HAL open science

Approximation-aware Task Deployment on Heterogeneous Multi-core Platforms with DVFS

Xinmei Li, Lei Mo, Angeliki Kritikakou, Olivier Sentieys

To cite this version:

Xinmei Li, Lei Mo, Angeliki Kritikakou, Olivier Sentieys. Approximation-aware Task Deployment on Heterogeneous Multi-core Platforms with DVFS. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2022, pp.1-15. 10.1109/TCAD.2022.3222293 . hal-03854671

HAL Id: hal-03854671

https://hal.science/hal-03854671v1

Submitted on 16 Nov 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

Approximation-aware Task Deployment on Heterogeneous Multi-core Platforms with DVFS

Xinmei Li*, *Student Member, IEEE,* Lei Mo*, *Member, IEEE,* Angeliki Kritikakou†, *Member, IEEE,* and Olivier Sentieys†, *Member, IEEE*

***Abstract*—Heterogeneous multi-core platforms, such as ARM big.LITTLE are widely used to execute embedded applications under multiple and contradictory constraints, such as energy consumption and real-time execution. To fulfill these constraints and optimize system performance, application tasks should be efficiently mapped on multi-core platforms. Embedded applications are usually tolerant to approximated results but acceptable Quality-of-Service (QoS). Modeling embedded applications by using the elastic task model, namely, Imprecise Computation (IC) task model, can balance system QoS, energy consumption, and real-time performance during task deployment. However, state-of-the-art approaches seldom consider the problem of IC task deployment on heterogeneous multi-core platforms. They typically neglect task migration, which can improve the solutions due to its flexibility during the task deployment process. This paper proposes a novel QoS-aware task deployment method to maximize system QoS under energy and real-time constraints, where frequency assignment, task allocation, scheduling, and migration are optimized simultaneously. The task deployment problem is formulated as mixed-integer non-linear programming. Then, it is linearized to mixed-integer linear programming to find the optimal solution. Furthermore, based on problem structure and problem decomposition, we propose a novel heuristic with low computational complexity. The sub-problems regarding frequency assignment, task allocation, scheduling, and adjustment are considered and solved in sequence. Finally, the simulation results show that the proposed task deployment method improves the system QoS by 31.2% on average (up to 112.8%) compared to the state-of-the-art methods and the designed heuristic achieves about 53.9% (on average) performance of the optimal solution with a negligible computing time.**

***Index Terms*—Heterogeneous multi-core, task deployment, task migration, imprecise computation, quality-of-service.**

I. Introduction

SINGLE-core and homogeneous multi-core platforms, namely, Symmetric Multi-core Processors (SMP), cannot follow the increasing pace of requirements for high computation capabilities and low energy consumption of embedded systems. As a result, heterogeneous multi-core platforms, namely, Asymmetric Multi-core Processors (AMP), such as ARM big.LITTLE platforms are widely used [1]. Applications can be executed on multiple processors simultaneously to achieve parallel and diverse processing. Energy efficiency and real-time execution are the primary concerns during system design since 1) embedded systems usually have energy constraints, especially when they are implemented on battery-powered platforms with limited energy capacity, and 2) many critical applications (e.g., target tracking or robot control) require real-time responsiveness, for an application's deadline miss can lead to serious or even disastrous consequences [2], [3].

To improve energy efficiency, platforms have been enhanced with Dynamic Voltage and Frequency Scaling (DVFS) [4], which can dynamically adjust the supply voltage and clock frequency of a processor to change task execution time and energy. In real-time application domains, approximated results obtained in time are preferred over accurate results obtained after the deadline. For example, the JPEG2000 codec [5] supports low-quality images rather than simply failing the execution in a limited time. In *k-means* algorithm [6], 5% of classification accuracy loss can achieve $50\times$ energy saving. In automotive systems [7], an approximate result produced by the traction control is better than an accurate result arriving too late. Such applications can be modeled by the Imprecise Computation (IC) task [8]. A task can be logically decomposed into a mandatory subtask and an optional subtask in the IC model. The mandatory subtask must be completed before a deadline to generate a baseline Quality-of-Service (QoS). In contrast, the optional subtask can be incompletely executed at the cost of decreased quality. The longer the optional subtasks are executed, the better the QoS of results. However, the requirements for high system QoS, low energy consumption, and real-time task execution often conflict with each other. The more the optional subtasks are executed, the higher the QoS is, while more energy and time are consumed. To balance these contradictory requirements, we propose a novel task deployment approach that simultaneously optimizes task allocation (on which processor each task is executed), task scheduling (when each task starts and ends its execution), and task frequency assignment. By adequate IC task deployment on DVFS-enabled AMP platforms, the system QoS can be further improved under limited system sources and application constraints.

A. Related Work

Table I summarizes several representative works about task deployment on multi-core platforms. Depending on the problem's objectives, the deployment approaches can be Energy-

*X. Li and L. Mo are with the School of Automation, Southeast University, 210096 Nanjing, China. E-mails: xinmeili@seu.edu.cn, lmo@seu.edu.cn.

†A. Kritikakou and O. Sentieys are with the University of Rennes, INRIA, IRISA, CNRS, 35042 Rennes, France. E-mails: angeliki.kritikakou@irisa.fr, olivier.sentieys@irisa.fr.

This work was supported in part by the National Key Research and Development Program of China under Grant 2022YFF0902800, in part by the Fundamental Research Funds for the Central Universities of China under Grants 2242021R10113 and 2242022K30038, in part by the Natural Science Foundation of Jiangsu Province of China under Grant BK20202006, in part by the Southeast University "Zhishan Scholars" Projects under Grant 2242021R40003, and in part by the Jiangsu Provincial Key Research and Development Program under Grant BE2022135.

TABLE I
TASK DEPLOYMENT METHODS

Ref.	*Task*				*Platform*		*Variables*				*Constraints*		*Objective*	
	D	ID	P	IP	HO	HE	TM	FA	TA	TS	RT	ES	QA	EA
[9]	√		√		√			√		√	√			√
[10]	√		√		√			√	√	√	√			√
[11]		√	√			√		√	√		√			√
[12]	√		√			√		√		√	√			√
[13]	√		√			√		√	√		√			√
[14]	√		√			√		√	√		√			√
[15]		√	√		√		√	√	√		√			√
[16]		√	√		√		√	√	√	√	√			√
[17]		√	√		√		√	√	√	√	√			√
[18]		√	√			√	√	√	√	√	√			√
[19]		√		√	√				√	√	√		√	
[20]		√		√	√				√		√	√	√	
[21]		√		√	√		√	√	√	√	√		√	
[22]		√		√	√			√	√		√	√	√	
[23]	√			√	√			√	√		√	√	√	
[24]	√			√	√			√	√	√	√	√	√	
[25]		√		√		√		√	√	√	√	√	√	
[5]	√			√		√		√		√	√	√	√	
[26]	√			√		√		√		√	√	√	√	
[27]	√			√		√	√		√	√	√	√	√	
Prop.	√			√		√	√	√	√	√	√	√	√	

