do**
4: **for** $k = 1$ **to** M **do**
5: Update $s_{ik}^{*(n)}$ and $t_{ik}^{*(n)}$ through (28) and (29);
6: **end for**
7: **end for**
8: $\boldsymbol{s}^{*(n)} = [s_{ik}^{*(n)}]_{N\times M}$ and $\boldsymbol{t}^{*(n)} = [t_{ik}^{*(n)}]_{N\times M}$;
9: Calculate $g(\boldsymbol{t}^{*(n)})$ by $\sum_{i\in\mathcal{N}}\sum_{k\in\mathcal{M}}(-p_i t_{ik}^{*(n)} + q_i\sqrt{t_{ik}^{*(n)}})$;
10: Calculate $\mathcal{L}(\boldsymbol{s}^{*(n)}, \boldsymbol{t}^{*(n)}, \boldsymbol{\alpha}^{(n)}, \boldsymbol{\beta}^{(n)}, \boldsymbol{\gamma}^{(n)}, \mu^{(n)})$ through (17);
11: **if** $|\mathcal{L}(\boldsymbol{s}^{*(n)}, \boldsymbol{t}^{*(n)}, \boldsymbol{\alpha}^{(n)}, \boldsymbol{\beta}^{(n)}, \boldsymbol{\gamma}^{(n)}, \mu^{(n)}) - (-g(\boldsymbol{t}^{*(n)}))| < \epsilon$ **then**
12: $\boldsymbol{s}^* \leftarrow \boldsymbol{s}^{*(n)}$ and $\boldsymbol{t}^* \leftarrow \boldsymbol{t}^{*(n)}$;
13: **return** ;
14: **end if**
15: $\boldsymbol{\lambda}^{(n+1)} \leftarrow [\boldsymbol{\lambda}^{(n)} + \delta_n \nabla\boldsymbol{\lambda}^{(n)}]^+$ by (31), where $\lambda = \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \mu$;
16: Set $n \leftarrow n + 1$, and update step size $\delta_{n+1} \leftarrow \delta_n/\sqrt{n}$;
17: **end while**

where s_{ik}^* and t_{ik}^* are corresponding optimal solutions of task allocation and execution time under the given multipliers $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, $\boldsymbol{\gamma}$, and $\boldsymbol{\mu}$, respectively.

Algorithm 1 summarizes the proposed MP2 solution. In each iteration n, under the given Lagrangian multipliers $\alpha_i^{(n)}$, $\beta_{ik}^{(n)}$, $\gamma_{ik}^{(n)}$, and $\mu^{(n)}$, the decisions regarding task allocation $s_{ik}^{*(n)}$ and execution time $t_{ik}^{*(n)}$ are updated through (28) and (29) (Line 5). Then, based on $s_{ik}^{*(n)}$ and $t_{ik}^{*(n)}$, the Lagrangian multipliers are updated by (31) (Line 15), where $[x]^+ \triangleq \max\{0, x\}$, and are used in the next iteration $n + 1$.

4.1.4 Complexity Analysis

In the n^{th} iteration, the primal variables $[s_{ik}^{*(n)}]_{N\times M}$ and $[t_{ik}^{*(n)}]_{N\times M}$ (task allocation and scheduling) are updated, leading to a computational cost of $O(NM)$. In addition, the calculation of the dual function $g(\boldsymbol{t}^{*(n)})$ and the Lagrangian $\mathcal{L}(\boldsymbol{s}^{*(n)}, \boldsymbol{t}^{*(n)}, \boldsymbol{\alpha}^{(n)}, \boldsymbol{\beta}^{(n)}, \boldsymbol{\gamma}^{(n)}, \mu^{(n)})$ requires the computation complexity $O(NM)$. On this basis, updating the dual variables α_i, β_{ik}, γ_{ik}, and μ via the sub-gradient method (31) takes $O(NM + N)$. Therefore, the overall complexity per iteration is dominated by $O(NM)$, and in the worst case, the complexity is $O(N_{\max}NM)$, where $N_{\max}$ is the maximum iteration number. Since $M \ll N$, i.e., the number of processors is much smaller than the number of tasks, the complexity can be simplified as $O(N_{\max}N)$, which is approximately linear with the task number N under the given $N_{\max}$. Therefore, the proposed method can solve problem (10) in polynomial time.

4.2 Determine Task Execution Sequence for SP

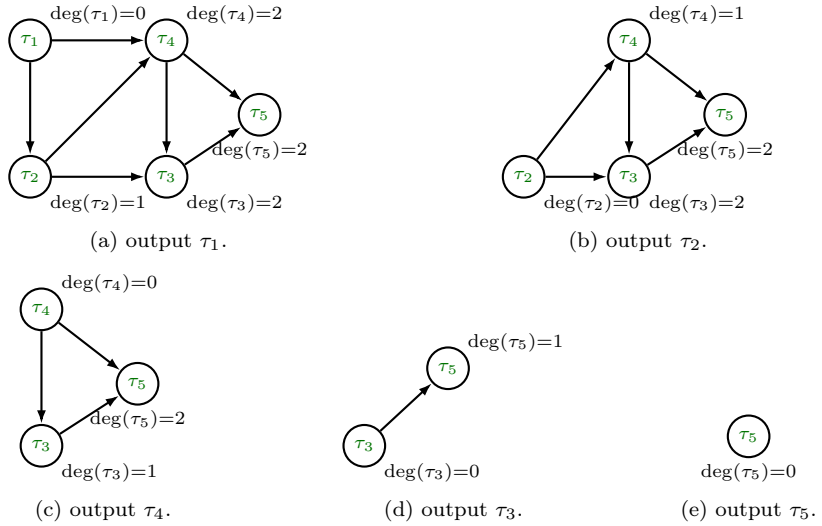
By using the solution to MP, the decisions regarding task-to-processor allocation s_{ik} and task execution time t_{ik} are known. To solve SP, we need to determine

the task execution sequence u_{ij}, and on this basis, we can calculate the start time for each task t_i^s. To optimize u_{ij} and t_i^s under multiple constraints, such as deadline (5), task dependency (7), and task non-overlapping (8), we propose a task sorting method based on task dependency, and the details are as follows.

Since the tasks are dependent, a task τ_j can start its execution only when all its predecessors are completed. Let $pred(\tau_j) = \{\tau_i \in \mathcal{T} \mid (\tau_i, \tau_j) \in \mathcal{E}\}$ denote the set of τ_j's predecessors, where $\mathcal{T}$ and $\mathcal{E}$ are the vertex and the edge of task DAG $G = (\mathcal{T}, \mathcal{E})$, respectively, and let t_j^{pred} denote the latest time when all predecessors of τ_j finish their executions. For each predecessor $\tau_i \in pred(\tau_j)$, since its end time can be given by $t_i^e = t_i^s + t_i^{\text{comp}}$, we have $t_j^{\text{pred}} = \max\{t_i^s + t_i^{\text{comp}} \mid \tau_i \in pred(\tau_j)\}$. On the other hand, if task τ_j is assigned to processor p_k, the earliest possible start time of τ_j, i.e., t_j^{earl}, is determined by the end time of τ_j's previous task on the same processor, i.e., avail_j^k. Therefore, we have $t_j^{\text{earl}} = \max\left\{t_j^{\text{pred}}, \text{avail}_j^k\right\}$.