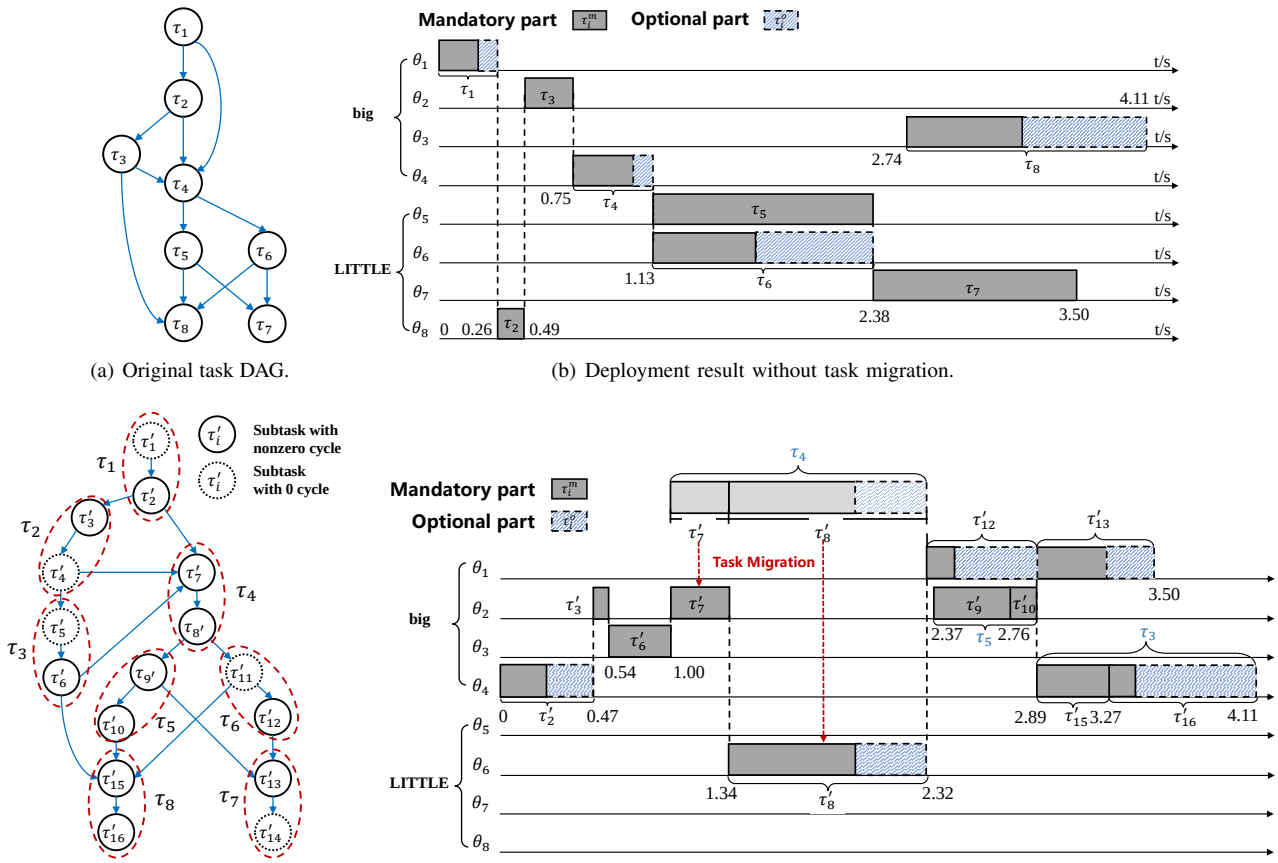
Aware (EA) or QoS-Aware (QA). The tasks can be Independent (I) or Dependent (D) and Precise (P) or Imprecise (IP). The platforms can be homogeneous (HO) or heterogeneous (HE). DVFS and task deployment can be used to manage system resources. The optimization variables include Task Migration (TM), Frequency Assignment (FA), Task Allocation (TA), and Task Scheduling (TS). These problems include Real-Time (RT) and Energy Supply (ES) constraints during the task deployment process.

1) Energy-aware Task Deployment: The majority of energy-aware task deployment approaches usually aim at minimizing energy consumption under system resource and application constraints [9]–[18]. For the homogeneous platforms, DVFS and DPM are combined in [9] and [10] to enhance the energy efficiency, where task-to-processor allocation is fixed in [9]. By contrast, task allocation, scheduling, and frequency assignment are jointly optimized in [10]. The extensions from homogeneous platforms to heterogeneous platforms are not straightforward, as additional variables regarding the core selection and Voltage/Frequency (V/F) selection are included in the task deployment problem. These variables are usually coupled with each other nonlinearly, which increases the difficulties of solving the problem. For the heterogeneous multi-core platforms, the allocation of independent tasks and the assignment of frequency are considered in [11], and mapping the dependent tasks on the NoC platform through task allocation and task scheduling is studied in [12]. Compared with [12], the approach in [11] mainly focuses on system-level DVFS, and the frequency of each processor cannot be adjusted individually. In [13] and [14], DVFS is combined into the task allocation process to minimize energy consumption under real-time constraints. Besides DVFS and DPM, task migration is also used in multi-core platforms to optimize energy consumption. More precisely, task migration is introduced into the allocation/scheduling process to deploy independent tasks on homogeneous platforms for optimizing energy consumption [15]–[17], where [15] mainly focuses on task allocation, while [16] and [17] consider both task allocation and task scheduling. As the processors are homogeneous and the tasks are independent, task migration in the above studies is usually used to minimize the overall energy consumption. A heterogeneous platform is considered in [18], where energy-aware task deployment is performed through DVFS, task allocation, scheduling, and migration. However, the approaches mentioned above are mainly based on the precise tasks, i.e., the execution cycles of each task are fixed. Therefore, the adjustment of optional subtasks is not taken into account, and thus, the improvement of system QoS is limited.

2) QoS-aware Task Deployment: The works consider QoS-aware task deployment problems adopting the IC task model and aim to maximize system QoS under real-time and/or energy constraints [5], [19]–[27]. The target multi-core platforms can be homogeneous [19]–[24] or heterogeneous [5], [25]–[27]. On the one hand, the reward-based (QoS-based) task allocation and scheduling problems are considered in [19] to improve the overall system QoS under task deadlines. Taking energy consumption into account, an energy-adaptive and QoS-driven task mapping method is studied in [20]. Compared with [19] and [20], the influences of frequency adjustment and task migration are studied in [21] during the reward-based task scheduling process, but the portions of task workloads are fixed during the migration process. To improve QoS with energy and time constraints, DVFS is considered in [22]. However, the tasks are independent in the above studies. For dependent IC tasks, DVFS is incorporated into the task mapping process considering time and energy budgets [23], [24]. On the other hand, the allocation of independent IC tasks on heterogeneous platforms with DVFS is considered in [25]. For dependent tasks, [5] and [26] consider DVFS and optimize frequency assignment and task scheduling simultaneously to enhance the system QoS under real-time and energy supply constraints. However, the above methods mainly focus on task allocation and/or task scheduling, where task migration is not considered. The task allocation, scheduling, and migration are jointly optimized in [27] to enhance QoS for heterogeneous platforms, but without considering DVFS.

B. Illustration Example

We use the example shown in Fig. 1 to describe and motivate our approach, where the deployment results from allocating and executing the dependent IC tasks on the AMP platforms without and with task migration are compared in Fig. 1(b) and Fig. 1(d). The AMP platform can support per-core DVFS, e.g., ARM DynamIQ big.LITTLE platform [28]. As the IC tasks are dependent, a Directed Acyclic Graph (DAG) $G(T, E)$ is used to describe these tasks, where the vertices T denote the tasks, and the edges E represent the dependencies between the tasks, as shown in Fig. 1(a). We assume that the cycles of mandatory and optional subtasks, namely, M_i and o_i, are within the range $[4 \times 10^7, 6 \times 10^8]$ [20], and the deadlines of the tasks are set to $D_1 = 3.27$ s, $D_2 = 3.48$ s, $D_3 = 3.50$ s, $D_4 = 4.06$ s, $D_5 = 3.57$ s, $D_6 = 3.42$ s, $D_7 = 3.50$ s,



(a) Original task DAG.

(b) Deployment result without task migration.

(c) DAG with task migration.

(d) Deployment result with task migration.

Fig. 1. The example of task deployment results under different schemes.

and $D_8 = 4.11$ s, which are based on the length of the critical path. The system QoS has a linear relationship with the cycles of optional subtasks [29], e.g., $f_i(o_i) = k_i o_i + R_i$, where k_i and R_i are constants. If task migration is performed, a task can be divided into two subtasks executed on the processors of the big and LITTLE clusters, respectively. Based on $G(T, E)$, a new DAG $G(T', E')$ is generated, as shown in Fig. 1(c). According to the task deployment results, the system QoS with and without task migration are 6.2022×10^7 (cycles) and 3.2581×10^7 (cycles), respectively, under the same real-time and energy supply constraints. Due to the task migration, tasks τ_4, τ_5, and τ_8 are divided into two subtasks, i.e., $\tau_4 \rightarrow \{\tau'_7, \tau'_8\}$, $\tau_5 \rightarrow \{\tau'_9, \tau'_{10}\}$, and $\tau_8 \rightarrow \{\tau'_{15}, \tau'_{16}\}$. These subtasks can be executed on different processors with different V/F levels. Note that task migration is not necessary for each task, and thus, the cycles of some subtasks are 0, e.g., subtasks τ'_1, τ'_4, τ'_5, τ'_{11} and τ'_{14} in Fig. 1(d). Therefore, by using task migration, we can better use system resources to execute more optional cycles to improve system QoS. For instance, without task migration, task τ_4 is executed on processor θ_4, and the number of its optional cycles o_4 is 1.7×10^8. On the contrary, when task migration is considered, the subtasks of τ_4, namely τ'_7 and τ'_8, are executed on different processors θ_2 and θ_6, and thus, the achieved optional cycles are 4.9×10^8.

C. Contributions

Complementary to the state-of-the-art, this paper proposes a joint optimization method for IC task deployment on AMP using task allocation, task scheduling, and task migration. The structure of the proposed task deployment schemes is shown in Fig. 2. We aim to determine: 1) which processor the task should be executed on; 2) what voltage/frequency level should be used for each task; 3) what is the execution sequence of the tasks on each processor; 4) when should a task start its execution; and 5) how many cycles of the optional subtasks are needed to be executed, such that the system QoS is maximized, and at the same time satisfying the real-time and energy constraints. Our main contributions are summarized as follows:

(1) To improve system QoS, IC tasks and task migration are involved in the task deployment process. We formulate the IC tasks deployment problem that simultaneously optimizes task allocation, frequency assignment, task sequence, task migration, task start time, and task adjustment as an MINLP problem. The objective is to maximize QoS without violating real-time and energy constraints.

(2) The nonlinear items are caused by the product of optimization variables related to task allocation, frequency assignment, task migration, and task adjustment. We prove that by replacing the nonlinear items with auxiliary variables and additional linear constraints in the optimization problem, the MINLP problem can be equivalently transformed into

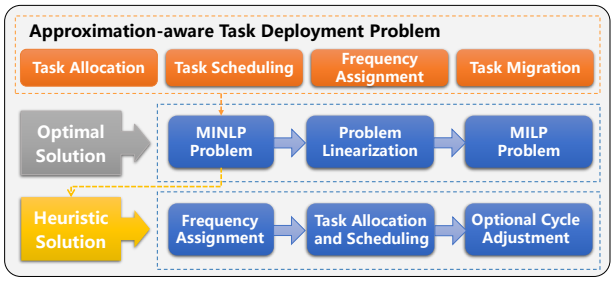


Fig. 2. The structure of the proposed task deployment scheme.

an MILP problem to find the optimal solution.

(3) Based on the idea of problem decomposition and the problem structure, we design a novel heuristic task deployment method. The proposed method decomposes the original problem into three smaller easier-to-solve problems. The first problem is an ILP problem for frequency assignment, and we utilize the Greedy algorithm to solve it. The second problem is an MILP problem for task allocation and scheduling, and we design a three-step heuristic to solve this problem. It includes task layer classification, task allocation and scheduling, and frequency adjustment. The third problem considers the adjustment of optional cycles, and we design a low-complexity algorithm to tackle it.

(4) Finally, we perform extensive simulations to analyze the solution quality, computation time, and scalability of the proposed task deployment scheme. The results show that the proposed task deployment scheme outperforms other schemes regarding QoS improvement and energy efficiency. Under the resource-limited situation, the joint optimization algorithm can improve the QoS by 31.2% on average (up to 112.8%). The heuristic algorithm can obtain an acceptable suboptimal solution in a negligible time, achieving about 53.9% (on average) performance of the optimal solution.

D. Paper Organization

The remainder of the paper is organized as follows. Section II presents the system models and problem formulation. Section III describes the details of the problem linearization, and Section IV presents the heuristic task deployment algorithm. Section V discusses the simulation results. Finally, Section VI concludes this paper.

II. System Models and Problem Formulation

A. System Model

1) Task Model: We consider a real-time task set $\mathcal{T}$ with N periodic IC tasks $\{\tau_1, \ldots, \tau_N\}$.

Definition 2.1: A task τ_i is defined as an IC task that can be logically decomposed into a mandatory subtask τ_i^m with M_i cycles and an optional subtask τ_i^o with o_i cycles. The mandatory subtask should be completed before the deadline to generate a baseline QoS. The optional subtask is to be executed after the mandatory subtask and still completed before the deadline, if there are available resources in the system to execute the optional subtasks without missing the deadlines. The more the optional subtasks are executed, the better the obtained QoS is.

For each task τ_i, M_i and o_i are measured by the Worst-Case Execution Cycles (WCEC). The mandatory cycles M_i are fixed, while the optional cycles o_i are adjustable and o_i has an upper bound O_i, i.e., $0 \leq o_i \leq O_i$. The tasks are released at time 0 and each task τ_i has a deadline D_i. In addition, the scheduling horizon H is also the period of task τ_i. As the example shown in Fig. 1(a), the tasks in the set $\mathcal{T}$ are described by a DAG $G(T, E)$, where the vertices T denote the tasks, and the edges E denote the dependencies between the tasks. Each task τ_i cannot start execution until the input data from all its predecessors have arrived. At the same time, the output data is concurrently available for all its successors only when it completes execution.

The system QoS is closely related to the task optional part. Usually, the more cycles of the optional subtasks are executed, the higher the generated QoS. The relationship between system QoS and optional tasks is measured by a QoS function. Linear and general concave functions are considered as the most realistic and typical QoS representation in the literature [20], [30], as they adequately capture the behavior of many application areas, such as image and speech processing, control engineering, and automatic target recognition. In this paper, we consider the linear function, $f_i(o_i) = k_i o_i + R_i$, describing the linear relationship between system QoS and optional cycles o_i, where R_i is the baseline QoS after executing mandatory cycles.

2) Platform Model: We consider a multi-core AMP platform with M processors $\{\theta_1, \ldots, \theta_M\}$. According to the characteristics of many heterogeneous platforms (e.g., ARM big.LITTLE) [27], we introduce the concept of *cluster*. For instance, the big.LITTLE platform consists of two heterogeneous clusters: the big cluster and the LITTLE cluster. The processors in the same cluster are homogeneous. Without loss of generality, we describe the proposed approach considering two clusters, such as in the ARM big.LITTLE platform.