Note that the task execution sequence u_{ij} is restricted by the task non-overlapping constraint (8). Instead of dealing with u_{ij} directly, we determine task execution order through two auxiliary variables t_j^{pred} and avail_j^k. t_j^{pred} records the latest completion time of the predecessors of task τ_j, and avail_j^k records the earliest available time of processor θ_k for task τ_j. In this way, task dependency is handled by t_j^{pred}, while task non-overlapping is handled by avail_j^k. The details of t_j^{pred} and avail_j^k calculations are as follows.

- In the initialization step, we compute the *in-degree* for each task τ_i, according to the task DAG $G(\mathcal{T}, \mathcal{E})$ (Line 1). Let $\deg(\tau_i) = |\{pred(\tau_i) \mid (\tau_i, \tau_j) \in \mathcal{E}\}|$ denote the in-degree of τ_i, representing the number of uncompleted predecessors of τ_i. For entry tasks, the in-degree is zero because they have no predecessors. All tasks with zero in-degree are inserted into the *Ready Queue*, which stores the tasks whose dependency constraints have been satisfied and are ready to be scheduled. Their start times are then determined according to t_j^{pred} and avail_j^k (Line 2).
- The tasks in the Ready Queue are handled in sequence. Suppose that task τ_i is selected from the Ready Queue. Based on the decision s_{ik}, the processor responsible for executing τ_i is known, e.g., processor θ_k with $s_{ik} = 1$. If τ_i is an entry task, its start time is set to $t_i^s = 0$. Taking task execution time t_{ik} into account, we can calculate its end time t_i^e (Line 10). Note that tasks sorted by Ready Queue satisfy the task dependency constraint (7).
- When calculating the end time t_i^e for each task τ_i, we can compare it with the scheduling horizon H to check whether the deadline constraint (5) is satisfied or not. If t_i^e violates the deadline constraint, we adjust the values of t_{ik} and s_{ik} via the method in Section 4.3.
- Once task τ_i is completed, we remove τ_i from the Ready Queue and decrease the in-degree of each successor of τ_i by one (Line 12). If a successor task τ_j reaches zero in-degree, it enters the Ready Queue. Assume that τ_j is also assigned to θ_k, i.e., $s_{jk} = 1$. To determine the start time of τ_j, we calculate t_j^{pred} (Line 6) and avail_j^k (Line 9). If τ_i has the latest finish time among τ_j's predecessors, then we have $t_j^{\text{pred}} = t_i^e$. If no other task is scheduled between τ_i and τ_j on θ_k, then we get $\text{avail}_j^k = t_i^e$. Therefore, the earliest possible start time of τ_j is determined by $t_j^{\text{earl}} = \max\left\{t_j^{\text{pred}}, \text{avail}_j^k\right\}$.



(a) output τ_1. (b) output τ_2.

(c) output τ_4. (d) output τ_3. (e) output τ_5.

Fig. 3: Task sorting according to task dependency.

Table 3: Scheduling of Dependent and Real-Time Tasks on Multi-Core Platform

Task	τ_1	τ_2	τ_4	τ_3	τ_5
Processor	P_1	P_3	P_2	P_1	P_3
t_i^{comp}	1.498	1.265	0.566	0.924	0.274
Predecessor	None	τ_1	τ_1, τ_2	τ_2, τ_4	τ_3, τ_4
Task Dependency Check	No Predecessors, P_1 available at 0	Wait for τ_1 (t_1^e=1.498), P_3 available at 0	Wait for τ_1 and τ_2 (max{t_1^e,t_2^e} =2.763), P_2 available at 0	Wait for τ_2 and τ_4 (max{t_2^e,t_4^e} =3.329), P_1 available at 1.498	Wait for τ_3 and τ_4 (max{t_3^e,t_4^e} =4.253), P_3 available at 2.763
t_i^s	0	1.498	2.763	3.329	4.253
t_i^e	1.498	2.763	3.329	4.253	4.527
avail_i^k	1.498	2.763	3.329	4.253	4.527

– By applying the above method for each task in the set $\mathcal{T}$, we can determine the task scheduling on the processors. Note that this method can meet the constraints regarding task dependency (7) and task non-overlapping (8) simultaneously.

Algorithm 2 summarizes the implementation details of the scheduling algorithm. We also use an example with five tasks $\tau_1 \sim \tau_5$ to illustrate the proposed dependency-aware task scheduling method, as shown in Fig. 3 and Table 3.

– Since τ_1 is an entry task, and τ_1 is assigned to processor P_1, we have $\deg(\tau_1) = 0$ for τ_1. τ_1 executes at $t = 0$ ms and completes at $t_1^e = 1.498$ ms, due to P_1 being available at $t = 0$ ms. Upon τ_1 finishes its execution, the values of $\deg(\tau_2)$ and $\deg(\tau_4)$ are reduced by one, as τ_2 and τ_4 are the successors of τ_1, making $\deg(\tau_2) = 0$ and $\deg(\tau_4) = 1$.

Algorithm 2 Dependency-Aware Task Scheduling

Input: Task allocation s_{ik}, task execution time t_{ik}, and task DAG $G(\mathcal{T}, \mathcal{E})$
Output: Start time t_i^s, end time t_i^e

Initialize $\text{avail}_i^k = 0$ $(\forall k \in \mathcal{M})$, $\deg(\tau_i)$ $(\forall i \in \mathcal{N})$
Initialize *Ready Queue*
while Ready Queue $\neq \emptyset$ **do**
    $\tau_i \leftarrow$ Ready Queue
    Assign task τ_i to processor θ_k through s_{ik}
    $t_i^{\text{pred}} = \max\left\{t_h^s + t_h^{\text{comp}} \mid \tau_h \in pred(\tau_i)\right\}$
    $t_i^s \leftarrow \max\left\{t_i^{\text{pred}}, \text{avail}_i^k\right\}$
    $t_i^{\text{comp}} \leftarrow \sum_{k\in\mathcal{M}} \left(s_{ik} \frac{M_i + o_i}{\lambda_{ik} f_k}\right)$
    $\text{avail}_i^k \leftarrow t_i^s + t_i^{\text{comp}}$
    $t_i^e \leftarrow t_i^s + t_i^{\text{comp}}$
    for each successor $\tau_j \in \text{succ}(\tau_i)$ **do**
        $\deg(\tau_j) \leftarrow \deg(\tau_j) - 1$
        if $\deg(\tau_j) = 0$ **then**
            τ_j enters Ready Queue
        end if
    end for
end while

– Since $\deg(\tau_2) = 0$, τ_2 enters the Ready Queue to determine its start time t_2^s, which is detailed in the second line of "Dependency Check" in Table 3. Note that τ_2 is assigned to P_3, and its start time is restricted by the end time of τ_1. τ_2 starts at $t_2^s = 1.498\ ms$ and ends at $t_2^e = 2.763\ ms$.
– Subsequently, with the completion of τ_2, we get $\deg(\tau_4) = 0$, and τ_4 is listed in the Ready Queue. Although P_2 is available at time $t = 0\ ms$, τ_4 must wait for its predecessors τ_1 and τ_2. Therefore, the start time and the end time of τ_4 are $t_4^s = \max\{\text{avail}_4^2, t_2^e, t_1^e\} = 2.763\,\text{ms}$ and $t_4^e = 3.329\ ms$, respectively.
– The task non-overlapping constraint is applied for τ_1 and τ_3, as they are independent and assigned to the same processor P_1. After completing τ_1, P_1 is available after $\text{avail}_1^1 = 1.498\ ms$. τ_3 starts its execution only after $t = 3.329\ ms$, due to the execution of τ_1 take $1.498\ ms$, forcing the start time to $t_3^s = \max\left\{t_4^e, \text{avail}_1^1\right\} = \max\{3.329, 1.498\} = 3.329\,\text{ms}$ and the end time to $t_3^e = 4.253\ ms$.
– Finally, τ_5 is scheduled on P_3 after its predecessors τ_3 and τ_4 are completed; driven by the latest completion of τ_3 at $4.253\ ms$ (which exceeds the earliest availability time of P_3 at $2.763\ ms$), τ_5 starts at $t_5^s = 4.253\ ms$ and ends at $t_5^e = 4.527\ ms$, ensuring a valid, conflict-free task mapping.