Let $\mathcal{P} \triangleq \{\mathcal{P}_B, \mathcal{P}_L\}$ denote the processor set, where $\mathcal{P}_B \triangleq \{\theta_1, \ldots, \theta_{M_B}\}$ is the set of the processors in the big cluster, while $\mathcal{P}_L \triangleq \{\theta_{M_B+1}, \ldots, \theta_{M_B+M_L}\}$ is the set of the processors in the LITTLE cluster. Moreover, M_B and M_L are the number of processors in the big and LITTLE clusters, respectively, and $M = M_B + M_L$. We assume that the processors in $\mathcal{P}_B$ have l_B discrete V/F levels $\mathcal{VF}_B \triangleq \{(V_1, f_1), \ldots, (V_{l_B}, f_{l_B})\}$, while the processors in $\mathcal{P}_L$ have l_L discrete V/F levels $\mathcal{VF}_L \triangleq \{(V_{l_B+1}, f_{l_B+1}), \ldots, (V_{l_B+l_L}, f_{l_B+l_L})\}$.

To indicate the heterogeneity among the processors in different clusters, we introduce a factor $\gamma_{i,l} \in (0, 1]$ [11], which represents the efficiency factor of heterogeneous processors executing task τ_i with (V_l, f_l). Therefore, when (V_l, f_l) is used to execute τ_i with $M_i + o_i$ cycles, the task execution time is $(M_i + o_i)/(\gamma_{i,l} f_l)$. In addition, since the processors are connected by the high-speed data bus, the communication costs (i.e., time and energy) among the processors are negligible, compared with task execution time and energy [11]. Due to the special interconnection bus designed for data transmission between the clusters, it is feasible to migrate a task from one cluster to another cluster during the task execution process [28]. We consider the task migrated only between the big and LITTLE clusters, since processors in the same cluster are homogeneous. The overhead of task migration is considered in the WCEC of the mandatory part, assuming that task migration

always takes place. Let τ'_{2i-1} and τ'_{2i} denote the subtasks of task τ_i executed on the processors of different clusters when task migration is performed. To model the workload of τ'_{2i-1} and τ'_{2i}, we introduce two continuous variables μ_{2i-1} and μ_{2i}, where we have $\mu_{2i-1}+\mu_{2i}=1$ and $\mu_{2i-1},\mu_{2i}\in[0,1]$. Therefore, the workloads of τ'_{2i-1} and τ'_{2i} are given by $\mu_{2i-1}(M_i+o_i)$ and $\mu_{2i}(M_i+o_i)$, respectively. By adjusting the values of μ_{2i-1} and μ_{2i}, we can change the workload proportions of task τ_i executed on different processors. Due to the dependencies between the subtasks, τ'_{2i} starts its execution only after its predecessor τ'_{2i-1} is completed. Therefore, we obtain a new DAG $G(T',E')$, as shown in Fig. 1(c). Note that the task set $\{\tau'_{2i-1},\tau'_{2i}\}$ $(1\le i\le N)$ can be rewritten as $\{\tau'_i\}$ $(1\le i\le 2N)$.

We consider that processors can operate in two modes: *idle* and *active* [9]. A processor executes a task in active mode; if the assigned tasks are finished, the processor goes into idle mode. During this switching process, processors' energy and time consumption are very small, compared with the task execution time and energy. Therefore, the transition time and energy are considered to be incorporated into task execution time and energy [18]. We assume the target platform can support per-core DVFS, e.g., ARM DynamIQ big.LITTLE platform [28]. Hence, the power consumption of a processor θ_k working on the V/F level (V_l,f_l) is given by

$$P_{core,l}=P_{sta,l}+P_{dyn,l}+P_{on}, \tag{1}$$

where $P_{sta,l}=L_g(V_lI_{sub}+|V_{bs}|I_j)$ is the static power when ready to execute tasks, $P_{dyn,l}=C_{eff}{V_l}^2f_l$ is the dynamic power during the task execution, and P_{on} is the inherent power which keeps processors on. The static power $P_{sta,l}$ is mainly contributed by the sub-threshold leakage current I_{sub}, the reverse bias junction current I_j, the number of devices in the circuit L_g, and the body bias voltage V_{bs}. For the dynamic power $P_{dyn,l}$, C_{eff} is an effective switching capacitance. The energy model (1) is adopted from [31]–[33], where the accuracy of this model has been verified through the SPICE simulations.

B. Problem Formulation

In this section, we consider the problem of deploying dependent IC tasks on the AMP platform, with the aim of maximizing the system QoS under energy and real-time constraints. Therefore, we need to determine: 1) task allocation, 2) frequency assignment, 3) task scheduling, 4) optional cycle adjustment, and 5) task migration. To formulate the problem, we introduce the following binary and continuous variables: 1) $x_{i,k}=1$, if τ'_i is assigned to θ_k, otherwise, $x_{i,k}=0$; 2) $c_{i,l}=1$, if τ'_i is executed with (v_l,f_l), otherwise, $c_{i,l}=0$; 3) $p_{i,j}=1$, if τ'_i proceeds τ'_j, otherwise, $p_{i,j}=0$; 4) ts_i denotes the start time of τ'_i; 5) o_i denotes the optional cycles of τ_i; 6) μ_{2i-1} and μ_{2i} denote the proportions of τ_i executed on different clusters $(\mu_{2i-1}+\mu_{2i}=1)$. Therefore, the execution cycles of τ'_{2i-1} and τ'_{2i} are $(M_i+o_i)\mu_{2i-1}$ and $(M_i+o_i)\mu_{2i}$, respectively. The main symbols used in the problem formulation are summarized in Table II. For the sake of paper presentation, let $\mathcal{N}\triangleq\{1,\dots,N\}$, $\mathcal{N}'\triangleq\{1,\dots,2N\}$, $\mathcal{M_B}\triangleq\{1,\dots,M_B\}$, $\mathcal{M_L}\triangleq\{M_B+1,\dots,M_B+M_L\}$, $\mathcal{M}\triangleq\{\mathcal{M_B},\mathcal{M_L}\}$, $\mathcal{L}_B\triangleq\{1,\dots,l_B\}$, $\mathcal{L}_L\triangleq\{l_B+1,\dots,l_B+l_L\}$, $\mathcal{L}\triangleq\{\mathcal{L}_B,\mathcal{L}_L\}$. The constraint descriptions are as follows:

TABLE II
SYMBOLS USED IN THE PROBLEM FORMULATION

Parameters	
N	number of tasks in $\mathcal{T}$
M_B	number of processors in $\mathcal{P}_B$
M_L	number of processors in $\mathcal{P}_L$
l_B	number of V/F levels in $\mathcal{P}_B$
l_L	number of V/F levels in $\mathcal{P}_L$
H	scheduling horizon
τ_i	the ith task in $\mathcal{T}$
τ_i^m, τ_i^o	mandatory and optional subtasks of τ_i
τ'_{2i-1}, τ'_{2i}	two parts of τ_i executed on different clusters
M_i	mandatory cycles of τ_i
O_i	maximum optional cycles of τ_i
D_i	deadline of task τ_i
θ_k	the kth processor in $\mathcal{P}$
(V_l,f_l)	the lth V/F level in $\{\mathcal{VF}_B,\mathcal{VF}_L\}$
$\gamma_{i,l}$	efficient factor when τ_i is executed with (V_l,f_l)
$q_{i,j}$	$=\begin{cases}1 & \text{if } \tau'_i \text{ is the predecessor of } \tau'_j\\ 0 & \text{else}\end{cases}$
Binary Variables	
$x_{i,k}$	$=\begin{cases}1 & \text{if } \tau'_i \text{ is executed on } \theta_k\\ 0 & \text{else}\end{cases}$
$c_{i,l}$	$=\begin{cases}1 & \text{if } \tau'_i \text{ is executed with } (V_l,f_l)\\ 0 & \text{else}\end{cases}$
$p_{i,j}$	$=\begin{cases}1 & \text{if } \tau'_i \text{ proceeds } \tau'_j\\ 0 & \text{else}\end{cases}$
λ_i	$=\begin{cases}1 & \text{if } \tau'_i \text{ is executed on the processor of } \mathcal{P}_B\\ 0 & \text{if } \tau'_i \text{ is executed on the processor of } \mathcal{P}_L\end{cases}$
Continuous Variables	
o_i	optional cycles of τ_i
μ_{2i-1}, μ_{2i}	workload proportions of τ'_{2i-1} and τ'_{2i}
ts_i	start time of τ'_i