4.3 Closed-Loop Joint Optimization Framework

Note that multiple constraints, such as task deadline (5), task dependency (7), and task non-overlapping (8), are considered in the SP (11) rather than the MP (10). The optimal objective function value achieved by MP is a lower bound of the original problem (9), leading to the corresponding task end time $t_i^e = t_i^s + t_i^{comp}$, which may violate the scheduling horizon H, i.e., the SP may be infeasible under the given MP solution. The infeasibility of SP is mainly caused by the time

constraint (5), as constraints (7) and (8) can be satisfied by the proposed task sorting method in Section 4.2. To enhance the feasibility of SP, in this section, we propose a closed-loop feedback framework to adjust the MP solution based on the SP result.

To guarantee a feasible solution of problem (9), the SP not only determines u_{ij} and t_i^s, but also acts as a feasibility check by calculating the actual finish time t_i^e and verifying time constraint (5). If the global deadline is violated, i.e., $t_i^e > H$, the residual time $t_i^e - H$ is used as feedback to indicate how far the current MP solution is from scheduling feasibility. The closed-loop method then updates the penalties and guides the MP to select faster processors or reduce optional execution in the next iteration.

As shown in Line 15 of Algorithm 1, in each MP iteration, the Lagrange multipliers $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, $\boldsymbol{\gamma}$, and μ act as sensitivity weights for the corresponding constraints. According to (31), we can see that α_i enforces task-to-processor assignment, β_{ik} controls the optional-execution bound, γ_{ik} penalizes timing violations, and μ regulates total energy consumption against the system budget. These multipliers are then embedded in the next MP update of s_{ik}^* and t_{ik}^*, forming the feedback loop between scheduling feasibility and optional-execution allocation.

The proposed closed-loop optimization framework consists of the following three steps: 1) primal solution and feasibility check; 2) multiplier update; and 3) termination of MP and SP iterations, and their details are as follows.

4.3.1 Primal Solution and Feasibility Check

In the n^{th} iteration, under the given multipliers $\alpha_i^{(n)}$, $\beta_{ik}^{(n)}$, $\gamma_{ik}^{(n)}$, and $\mu^{(n)}$, the MP provides the task assignment $s_{ik}^{*(n)}$ according to (28) and the optional time $t_{ik}^{*(n)}$ according to (29). Then, we substitute $s_{ik}^{*(n)}$ and $t_{ik}^{*(n)}$ into the SP and solve it using the dependency-aware scheduling method in Algorithm 2. The SP returns the actual task end time t_i^e, where the start time t_i^s is determined by both the latest completion time t_i^{pred} of the predecessors and the earliest availability time avail_i^k of the assigned processor θ_k.

4.3.2 Multiplier Update

In the $(n+1)^{th}$ iteration, we update the Lagrange multipliers $\alpha_i^{(n+1)}$, $\beta_{ik}^{(n+1)}$, $\gamma_{ik}^{(n+1)}$, and $\mu^{(n+1)}$ using the MP and SP solutions obtained in the n^{th} iteration, and the details are as follows:

– Allocation and Energy Multipliers (α_i and μ), which correspond to constraints (3) and (6), respectively. Since the MP solution $s_{ik}^{*(n)}$ is derived through (28), task allocation constraint (3) is satisfied in each iteration. In addition, the energy constraint (6) is handled in the MP. Let $[\cdot]^+ \triangleq \max\{x, 0\}$. Based on the gradients $\nabla\alpha_i^{(n)}$ and $\nabla\mu^{(n)}$ given by (31), the updates of $\alpha_i^{(n)}$ and $\mu^{(n)}$ are

$$\alpha_i^{(n+1)} = \alpha_i^{(n)} + \delta_n \nabla\alpha_i^{(n)}, \tag{32}$$

$$\mu^{(n+1)} = \left[\mu^{(n)} + \delta_n \nabla\mu^{(n)}\right]^+. \tag{33}$$

– Deadline Multiplier (γ_{ik}), which corresponds to time constraint (4). In the MP solution, γ_{ik} mainly penalizes the violation of the relative deadline $t_i^{comp} \leq D_i$ for task τ_i. Although constraint (4) only limits the task execution time, the task end time is given by $t_i^e = t_i^s + t_i^{comp}$, i.e., adjusting t_i^{comp} can also help us to reduce the global deadline violation. Therefore, we define a timing gradient $\nabla\gamma_{ik}^{(n)}$ that considers the violation between the relative deadline constraint (4) and the global deadline constraint (5), which can be defined as follows:

$$\nabla\gamma_{ik}^{(n)} = \max\Big\{ \underbrace{t_i^{comp(n)} - D_i}_{\text{relative deadline}}, \quad \underbrace{t_i^{e(n)} - H}_{\text{global deadline}} \Big\}, \tag{34}$$

where $t_i^{comp(n)}$ is given by substituting $s_{ik}^{*(n)}$ and $o_i^{*(n)}$ into $\sum_{k\in\mathcal{M}} \left(s_{ik}\frac{M_i+o_i}{\lambda_{ik}f_k}\right)$, and $t_i^{e(n)}$ is given by the SP solution. Hence, the update of $\gamma_{ik}^{(n)}$ is given by

$$\gamma_{ik}^{(n+1)} = \left[\gamma_{ik}^{(n)} + \delta_n \cdot \nabla\gamma_{ik}^{(n)}\right]^+. \tag{35}$$

According to (35), γ_{ik} serves as a sensitivity weight that amplifies the importance of task execution time t_i^{comp}. If a task τ_i satisfies its relative deadline D_i in constraint (4) but violates the global deadline H in constraint (5), then $t_i^{e(n)} - H$ is positive, and further, $\nabla\gamma_{ik}^{(n)}$ becomes positive and increases the timing penalty. This raises the execution-time cost in the MP, forcing the task to reduce its optional part o_i or move to a faster core in the next iteration.

– Optional Execution Multiplier (β_{ik}), which corresponds to the constraint $0 \leq t_{ik} \leq \frac{O_i}{\lambda_{ik}f_k}$, i.e., bounds the optional cycles. Similar to γ_{ik}, by adjusting the value of β_{ik}, we can change task execution time t_{ik}, and further, change the task end time t_i^e. Based on this idea, the update of $\beta_{ik}^{(n)}$ is set to

$$\beta_{ik}^{(n+1)} = \left[\beta_{ik}^{(n)} + \delta_n \left(t_{ik}^{(n)} - s_{ik}^{(n)}\frac{O_i}{\lambda_{ik}f_k} + \rho_1\nabla\gamma_{ik}^{(n)}\right)\right]^+, \tag{36}$$

where ρ_1 is a positive weight. On the one hand, $t_{ik}^{(n)} - s_{ik}^{(n)}\frac{O_i}{\lambda_{ik}f_k}$ represents the residual of the optional-execution upper bound, as $t_{ik} \leq s_{ik}\frac{O_i}{\lambda_{ik}f_k}$. If this residual is positive, the optional execution time exceeds its upper bound. β_{ik} is increased to penalize this violation in the next MP iteration. On the other hand, $\rho_1\nabla\gamma_{ik}^{(n)}$ transfers the global timing violation to the optional-execution multiplier. If the global deadline (5) is violated, the positive gradient $\nabla\gamma_{ik}^{(n)}$ increases β_{ik} and raises the cost of optional execution. In the next MP iteration, this higher cost reduces task execution time t_{ik}, shortens task end time t_i^e, and makes time constraint (5) easier to satisfy.

Remark 4.1 *(Enhanced Energy Efficiency): In constraint* (13)*, the duration of static power P_k^s used to calculate E^{comp} is the scheduling horizon H, which means that processors may remain idle after their assigned tasks finish. However, once the assigned tasks are completed, processors can switch to a low-power state through the Dynamic Power Management (DPM) technology [7] (e.g., from the idle state to the sleep state), thereby further reducing energy consumption. By using the actual*

task end times t_i^e from the SP (11)*, we can refine the energy consumption E^{comp}. Let $T_{total} = \max_{\forall i \in \mathcal{N}}\{t_i^e\}$ denote the task makespan. Therefore, we have*

$$E^{comp} = \sum\nolimits_{k \in \mathcal{M}} \left[\sum\nolimits_{i \in \mathcal{N}} \left(s_{ik} \frac{M_i}{\lambda_{ik} f_k} + t_{ik} \right) P_k^d + T_{total} P_k^s \right]. \tag{37}$$

By incorporating (37) *into* (33)*, the MP can exploit the available energy margin to adjust $s_{ik}^{*(n)}$ and $o_i^{*(n)}$ more efficiently.*

4.3.3 Termination of MP and SP Iterations

At each iteration n, the MP produces the task assignment and optional-execution solution $(\boldsymbol{s}^{*(n)}, \boldsymbol{t}^{*(n)})$ by the closed-form updates in Algorithm 1 (Line 5). The corresponding optional cycle vector $\boldsymbol{o}^{*(n)}$ is recovered from $\boldsymbol{t}^{*(n)}$ through the selected processor, and the task QoS $g(\boldsymbol{o}^{*(n)})$ is evaluated by (2). The MP solution $(\boldsymbol{s}^{*(n)}, \boldsymbol{t}^{*(n)})$ and the multipliers $\boldsymbol{\lambda}^{(n)} = (\boldsymbol{\alpha}^{(n)}, \boldsymbol{\beta}^{(n)}, \boldsymbol{\gamma}^{(n)}, \mu^{(n)})$ are substituted into (17) to obtain the Lagrangian value $\mathcal{L}^{(n)} \triangleq \mathcal{L}(\boldsymbol{s}^{*(n)}, \boldsymbol{t}^{*(n)}, \boldsymbol{\alpha}^{(n)}, \boldsymbol{\beta}^{(n)}, \boldsymbol{\gamma}^{(n)}, \mu^{(n)})$. Then, the SP converts the solution $(\boldsymbol{s}^{*(n)}, \boldsymbol{t}^{*(n)})$ into a feasible task execution order and start-time vector $\boldsymbol{t}^{s(n)}$ through the topological scheduling procedure in Algorithm 2 (Lines 1–12).

Since our purpose is to obtain a high-quality, feasible solution with a low computational overhead, we consider two termination criteria for the closed-loop iteration. The first criterion evaluates solution quality from an optimality perspective, while the second criterion evaluates convergence behavior from a stability perspective.

- The relative duality gap is defined as follows:

$$\Gamma_{\text{gap}}^{(n)} = \frac{\left| g(\boldsymbol{o}^{*(n)}) - \mathcal{L}^{(n)} \right|}{\left| g(\boldsymbol{o}^{*(n)}) \right| + \xi}, \tag{38}$$

  where ξ is a small positive constant to make (38) meaningful. In (38), $g(\boldsymbol{o}^{*(n)})$ is the QoS value of the recovered primal solution at iteration n, while $\mathcal{L}^{(n)}$ is the corresponding Lagrangian surrogate value computed with the current multipliers. Therefore, the item $\left| g(\boldsymbol{o}^{*(n)}) - \mathcal{L}^{(n)} \right|$ quantifies the gap between the primal solution and the dual-guided surrogate, and (38) normalizes this gap with respect to the current primal QoS scale. A smaller $\Gamma_{\text{gap}}^{(n)}$ indicates that the current solution is closer to the dual-guided optimum, and therefore indicates better solution quality.
- From the stability perspective, we evaluate whether the primal QoS has become stable across two adjacent iterations. Specifically, we use $\left| g(\boldsymbol{o}^{*(n)}) - g(\boldsymbol{o}^{*(n-1)}) \right|$ to measures the QoS change from iteration $n-1$ to iteration n, and use the item $\left| g(\boldsymbol{o}^{*(n-1)}) \right| + \xi$ to normalize this change. Therefore, the relative objective gap is defined as follows:

$$\Delta_{\text{obj}}^{(n)} = \frac{\left| g(\boldsymbol{o}^{*(n)}) - g(\boldsymbol{o}^{*(n-1)}) \right|}{\left| g(\boldsymbol{o}^{*(n-1)}) \right| + \xi}. \tag{39}$$

Algorithm 3 Closed-Loop Joint Optimization

Input: Task DAG $G(\mathcal{T}, \mathcal{E})$, energy budget E_s, maximum iteration number $N_{\max}$
Output: Mapping $\boldsymbol{s}$, optional-execution time $\boldsymbol{t}$, schedule $\boldsymbol{t}^s$
1: Initialize $\boldsymbol{\lambda}^{(0)} = (\boldsymbol{\alpha}^{(0)}, \boldsymbol{\beta}^{(0)}, \boldsymbol{\gamma}^{(0)}, \mu^{(0)})$, step size δ_0, and tolerances $\epsilon_{\text{gap}}, \epsilon_{\text{obj}}$
2: **for** $n = 1$ **to** $N_{\max}$ **do**
3: Solve the MP and obtain $\boldsymbol{s}^{*(n)}$ and $\boldsymbol{t}^{*(n)}$ using (28) and (29)
4: Run Algorithm 2 with $(\boldsymbol{s}^{*(n)}, \boldsymbol{t}^{*(n)})$ to obtain $\boldsymbol{t}^{s(n)}$ and $\boldsymbol{t}^{e(n)}$
5: Update $\boldsymbol{\lambda}^{(n+1)}$ by (32), (33), (35), and (36)
6: Evaluate $\Gamma_{\text{gap}}^{(n)}$ by (38) and $\Delta_{\text{obj}}^{(n)}$ by (39)
7: **if** $\Gamma_{\text{gap}}^{(n)} \leq \epsilon_{\text{gap}}$ **and** $\Delta_{\text{obj}}^{(n)} \leq \epsilon_{\text{obj}}$ **then**
8: **break**
9: **end if**
10: Update the step size $\delta_{n+1} \leftarrow \delta_n / \sqrt{n}$
11: **end for**
12: **return** $(\boldsymbol{s}^{*(n)}, \boldsymbol{t}^{*(n)}, \boldsymbol{t}^{s(n)})$

The algorithm stops only when $\Gamma_{\text{gap}}^{(n)} \leq \epsilon_{\text{gap}}$ and $\Delta_{\text{obj}}^{(n)} \leq \epsilon_{\text{obj}}$ are satisfied simultaneously, where ϵ_{gap} and ϵ_{obj} are predefined small tolerances. Otherwise, the feedback updates in (32)–(36) are applied, and the iteration between MP and SP is continued until the maximum iteration number $N_{\max}$ is reached.