1) Task Allocation Constraints: The task allocation variable $x_{i,k}$ is bounded by

$$\sum\nolimits_{k\in\mathcal{M}} x_{i,k}=1,\ \forall i\in\mathcal{N}', \tag{2}$$

$$x_{2i-1,p}+x_{2i,q}\le 1,\ \forall i\in\mathcal{N},\ \forall p\ne q\in\mathcal{M}_B\ or\ \mathcal{M}_L, \tag{3}$$

where (2) ensures that each task τ'_i, i.e., τ'_{2i-1} or τ'_{2i}, is assigned to a processor, and (3) ensures that τ'_{2i-1} and τ'_{2i}, i.e., two parts of task τ_i, cannot be executed on the different processors in the same cluster, as there is no task migration among the processors in the same cluster.

2) Frequency Assignment Constraints: We consider task-level DVFS, i.e., each processor θ_k uses one V/F level to execute the assigned task τ'_i. For instance, the ARM DynamIQ technology [34] supports such kind of task-level DVFS, which increases the flexibility of DVFS. Since the processors in different clusters are heterogeneous, they have different V/F levels. We need to determine the range of V/F for each task, i.e., $\mathcal{VF}_B$ or $\mathcal{VF}_L$, according to the task allocation variable $x_{i,k}$. Therefore, we introduce an auxiliary (binary) variable λ_i. If $\lambda_i=1$ $(\lambda_i=0)$, task τ'_i is executed on a processor of $\mathcal{P}_B$ $(\mathcal{P}_L)$. Based on the definition of λ_i, we have

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

If $\lambda_i = 1$, the V/F level assigned to task τ_i' is selected from the big cluster V/F set $\mathcal{VF}_B$, otherwise ($\lambda_i = 0$), the V/F level is selected from the LITTLE cluster V/F set $\mathcal{VF}_L$. Hence, we have

$$\lambda_i \sum\nolimits_{l \in \mathcal{L}_B} c_{i,l} + (1-\lambda_i) \sum\nolimits_{l \in \mathcal{L}_L} c_{i,l} = 1,\ \forall i \in \mathcal{N}'. \quad (5)$$

Note that, for the AMP platforms that only support cluster-level DVFS [18], as the processors in the same cluster operate with the same V/F, additional constraints, i.e., $c_{i,m} = c_{j,n}$ ($\forall m \neq n \in \mathcal{P}_B$ *or* $\mathcal{P}_L$), should be added into the problem.

3) Real-time Constraints: When task τ_{2i-1}' is allocated to processor θ_k, and the V/F level (v_l, f_l) is used to execute τ_{2i-1}', the task execution time is $(M_i+o_i)\mu_{2i-1}/(\gamma_{2i-1,l} f_l)$. Since task τ_{2i-1}' is executed before task τ_{2i}', we get

$$te_{2i-1} \leq ts_{2i},\ \forall i \in \mathcal{N}, \quad (6)$$

where

$$te_{2i-1} = ts_{2i-1} + \sum\nolimits_{l \in \mathcal{L}} \frac{c_{2i-1,l}(M_i+o_i)\mu_{2i-1}}{\gamma_{2i-1,l} f_l},\ \forall i \in \mathcal{N}, \quad (7)$$

$$te_{2i} = ts_{2i} + \sum\nolimits_{l \in \mathcal{L}} \frac{c_{2i,l}(M_i+o_i)\mu_{2i}}{\gamma_{2i,l} f_l},\ \forall i \in \mathcal{N}, \quad (8)$$

Since the end time of task τ_i is equal to the end time of task τ_{2i}', and each task τ_i must be completed within the deadline D_i, we have

$$te_{2i} \leq D_i,\ \forall i \in \mathcal{N}. \quad (9)$$

4) Task Non-preemption Constraints: For the tasks without dependency, e.g., τ_i' and τ_j', where $q_{i,j} = 0$, if they are assigned to the same processor, their execution sequence should be determined, as one processor cannot execute multiple tasks at the same time. Therefore, we introduce the following constraints:

$$te_i \leq ts_j + (2 - x_{i,k} - x_{j,k})H + (1 - p_{i,j})H, \quad \forall i \neq j \in \mathcal{N}',\ q_{i,j} = 0,\ \forall k \in \mathcal{M}, \quad (10)$$

$$te_j \leq ts_i + (2 - x_{i,k} - x_{j,k})H + p_{i,j}H, \quad \forall i \neq j \in \mathcal{N}',\ q_{i,j} = 0,\ \forall k \in \mathcal{M}. \quad (11)$$

If τ_i' and τ_j' are executed on the same processor, e.g., θ_k, we have $x_{i,k} = x_{j,k} = 1$, and thus, (10) and (11) are meaningful. If τ_i' is executed before τ_j', i.e., $p_{i,j} = 1$, (10) is relaxed to $te_i \leq ts_j$, which bounds the execution sequence of τ_i' and τ_j'; and (11) is relaxed to $te_i \leq ts_j + H$, which is always satisfied, and thus, it can be ignored. Similarly, if τ_i' is executed after τ_j', i.e., $p_{i,j} = 0$, (10) can be ignored since $te_i \leq ts_j + H$, and (11) is relaxed to $te_j \leq ts_i$.

5) Task Dependency Constraints: For the dependent tasks, e.g., τ_i' and τ_j', where $q_{i,j} = 1$, no matter if they are assigned to the same processor or different processors, the execution sequence among these tasks is fixed. Hence, we have

$$te_i \leq ts_j + (1 - q_{i,j})H,\ \forall i \neq j \in \mathcal{N}', \quad (12)$$

where (12) ensures that if τ_i' proceeds τ_j', i.e., $q_{i,j} = 1$, we have $te_i \leq ts_j$, otherwise, (12) is always satisfied.

6) Energy Constraints: Since the total energy consumed by M processors to execute N tasks during the hyper-period H should not exceed the energy budget E_{buget}, we have

$$\sum\nolimits_{i \in \mathcal{N}} \sum\nolimits_{l \in \mathcal{L}} \left[\frac{c_{2i-1,l}(M_i+o_i)\mu_{2i-1}}{\gamma_{2i-1,l} f_l} + \frac{c_{2i,l}(M_i+o_i)\mu_{2i}}{\gamma_{2i,l} f_l} \right] (P_{sta,l} + P_{dyn,l}) + (M_B + M_L) H P_{on} = E_{total} \leq E_{buget}. \quad (13)$$

7) Primal Problem: Based on the constraints and the objective function (maximizing system QoS) mentioned above, the task deployment problem PP is formulated as follows:

$$\textbf{PP}: \max_{\boldsymbol{x},\boldsymbol{c},\boldsymbol{p},\boldsymbol{o},\boldsymbol{\mu},\boldsymbol{t}} \sum\nolimits_{i \in \mathcal{N}} f_i(o_i) \quad (14)$$

$$\text{s.t.} \begin{cases} (2) - (13), \\ 0 \leq o_i \leq O_i,\ 0 \leq ts_i \leq H,\ \forall i \in \mathcal{N}, \\ \mu_{2i-1} + \mu_{2i} = 1,\ 0 \leq \mu_{2i-1}, \mu_{2i} \leq 1,\ \forall i \in \mathcal{N}, \\ x_{i,k}, c_{i,l}, p_{i,j} \in \{0,1\},\ \forall i \neq j \in \mathcal{N}',\ \forall k \in \mathcal{M},\ \forall l \in \mathcal{L}. \end{cases}$$

where the system QoS function $f_i(o_i) = k_i o_i + R_i$, $\boldsymbol{x} = [x_{i,k}]_{2N \times (M_B+M_L)}$, $\boldsymbol{c} = [c_{i,l}]_{2N \times (l_B+l_L)}$, $\boldsymbol{p} = [p_{i,j}]_{2N \times 2N}$, $\boldsymbol{o} = [o_i]_{1 \times N}$, $\boldsymbol{\mu} = [\mu_i]_{1 \times 2N}$, and $\boldsymbol{t} = [ts_i]_{1 \times 2N}$.

Remark 2.1: Although the number of optional cycles o_i is an integer variable, we consider o_i a continuous variable to simplify the problem. After problem (14) is solved, the result of o_i is rounded down, which does not affect the real-time constraint (9) and the energy constraint (13). The influence of this one-cycle approximation is small since a task is usually executed in hundreds and thousands of cycles.

Considering the overhead of task migration, extra energy E_i^M and time t_i^M should be added to problem (14). First, task end time is updated by $te_{2i-1}' = te_{2i-1} + \alpha_i t_i^M$, where binary variable α_i denotes task τ_i is migrated or not. Then, te_{2i-1} in time constraints (6) and (9)–(12) is replaced by te_{2i-1}'. Finally, energy constraint (13) is rewritten as $E_{total} + \sum_{i \in \mathcal{N}} \alpha_i E_i^M \leq E_{buget}$. Moreover, the proposed approach can be extended to completely heterogeneous platforms by modifying the constraints in problem (14), where we assume that the platforms contain multiple clusters and each cluster has a single core.

Since $x_{i,k}$ and $c_{i,l}$ are binary variables, while o_i, μ_{2i-1} and μ_{2i} are continuous variables, and the nonlinear terms $x_{i,k}c_{i,l}$, $c_{2i-1,l}(M_i+o_i)\mu_{2i-1}$ and $c_{2i,l}(M_i+o_i)\mu_{2i}$ are included in (5)–(13), PP is an MINLP problem.

Theorem 2.1: The QoS-aware IC-task deployment problem based on DVFS (i.e., PP) is $\mathcal{NP}$-hard.

Proof: Please refer to [35] for the details. ■

III. Problem Linearization

To find the optimal solution to PP, we equivalently transform it into an MILP problem according to the following linearization methods.

1) Nonlinear items caused by the products of continuous variables: Since o_i, μ_{2i-1} and μ_{2i} are the continuous variables, the nonlinear items $o_i\mu_{2i-1}$ and $o_i\mu_{2i}$ are hard to linearize directly. Note that $(M_i+o_i)\mu_{2i-1}$ and $(M_i+o_i)\mu_{2i}$ are the execution cycles of task τ_i run on the processors of different clusters. To deal with these nonlinear items, we propose a

linearization method according to the physical meaning of task execution cycles.

Let $\varphi_{2i-1} = (M_i + o_i)\mu_{2i-1}$ and $\varphi_{2i} = (M_i + o_i)\mu_{2i}$, where φ_{2i-1} and φ_{2i} are auxiliary variables. Since $0 \le o_i \le O_i$ and $0 \le \mu_{2i-1} \le 1$, φ_{2i-1} has the maximum value $\varphi_{2i-1,\max} = M_i + O_i$ and the minimum value $\varphi_{2i-1,\min} = 0$. Similarly, we have $\varphi_{2i,\min} = 0 \le \varphi_{2i} \le \varphi_{2i,\max} = M_i + O_i$. Since φ_{2i-1} and φ_{2i} represent the execution cycles of task τ'_{2i-1} and τ'_{2i}, we have $M_i \le \varphi_{2i-1} + \varphi_{2i} \le M_i + O_i$. Therefore, the execution cycles of optional subtask τ_i^o is $o_i = \varphi_{2i-1} + \varphi_{2i} - M_i$, and the QoS function is $\sum_{i\in\mathcal{N}}[k_i(\varphi_{2i-1} + \varphi_{2i} - M_i) + R_i]$.

Taking the auxiliary variables φ_{2i-1} and φ_{2i} into account, the end times of task τ'_{2i-1} and τ'_{2i}, i.e., (7) and (8), have the form

$$te_{2i-1} = ts_{2i-1} + \sum\nolimits_{l\in\mathcal{L}} \frac{c_{2i-1,l}\varphi_{2i-1}}{\gamma_{2i-1,l} f_l},\ \forall i \in \mathcal{N}, \tag{15}$$

$$te_{2i} = ts_{2i} + \sum\nolimits_{l\in\mathcal{L}} \frac{c_{2i,l}\varphi_{2i}}{\gamma_{2i,l} f_l},\ \forall i \in \mathcal{N}. \tag{16}$$

In addition, the energy constraint (13) can be rewritten as

$$\sum\nolimits_{i\in\mathcal{N}} \sum\nolimits_{l\in\mathcal{L}} \left(\frac{c_{2i-1,l}\varphi_{2i-1}}{\gamma_{2i-1,l} f_l} + \frac{c_{2i,l}\varphi_{2i}}{\gamma_{2i,l} f_l} \right) (P_{sta,l} + P_{dyn,l}) + (M_B + M_L) H P_{on} \le E_{buget}. \tag{17}$$

2) Nonlinear items caused by the products of continuous and binary variables: Since $c_{2i-1,l}$ is a binary variable, while φ_{2i-1} is a continuous variable, to deal with the nonlinear items $c_{2i-1,l}\varphi_{2i-1}$ and $c_{2i,l}\varphi_{2i}$ in (15), (16) and (17), we introduce the following lemma.

Lemma 3.1: Assume that the constant $s_1, s_2 > 0$ and two constraint spaces $P_1 = \{[t, b, x] | t = bx, -s_1 \le x \le s_2, b \in \{0, 1\}\}$ and $P_2 = \{[t, b, x] | -bs_1 \le t \le bs_2, t + bs_1 - x - s_1 \le 0, t - bs_2 - x + s_2 \ge 0, b \in \{0, 1\}\}$, then $P_1 \rightleftharpoons P_2$.

Proof: $P_1 \Rightarrow P_2$. Since $t = bx$ and $-s_1 \le x \le s_2$, we have $-bs_1 \le t \le bs_2$. According to $-s_1 \le x \le s_2$ and $b \in \{0, 1\}$, we get $(b - 1)(x - s_2) \ge 0$ and $(b - 1)(x + s_1) \le 0$. Then, $t + bs_1 - x - s_1 \le 0$ and $t - bs_2 - x + s_2 \ge 0$ hold. $P_2 \Rightarrow P_1$. If $b = 0$, we have $t = 0$ and $-s_1 \le x \le s_2$. If $b = 1$, we get $-s_1 \le t = x \le s_2$. Thus, $P_1 \rightleftharpoons P_2$. ■