Algorithm 3 shows the iteration between the MP and the SP, where Line 3 performs the MP update, Line 4 considers the dependency-aware task scheduling, Line 5 feeds the SP-induced timing and energy residuals back to the multipliers, and Lines 6–7 implement the termination test. Note that our method is not limited to the QoS function described by (2), and it can be extended to the general convex scenarios.

4.3.4 Computational Complexity Analysis

The computational complexity of our method is determined by three parts in each iteration: the MP update, the SP scheduling, and the multiplier feedback.

- In the MP update, the primal variables $[s_{ik}^{*(n)}]_{N\times M}$ and $[t_{ik}^{*(n)}]_{N\times M}$ are computed for all N tasks and M processors according to Algorithm 1 (Line 5), leading to a computational cost of $\mathcal{O}(NM)$. In addition, evaluating the QoS objective and the Lagrangian $\mathcal{L}(\boldsymbol{s}^{*(n)}, \boldsymbol{t}^{*(n)}, \boldsymbol{\alpha}^{(n)}, \boldsymbol{\beta}^{(n)}, \boldsymbol{\gamma}^{(n)}, \mu^{(n)})$ also requires scanning the same task-processor pairs, and thus takes $\mathcal{O}(NM)$ operations. The complexity of the MP update is $\mathcal{O}(NM)$.
- For the SP scheduling, the proposed topological scheduling method initializes the in-degree of all tasks and maintains the Ready Queue. For a DAG $G(\mathcal{T}, \mathcal{E})$ with $|\mathcal{T}| = N$ and $|\mathcal{E}| = E$, each task enters and leaves the Ready Queue at most once, and each edge is checked once when updating the in-degree of successor tasks. Therefore, the complexity of the SP scheduling is $\mathcal{O}(N + E)$.
- For the multiplier feedback, updating α_i and γ_{ik} is performed at the task level and requires $\mathcal{O}(N)$ operations. Updating β_{ik} is performed for all task-processor pairs and requires $\mathcal{O}(NM)$ operations. The update of μ evaluates the energy terms over task-processor pairs and requires $\mathcal{O}(NM)$ operations. Hence, the complexity of the multiplier feedback is $\mathcal{O}(NM + N)$.

Taking the above three parts into account, the per-iteration complexity of our method is $\mathcal{O}(NM+N+E)$. On this basis, with at most $N_{\max}$ outer iterations, the

Table 4: Task and Platform Parameters

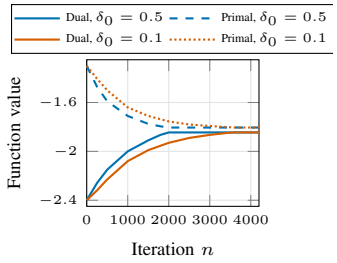
Processor Model	
M	5
V_P (V)	[0.65, 0.70, 0.75, 0.80, 0.85]
f_P (Hz)	$10^9 \times$[1.01, 1.26, 1.53, 1.81, 2.10]
P_k^d (W)	$10^{-3} \times$[184.9, 266.7, 370.4, 498.9, 655.5]
P_k^s (W)	$10^{-3} \times$[246.0, 290.1, 340.3, 397.6, 462.7]
η	$\eta \in (0, 1]$
λ_{ik}	$\lambda_{ik} = 0.9$
E_s	$E_s = \eta[MH \max_k\{P_k^s\} + \sum_i^N (M_i + O_i) \max_k\{P_k^d/f_k\}]$
Task Model	
N	$N \in \{1, 2, \ldots, 200\}$
M_i	$M_i \in [4 \times 10^7,\ 6 \times 10^8]$
O_i	$O_i \in [4 \times 10^7,\ 6 \times 10^8]$
DAG	`{LE, JPEG, FFT, MW}`
Real-Time Constraints	
H	$H = (1-\beta)\, H_{\min} + \beta\, H_{\max}$
$H_{\min}$	$H_{\min} = \min_{k\in\mathcal{M}} \left\{ \frac{\sum_{i\in\mathrm{CPT}} M_i}{\sum_{i\in\mathrm{CPT}} \lambda_{ik} f_k} \right\}$
$H_{\max}$	$H_{\max} = \max_{k\in\mathcal{M}} \left\{ \frac{\sum_{i\in\mathrm{CPT}} (M_i+O_i)}{\sum_{i\in\mathrm{CPT}} \lambda_{ik} f_k} \right\}$
D_i	$D_i = (1-\theta)\, D_i^{\min} + \theta\, D_i^{\max}$
$D_i^{\min}$	$D_i^{\min} = \min_{k\in\mathcal{M}} \left\{ \frac{M_i}{\lambda_{ik} f_k} \right\}$
$D_i^{\max}$	$D_i^{\max} = \max_{k\in\mathcal{M}} \left\{ \frac{M_i+O_i}{\lambda_{ik} f_k} \right\}$
Objective Model	
Φ	$\Phi = \sum_{i=1}^N \left(-p_i\, o_i + q_i \sqrt{o_i}\right)$

overall complexity is $\mathcal{O}(N_{\max}(NM+N+E))$. For sparse DAGs, where the number of edges is usually proportional to the number of tasks, i.e., $E = \mathcal{O}(N)$. Moreover, the number of processors is usually much smaller than the number of tasks, i.e., $M \ll N$. Therefore, the overall complexity can be simplified to $\mathcal{O}(N_{\max}N)$, i.e., the proposed method scales approximately linearly with the task number N under the given iteration number $N_{\max}$.

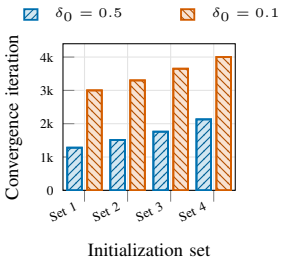
5 Experimental Results and Analysis

5.1 Experimental Setup

We conduct a set of experiments to evaluate the proposed topological-sorting-assisted dual decomposition task deployment method (TS-DD). The multicore platform contains $M = 5$ heterogeneous processors, whose parameters are adopted from prior heterogeneous real-time scheduling studies [7, 38]. The parameters regarding voltage, frequency, dynamic power, and static power are summarized in Table 4. For each task τ_i, the mandatory cycles M_i and maximum optional cycles O_i are set within the range $[4 \times 10^7,\ 6 \times 10^8]$ [21, 38], and the heterogeneity factor is set to $\lambda_{ik} = 0.9$ [28].



(a) Convergence under two initial step sizes.



(b) Initialization sensitivity.

Fig. 4: Convergence and iteration changes under different δ_0, $\alpha^{(0)}$, $\beta^{(0)}$, $\gamma^{(0)}$, and $\mu^{(0)}$ parameters.

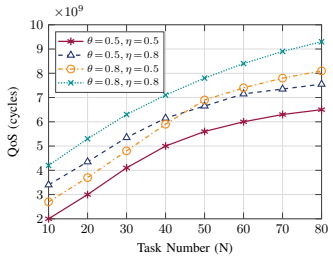
The available energy is set to $E_s = \eta E_h$, where $\eta \in (0, 1]$ is the energy-supply factor that controls the tightness of the energy constraint, and E_h is an upper bound on energy consumption, as it considers that all the tasks are executed with maximum static and dynamic powers and each task with $M_i + O_i$ cycles. The tightness of timing is controlled by θ for task-level deadlines and by β for the scheduling horizon. The QoS coefficients p_i and q_i are adopted within the range $(0, 1]$. To evaluate both random and structured dependency patterns, we randomly generate DAGs through TGFF and use four real application DAGs (LE, JPEG, FFT, and MW), which are shown in Fig. 7. Note that we can apply the proposed method to task deployment problems with different task and platform parameters, since changes in the parameters do not affect the structure of our problem.

5.2 Convergence and Optimality Gap

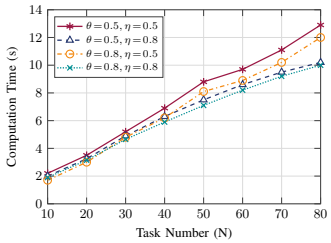
Fig. 4 first studies the influence of initial step size δ_0 on the iteration number n. In Fig. 4a, the dual and the primal values move rapidly toward each other at the beginning, and then convergence becomes slow. With $\delta_0 = 0.5$, most of the change is completed within about 2,000 iterations. With $\delta_0 = 0.1$, the same stable region is reached at about 3,600 iterations. This behavior follows the update rule in Algorithm 3: a larger initial step allows deadline and energy violations returned by the SP to change the multipliers more strongly, and thus, the MP adjusts its mapping and optional workload more quickly. A smaller step makes each correction more conservative and therefore needs more iterations.

Fig. 4a also shows that both initial step sizes $\delta_0 = 0.5$ and $\delta_0 = 0.1$ reach almost the same terminal primal and dual values. The final differences are small but nonzero. This is because the algorithm stops with finite tolerances (e.g., ϵ_{gap}, ϵ_{obj}, and $N_{\max}$). In addition, as the original problem contains discrete mapping decisions (s_{ik} in (9)), TS–DD uses relaxation (s_{ik} in (16)) and dual updates ((32)–(36)) to find a near-optimal solution.

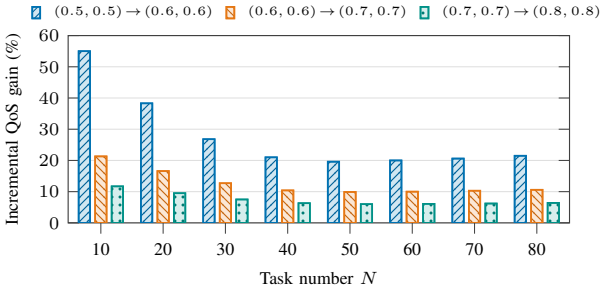
Fig. 4b further evaluates the influence of initial multipliers on the iteration number n, where $(\alpha^{(0)}, \beta^{(0)}, \gamma^{(0)}, \mu^{(0)})$ in Sets 1, 2, 3, and 4 are set to $(0, 0, 0, 0)$, $(0.5, 0.5, 0.5, 0.5)$, $(0.2, 0.5, 1, 0.5)$, and $(2, 1, 0.2, 2)$, respectively. The results show that the iteration differences among different sets are minute. TS-DD has good



(a) QoS with task number N varying.



(b) Time with task number N varying.



(c) QoS gains under different time and energy parameters (θ, η).

Fig. 5: Performance under different N, θ, and η parameters.

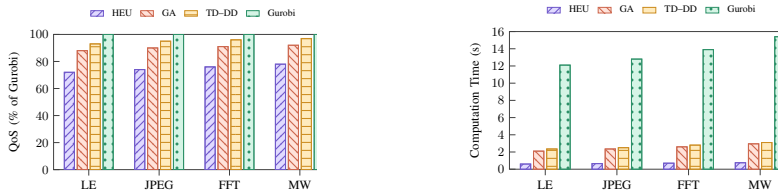
adaptability in initial multiplier selection. In addition, the trend of iteration number increases with the values of set elements, as more multiplier deviation requires more iterations to converge.

For every initialization set, increasing δ_0 from 0.1 to 0.5, the required iteration number is reduced by about 47%–57%. This result shows that the step size δ_0 has a higher priority in affecting the iteration number n than the initial multipliers $(\alpha^{(0)}, \beta^{(0)}, \gamma^{(0)}, \mu^{(0)})$.

5.3 Performance Under Limited Resource Budgets

Fig. 5a compares the system QoS when the task number N is changed from 10 to 80 under different θ and η settings, i.e., time and energy parameters. The results show that QoS increases with N, θ, and η, as more optional cycles can contribute to the QoS objective. Fig. 5c shows the influences of θ and η on the incremental QoS gains, where θ and η are changed from 0.5 to 0.8, with a step of 0.1. The QoS gain is defined as $G_{\theta\to\eta}(N) = [Q_{\eta,\eta}(N) - Q_{\theta,\theta}(N)]\,/Q_{a,a}(N) \times 100\%$, where $Q_{a,b}(N)$ denote the system QoS achieved with N tasks under given a and b parameters. The trends of the bars indicate that with the values of θ and η increasing, the improvement in QoS gain is limited. Under the given task number N, the increment is saturated when θ and η reach certain values, as the optional cycles are bounded by $o_i \le O_i$.

Fig. 5b shows the corresponding computation time, i.e., the time required to solve problem (9). When N is changed from 10 to 80, the time increases from



(a) QoS normalized by optimum.

(b) Computation time.

Fig. 6: QoS and time comparisons under different methods and applications.

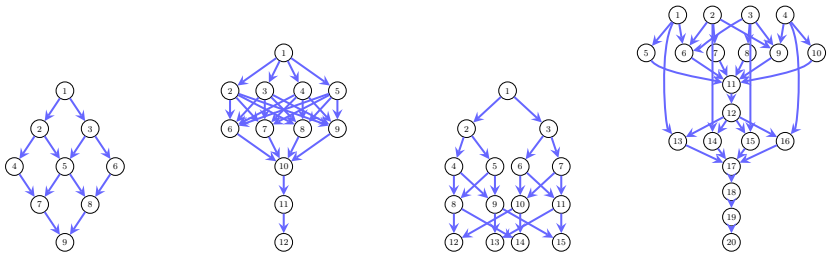
$1.7 \sim 2.2$ s to $10 \sim 13$ s, as more variables and constraints are involved in (9); the complex task precedence and conflicting time and energy constraints complicate the solving process. However, the time increment is almost linear with N, which shows the scalability of our method. This result is in line with the complexity analysis in Section 4.3.4: $\mathcal{O}(NM+N+E)$ for each iteration. We have $E = \mathcal{O}(N)$ for task DAGs and a fixed processor number $M = 5$. In each iteration, the complexity grows approximately linearly with task number N. In addition, under the given N parameter, the smaller θ and η values require relatively longer computation time, as they tighten the time and energy constraints, making the optimal solution harder to find.

5.4 Comparison With Optimal and Sub-Optimal Methods

We consider four real-world applications, e.g., LE (Laplace Equation), JPEG, FFT (Fast Fourier Transform), and MW (Montage Workflows), and compare our method with the commercial solver Gurobi (GUB) [12], which directly solves the original problem (9) and provides the reference optimum solution; a Genetic Algorithm (GA) [34], which represents population-based metaheuristic search; and a two-step heuristic (HEU) [38], which first determines a task-to-processor allocation and then schedules the start and end time of tasks, including optional cycle adjustment. The task DAGs shown in Fig. 7 include different task numbers, dependency structures, and parallelism levels.