According to *Lemma* 3.1, two auxiliary (continue) variables $\Phi_{2i-1,l}$ and $\Phi_{2i,l}$ are introduced to replace the nonlinear terms $c_{2i-1,l}\varphi_{2i-1}$ and $c_{2i,l}\varphi_{2i}$. Since $\varphi_{2i-1,\min} \le \varphi_{2i-1} \le \varphi_{2i-1,\max}$ and $\varphi_{2i,\min} \le \varphi_{2i} \le \varphi_{2i,\max}$, we add the following constraints into PP:

$$c_{i,l}\varphi_{i,\min} \le \Phi_{i,l} \le c_{i,l}\varphi_{i,\max},\ \forall i \in \mathcal{N}',\ \forall l \in \mathcal{L}, \tag{18}$$

$$\Phi_{i,l} - c_{i,l}\varphi_{i,\min} - \varphi_i + \varphi_{i,\min} \le 0,\ \forall i \in \mathcal{N}',\ \forall l \in \mathcal{L}, \tag{19}$$

$$\Phi_{i,l} - c_{i,l}\varphi_{i,\max} - \varphi_i + \varphi_{i,\max} \ge 0,\ \forall i \in \mathcal{N}',\ \forall l \in \mathcal{L}. \tag{20}$$

Substituting $\Phi_{2i-1,l} = c_{2i-1,l}\varphi_{2i-1}$ and $\Phi_{2i,l} = c_{2i,l}\varphi_{2i}$ into (15), (16) and (17), we have

$$te_{2i-1} = ts_{2i-1} + \sum\nolimits_{l\in\mathcal{L}} \frac{\Phi_{2i-1,l}}{\gamma_{2i-1,l} f_l}, \tag{21}$$

$$te_{2i} = ts_{2i} + \sum\nolimits_{l\in\mathcal{L}} \frac{\Phi_{2i,l}}{\gamma_{2i,l} f_l}, \tag{22}$$

$$\sum\nolimits_{i\in\mathcal{N}} \sum\nolimits_{l\in\mathcal{L}} \left(\frac{\Phi_{2i-1,l}}{\gamma_{2i-1,l} f_l} + \frac{\Phi_{2i,l}}{\gamma_{2i,l} f_l} \right) (P_{sta,l} + P_{dyn,l}) + (M_B + M_L) H P_{on} \le E_{buget}. \tag{23}$$

3) Nonlinear items caused by the products of binary variables: For the nonlinear item $x_{i,k}c_{i,l}$ in (4) and (5), since $x_{i,k}$ and $c_{i,l}$ are binary variables, we propose the following lemma to deal with this nonlinear item.

Lemma 3.2: Assume that x_1 and x_2 are the binary variables. The nonlinear term x_1x_2 can be replaced by an auxiliary (binary) variable y, where $y = x_1x_2$, and the additional constraints $y \le x_1$, $y \le x_2$ and $y \ge x_1 + x_2 - 1$.

Proof: When binary variables $x_1 = 1$ and $x_2 = 1$, the additional constraint is transformed to $1 \le y \le 1$, and thus, $y = 1$ holds. Similarly, for the cases 1) $x_1 = 0$ and $x_2 = 0$; 2) $x_1 = 1$ and $x_2 = 0$; 3) $x_1 = 0$ and $x_2 = 1$, we have $y = x_1x_2 = 0$. ■

Based on *Lemma* 3.2, we introduce an auxiliary (binary) variable $z_{i,k,l}$, where $z_{i,k,l} = x_{i,k}c_{i,l}$, and add the following constraints into PP.

$$z_{i,k,l} \le x_{i,k},\ z_{i,k,l} \le c_{i,l},\ z_{i,k,l} \ge x_{i,k} + c_{i,l} - 1,\quad \forall i \in \mathcal{N}',\ \forall k \in \mathcal{M},\ \forall l \in \mathcal{L}, \tag{24}$$

On this basis, substituting (4) into (5) and recalling that $\lambda_i c_{i,l} = \sum_{k\in\mathcal{M}_B} x_{i,k}c_{i,l} = \sum_{k\in\mathcal{M}_B} z_{i,k,l}$, (5) can be linearized as follows:

$$\sum\nolimits_{k\in\mathcal{M}_B} \left(\sum\nolimits_{l\in\mathcal{L}_B} z_{i,k,l} - \sum\nolimits_{l\in\mathcal{L}_L} z_{i,k,l} \right) + \sum\nolimits_{l\in\mathcal{L}} c_{i,l} = 1,\quad \forall i \in \mathcal{N}'. \tag{25}$$

4) MILP formulation: Based on the linearization methods mentioned above, PP can be transformed into the following problem:

$$\textbf{PP1}:\ \max_{\substack{\boldsymbol{x},\boldsymbol{c},\boldsymbol{p},\boldsymbol{t},\\ \boldsymbol{\varphi},\boldsymbol{\Phi},\boldsymbol{z}}} \sum\nolimits_{i\in\mathcal{N}} [k_i(\varphi_{2i-1} + \varphi_{2i} - M_i) + R_i] \tag{26}$$

$$\text{s.t.} \begin{cases} (2), (3), (6), (9) - (12), (18) - (25), \\ 0 \le ts_i \le H,\ \forall i \in \mathcal{N}, \\ M_i \le \varphi_{2i-1} + \varphi_{2i} \le M_i + O_i,\ \forall i \in \mathcal{N}, \\ 0 \le \varphi_{2i-1}, \varphi_{2i} \le M_i + O_i,\ \forall i \in \mathcal{N}, \\ x_{i,k}, c_{i,l}, p_{i,j}, z_{i,k,l} \in \{0, 1\}, \forall i \ne j \in \mathcal{N}', \forall k \in \mathcal{M}, \forall l \in \mathcal{L}. \end{cases}$$

where $\boldsymbol{\varphi} = [\varphi_i]_{1\times 2N}$, $\boldsymbol{\Phi} = [\Phi_{i,l}]_{2N\times(l_B+l_L)}$, and $\boldsymbol{z} = [z_{i,k,l}]_{2N\times(M_B+M_L)\times(l_B+l_L)}$.

Remark 3.1: PP1 is an MILP problem, as the binary and continuous variables are coupled linearly. Hence, it is much easier to solve the MILP-based PP1 than the MINLP-based PP. Furthermore, *Lemma* 3.1 and *Lemma* 3.2 imply that the variable replacement will not change the feasible region of the problem. In addition, the objective functions of PP and PP1 are the same. Hence, PP1 is equivalent to PP (i.e., the optimal objective function values of PP and PP1 are the same).

IV. Heuristic Task Deployment Algorithm

Since the MINLP-based PP is linearized to the MILP-based PP1, the optimal solution to PP1 can be found by the existing solver, e.g., CPLEX or Gurobi [9]. However, finding the optimal

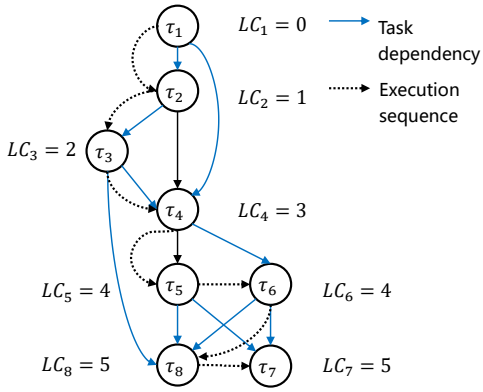


Fig. 3. Example of task layer classification.

solution is still time-consuming, especially when the size of the problem is large. The complexity of an optimization problem is related to the number of variables and constraints. Based on the idea of problem decomposition, we propose a novel heuristic method for solving PP to improve the scalability of the proposed method. The reasons for solving PP rather than PP1 are that 1) these two problems are equivalent, but the problem size of PP is smaller than PP1; and 2) the heuristic method considers the variables of PP in sequence, and thus, the coupling among variables can be avoided. According to the relationship among variables, we decompose PP into the following sub-problems and solve them in sequence: 1) Frequency Assignment (SP1), 2) Task Allocation and Task Scheduling (SP2), and 3) Optional Cycle Adjustment (SP3).