Fig. 6a compares the solution quality (QoS) under different applications. The results show that our method achieves around 93%, 95%, 96%, and 97% of the Gurobi QoS on LE, JPEG, FFT, and MW, respectively, with a 95.25% average ratio. In contrast, GA obtains $88\% \sim 92\%$ of the Gurobi QoS, with an average of 90.25%, whereas HEU obtains $72\% \sim 78\%$, with an average of 75.00%. Therefore, TS-DD improves QoS by about 5.5% over GA and 27.0% over HEU. GA provides a sub-optimal solution, e.g., local optimum, while our method provides a global optimum, as we solve the original problem in a convex manner. HEU decomposes the original problem into several sub-problems and solves them in sequence. In contrast, our method provides a feedback mechanism among the sub-problems to improve overall performance.

Fig. 6b compares the computation time of different methods. From it, we can see that the computation times of Gurobi, TS-DD, GA, and HEU are $12.10 \sim 15.40$ s (13.55 s on average), $2.35 \sim 3.10$ s (2.6875 s on average), $2.10 \sim 2.95$ s (2.50 s on average), and $0.60 \sim 0.75$ s (0.675 s on average), respectively. The result



(a) LE (b) JPEG (c) FFT (d) MW

Fig. 7: Task DAGs of real-world applications.

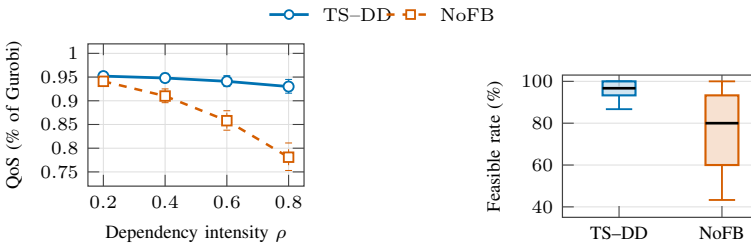
shows that TS-DD uses only 19.8% of the Gurobi time, while achieving 95.25% of the Gurobi QoS; we can make a better trade-off between computation time and solution quality. Although the computation time of HEU is low, the solution quality is hard to control, as it only provides a feasible solution. Compared with HEU, TS–DD provides a controllable trade-off between computation time and solution quality by adjusting the maximum iteration number $N_{\max}$ (see convergence in Fig. 4 and complexity analysis in Section 4.3.4). GA and TS-DD have similar computation times, but TS-DD has higher QoS than GA. GA contains several steps, such as selection, reproduction, mutation, and crossover; the computation time and solution quality are sensitive to the algorithm parameter selection.

5.5 QoS Enhancing Mechanism

We introduce a ratio $\rho = N_{\mathrm{CP}}/N$ to describe the intensity of task dependency, where N_{CP} and N are the numbers of tasks on the critical path and in the whole DAG, respectively. A large value of ρ means this DAG has a complex structure and will lead to more precedence-induced waiting. We set $\rho \in \{0.2, 0.4, 0.6, 0.8\}$ and $N \in \{20, 40, 80\}$. For each (N, ρ) pair, we randomly generate 30 DAGs and show the achieved QoS under TS–DD and NoFB methods. Compared with TS–DD, NoFB considers the same MP and topological scheduler but removes the closed-loop feedback from SP to MP [21], i.e., omit the task completion-time residual $t_i^e - H$ in the updates of γ_{ik} and β_{ik} (see (34), (35), and (36)).

Fig. 8a shows the average QoS of TS-DD and NoFB, normalized by the Gurobi result. When ρ increases from 0.2 to 0.8, due to task dependency and real-time restrictions, the normalized QoS of TS-DD decreases only from 0.952 to 0.930; the degradation is acceptable. However, without the feedback from SP to MP, i.e., the further adjustment of task scheduling, the task dependency and real-time constraints are harder to satisfy, especially when the DAG structure is complex, leading to lower normalized QoS for NoFB (0.941 $\rightarrow$ 0.781). Note that when $\rho = 0.2$, the DAG dependency structure is relatively simple and imposes fewer restrictive precedence constraints. TS-DD and NoFB achieve similar QoS values. With ρ increasing, e.g., $\rho = 0.8$, the complex task dependency makes the closed-loop feedback from SP to MP essential, leading to a large QoS gap between TS-DD and NoFB.

Fig. 8b compares the feasibility rates across different ρ and N parameters. The feasibility rate is defined as $R_{\mathrm{fea}} = N_{\mathrm{fea}}/N_{\mathrm{run}} \times 100\%$, where N_{fea} and N_{run} are



(a) QoS normalized by optimum.

(b) Task schedulability.

Fig. 8: QoS and feasibility comparisons with and without feedback mechanism.

the number of feasible cases and total cases, respectively. Note that only when all constraints, such as deadline, energy, precedence, and non-overlap, are satisfied, the found solution is feasible. The results show that TS-DD achieves an average feasibility rate of 96.7%, compared with 80.0% for NoFB. When the values of θ and η are small, under the limited time and energy budgets, the original problem (9) is infeasible, no matter through Gurobi or TS-DD. Under the same parameters, the feasibility rate of TS-DD is higher than NoFB. Together with Fig. 8a, the closed-loop feedback mechanism introduced in TS-DD enhances system QoS and task schedulability, especially for complex DAGs.

6 Conclusion

In this paper, we investigated the allocation and scheduling problem for dependent IC tasks on heterogeneous multicore platforms for QoS improvement. The target problem was formulated as an MINLP that jointly determines task-to-processor assignment, optional-execution allocation, and start-time determination, while enforcing deadline, energy, precedence, and non-overlap constraints. To improve the scalability of the proposed method, we developed a TS-DD framework that decomposes the original problem into an MP for task-to-processor assignment and optional-execution allocation, and an SP for dependency-aware scheduling. The MP and SP are solved in sequence, with low-complexity algorithms. On this basis, a closed-loop multiplier update mechanism is introduced to adjust the MP solution according to the constraint violations from the SP, enabling the algorithm to balance QoS improvement with task schedulability. Experimental results show that TS-DD achieves 95.25% of the optimal QoS found by the Gurobi solver while using only 19.8% of its computation time. The runtime of TS-DD grows approximately linearly with task DAG scale, enhancing the scalability of TS-DD under complex application and resource-constrained scenarios.

References

1. Aydin, H., Melhem, R., Mossé, D., Mejía-Alvarez, P.: Optimal scheduling of imprecise computation tasks in the presence of multiple faults. In: International Conference on Real-Time Computing Systems and Applications, pp. 1–6 (2000)
2. Aydin, H., Melhem, R., Mosse, D., Mejia-Alvarez, P.: Optimal reward-based scheduling for periodic real-time tasks. IEEE Transactions on Computers **50**(2), 111–130 (2001)

3. Balasubramanian, P., Zaheen, S.M.M.A.H., Maskell, D.L.: Machine learning using approximate computing. Journal of Low Power Electronics and Applications **15**(2), 21 (2025)
4. Boyd, S., Vandenberghe, L.: Convex Optimization. Cambridge Univ. Press, Cambridge, U.K. (2004)
5. Burer, S., Letchford, A.N.: Non-convex mixed-integer nonlinear programming: A survey. Surveys in Operations Research and Management Science **17**(2), 97–106 (2012)
6. Chakraborty, S., Saha, S., Själander, M., Mcdonald-Maier, K.: PREPARE: Power-aware approximate real-time task scheduling for energy-adaptive QoS maximization. ACM Transactions on Embedded Computing Systems **20**(5), 1–25 (2021)
7. Chen, G., Huang, K., Knoll, A.: Energy optimization for real-time multiprocessor systems-on-a-chip with optional DVFS and DPM combination. ACM Transactions on Embedded Computing Systems **13**(3), 1–21 (2014)
8. Chen, J., He, Y., Zhang, Y., Han, P., Du, C.: Energy-aware scheduling for dependent tasks in heterogeneous multiprocessor systems. Journal of Systems Architecture **129**, 102598 (2022)
9. Cortes, L.A., Eles, P., Peng, Z.: Quasi-static assignment of voltages and optional cycles in imprecise-computation systems with energy considerations. IEEE Transactions on Very Large Scale Integration (VLSI) Systems **14**(10), 1117–1129 (2006)
10. Esmaili, A., Nazemi, M., Pedram, M.: Energy-aware scheduling of task graphs with imprecise computations and end-to-end deadlines. ACM Transactions on Design Automation of Electronic Systems **25**(1), 1–21 (2019)
11. Gammoudi, A., BenZina, A., Khalgui, M., Chillet, D.: Energy-efficient scheduling of real-time tasks in reconfigurable homogeneous multicore platforms. IEEE Transactions on Systems, Man, and Cybernetics: Systems **50**(12), 5092–5105 (2020)
12. Guasque, A., Balbastre, P.: Evaluation and comparison of integer programming solvers for hard real-time scheduling. IEICE Transactions on Information and Systems **E105.D**(10), 1726–1733 (2022)
13. He, J., Xiao, Y., Bogdan, C., Nazarian, S., Bogdan, P.: A design methodology for energy-aware processing in unmanned aerial vehicles. ACM Transactions on Design Automation of Electronic Systems **27**(4), 1–20 (2021)
14. Islam, A., Ghose, M.: BQProfit: Budget and QoS-aware task offloading and resource allocation for profit maximization under an MEC platform. Journal of Systems Architecture **166** (2025)
15. Kanoun, K., Mastronarde, N., Atienza, D., van der Schaar, M.: Online energy-efficient task-graph scheduling for multicore platforms. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **33**(8), 1194–1207 (2014)
16. Li, D., Wu, J.: Minimizing energy consumption for frame-based tasks on heterogeneous multiprocessor platforms. IEEE Transactions on Parallel and Distributed Systems **26**(3), 810–823 (2015)
17. Liu, J.W.S., Shih, W.K., Lin, K.J., Bettati, R., Chung, J.Y.: Imprecise computations. Proceedings of the IEEE **82**(1), 83–94 (1994)
18. Liu, Y., Wang, S., Zhao, Q., Du, S., Zhou, A., Ma, X., Yang, F.: Dependency-aware task scheduling in vehicular edge computing. IEEE Internet of Things Journal **7**(6), 4961–4971 (2020)
19. Mo, L., Kritikakou, A., Sentieys, O.: Decomposed task mapping to maximize QoS in energy-constrained real-time multicores. In: IEEE International Conference on Computer Design, pp. 493–500 (2017)
20. Mo, L., Kritikakou, A., Sentieys, O.: Controllable QoS for imprecise computation tasks on DVFS multicores with time and energy constraints. IEEE Journal on Emerging and Selected Topics in Circuits and Systems **8**(4), 708–721 (2018)
21. Mo, L., Li, X., Kritikakou, A., You, P.: Optimal IC task mapping to maximize QoS on heterogeneous multicore systems. IEEE Transactions on Circuits and Systems II: Express Briefs **71**(4), 2019–2023 (2024)
22. Oliveira, R.S., Fraga, J.S.: Scheduling imprecise computation tasks with intra-task/inter-task dependence. IFAC Proceedings Volumes **29**(5), 63–68 (1996)
23. Qin, Y., Zeng, G., Kurachi, R., Matsubara, Y., Takada, H.: Energy-aware task allocation for large task sets on heterogeneous multiprocessor systems. In: IEEE International Conference on Embedded and Ubiquitous Computing, pp. 158–165 (2018)
24. Qin, Y., Zeng, G., Kurachi, R., Matsubara, Y., Takada, H.: Energy-aware task allocation for heterogeneous multiprocessor systems by using integer linear programming. Journal of Information Processing **27**, 136–148 (2019)

25. Saha, S., Chakraborty, S., Agarwal, S., Själander, M., McDonald-Maier, K.D.: ARCTIC: Approximate real-time computing in a cache-conscious multicore environment. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **43**(10), 2944–2957 (2024)
26. Saha, S., Chakraborty, S., Zhai, X., Ehsan, S., McDonald-Maier, K.D.: ACCURATE: Accuracy maximization for real-time multicore systems with energy-efficient way-sharing caches. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **41**(12), 5246–5260 (2022)
27. Saha, S., Zhai, X., Ehsan, S., Majeed, S., McDonald-Maier, K.: RASA: Reliability-aware scheduling approach for FPGA-based resilient embedded systems in extreme environments. IEEE Transactions on Systems, Man, and Cybernetics: Systems **52**(6), 3885–3899 (2022)
28. Sandokji, S., Eassa, F.: Communication and computation aware task scheduling framework toward exascale computing. International Journal of Advanced Computer Science and Applications **10**(7), 119–128 (2019)
29. Su, R., Riahi, A., Natalizio, E., Moyal, P., Saint-Jore, A., Ye-Qiong, S.: Assessing intra- and inter-community trustworthiness in IoT: A role-based attack-resilient dynamic trust management model. Internet of Things **36**, 1–21 (2024)
30. Suyyagh, A., Zilic, Z.: Energy and task-aware partitioning on single-ISA clustered heterogeneous processors. IEEE Transactions on Parallel and Distributed Systems **31**(2), 306–317 (2020)
31. Wan, G., Leung, J.Y.T., Pinedo, M.L.: Scheduling imprecise computation tasks on uniform processors. Information Processing Letters **104**(2), 45–52 (2007)
32. Wang, Y., Li, K., Chen, H., He, L., Li, K.: Energy-aware data allocation and task scheduling on heterogeneous multiprocessor systems with time constraints. IEEE Transactions on Emerging Topics in Computing **2**(2), 134–148 (2014)
33. Wei, T., Zhou, J., Cao, K., Cong, P., Chen, M., Zhang, G., Hu, X.S., Yan, J.: Cost-constrained QoS optimization for approximate computation real-time tasks in heterogeneous MPSoCs. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **37**(9), 1733–1746 (2017)
34. Yano, A., Azumi, T.: CQGA-HEFT: Q-learning-based DAG scheduling algorithm using genetic algorithm in clustered many-core platform. Journal of Information Processing **30**, 659–668 (2022)
35. Yao, J., Ansari, N.: Enhancing federated learning in fog-aided IoT by cpu frequency and wireless power control. IEEE Internet of Things Journal **8**(5), 3438–3445 (2021)
36. Yu, H., Veeravalli, B., Ha, Y.: Dynamic scheduling of imprecise-computation tasks in maximizing QoS under energy constraints for embedded systems. In: Asia and South Pacific Design Automation Conference, pp. 452–455 (2008)
37. Zhang, K., Gao, Q., Lü, J., Ogorzałek, M., Deng, Y.: A subgraph-based hierarchical Q-learning approach to optimal resource scheduling for complex industrial networks. IEEE Transactions on Circuits and Systems I: Regular Papers **71**(5), 2146–2156 (2024)
38. Zhou, J., Yan, J., Wei, T., Chen, M., Hu, X.S.: Energy-adaptive scheduling of imprecise computation tasks for QoS optimization in real-time MPSoC systems. In: Design, Automation and Test in Europe Conference, pp. 1402–1407 (2017)