A. Frequency Assignment

The task deployment problem PP is restricted by the time and energy constraints, which are influenced by the V/F level (V_l, f_l) used to execute tasks. Hence, we consider frequency assignment c_{il} at the first step. Note that the number of cycles of the optional subtask o_i is coupled nonlinearly with the portion of task migration μ_i in (13), i.e., $c_{2i-1,l}(M_i + o_i)\mu_{2i-1}$ and $c_{2i,l}(M_i + o_i)\mu_{2i}$, which makes the problem difficult to solve. We consider task migration is performed between mandatory subtask τ_i^m and optional subtask τ_i^o. Therefore, τ_i^m and τ_i^o are equivalent to τ'_{2i-1} and τ'_{2i}, i.e., $M_i = (M_i + o_i)\mu_{2i-1}$ and $o_i = (M_i + o_i)\mu_{2i}$.

Since the subtask τ_i^m must be executed, while the subtask τ_i^o is optional and the execution cycles of τ_i^o can be adjusted within the range $0 \le o_i \le O_i$, we consider the frequency assignment of mandatory subtask τ_i^m firstly and omit optional subtask τ_i^o. Hence, we set $o_i = 0$ at the current step. When (V_l, f_l) is used to execute a subtask τ_i^m with M_i cycles, the task execution time and energy are

$$t_{2i-1,l} = \frac{M_i}{\gamma_{2i-1,l} f_l}, \ \forall i \in \mathcal{N}, \tag{27}$$

$$E_{2i-1,l} = t_{2i-1,l}(P_{sta,l} + P_{dyn,l}), \ \forall i \in \mathcal{N}. \tag{28}$$

Note that the QoS function is given by $\sum_{i\in\mathcal{N}}(k_i o_i + R_i)$. The more optional cycles are executed, the higher system QoS is generated, and the more energy is consumed for the task execution. Since the system energy budget E_{buget} is limited, to increase the number of optional cycles at the next step, SP1 aims to minimize the execution energy of mandatory subtasks.

Algorithm 1: Frequency Assignment

Input : Parameters in Table II, CPT
Output : Frequency assignment $\boldsymbol{c}$
Initialization: $c_{2i-1,l} = -1$, $E_{total} = 0$, $t_{cp} = 0$, $E_{min} = \infty$

1 **for** $i \leftarrow 1$ **to** N **do**
2 **for** $l \leftarrow 1$ **to** $(l_B + l_L)$ **do**
3 Calculate $t_{2i-1,l}$ and $E_{2i-1,l}$ through (27) and (28);
4 **if** $E_{total} + E_{2i-1,l} \le E_{buget}$ *and* $t_{2i-1,l} + t_{cp} \le D_i$ $(i \in CPT)$ **then**
5 **if** $E_{min} > E_{total} + E_{2i-1,l}$ **then**
6 $E_{min} = E_{total} + E_{2i-1,l}$, $l^* = l$;
7 **end**
8 **end**
9 **end**
10 **if** $E_{min} \le E_{buget}$ **then**
11 $E_{total} = E_{min}$, $E_{min} = \infty$, $c_{2i-1,l^*} = 1$;
12 $t_{cp} = t_{cp} + t_{2i-1,l^*} (i \in CPT)$;
13 **else**
14 **Stop.**
15 **end**
16 **end**

With the fixed task execution cycles, minimizing task execution energy will increase task execution time, as a lower V/F level may be prone to use. To avoid missing the task deadline, the real-time constraint should be considered. Since the allocation of mandatory subtask τ_i^m, i.e., $x_{2i-1,k}$, is unknown at the current step, we focus on the real-time constraints for the tasks on the critical path. Let CPT denote the index set of the tasks on the critical path. For example in Fig. 3, we have $CPT = \{1, 2, 3, 4, 5, 8\}$ for the critical path of consisting of tasks τ_1, τ_2, τ_3, τ_4, τ_5, and τ_8. Therefore, the frequency assignment problem is formulated as follows:

$$\textbf{SP1}: \min_{\boldsymbol{c}} E_{total},$$
$$\text{s.t.} \begin{cases} \sum_{l\in\mathcal{L}} c_{2i-1,l} = 1, \ \forall i \in \mathcal{N}, \\ \sum_{l\in\mathcal{L}} c_{2i-1,l} t^e_{2i-1,l} \le D_i, \ \forall i \in CPT, \\ E_{total} \le E_{buget}. \end{cases}$$

where $E_{total} = \sum_{i\in\mathcal{N}} \sum_{l\in\mathcal{L}} c_{2i-1,l} E_{2i-1,l} + (M_B + M_L) HP_{on}$ is the total energy required to execute all the mandatory subtasks. $t^e_{2i-1,l}$ is the end time of subtask τ_i^m, and we assume that the tasks in the set CPT are executed in sequence. Thus, the start time of τ_i^m equals the end time of τ_i^m's predecessor.

Since SP1 is an ILP-based problem, based on the Greedy algorithm, we propose Algorithm 1 to solve SP1. For the sake of presentation, frequency assignment variable is initialized as $c_{2i-1,l} = -1$. Algorithm 1 firstly finds a proper frequency for each mandatory subtask τ_i^m, under the real-time and energy constraints (Line 4). The selected frequency should cause the minimum increase of task execution energy among the tasks that have already been assigned the frequency (Line 1–9). During this process, if the frequency assignment, e.g., $c_{2i-1,l} = 1$, violates the real-time and energy constraints, the assignment $c_{2i-1,l} = 1$ will be excluded (Line 10–15). Applying the above method for each mandatory subtask τ_i^m, the solution $c_{2i-1,l}$ to the SP1 can be found. In Algorithm 1, the inner loop l runs

$l_B + l_L$ times for each outer loop i. Since the outer loop i runs N times, the time complexity is $O(N(l_B + l_L))$.

B. Task Allocation and Task Scheduling

At this step, we mainly consider the deployment of mandatory subtasks. Substituting frequency assignment $c_{2i-1,l}$, i.e., the solution to SP1, into (27) and (28), the execution time t_{2i-1} and energy E_{2i-1} of each mandatory subtask τ_i^m can be determined. According to task allocation $x_{2i-1,k}$, the time and energy of each processor used for executing tasks can be adjusted. With $x_{2i-1,k}$ and t_{2i-1}, the task execution time of processor θ_k is $tp_k = \sum_{i\in\mathcal{N}} x_{2i-1,k} t_{2i-1}$. In SP2, we aim to balance the workloads of the processors, i.e., $\min(\max_{k\in\mathcal{M}}\{tp_k\})$. By doing so, we can increase the idle time of each processor, i.e., $H - tp_k$. This idle time can be used to execute the optional subtasks, which is the main purpose of the next step.

Since the tasks are dependent, the allocation and the scheduling of these tasks should be considered concurrently. Therefore, the constraints regarding task non-preemptive (10)–(11) and task dependency (12) should also be taken into account. Hence, the task allocation and scheduling problem has the form:

$$\textbf{SP2}: \min_{\boldsymbol{x},\boldsymbol{p},\boldsymbol{t}} \left(\max_{k\in\mathcal{M}} \{tp_k\} \right) \tag{29}$$

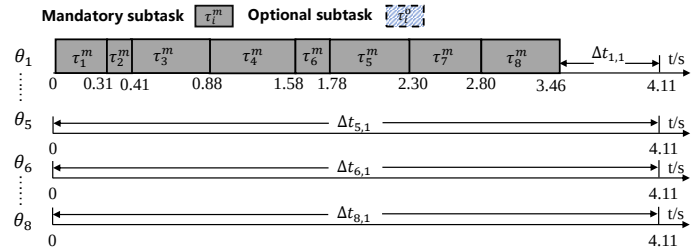
$$\text{s.t.} \begin{cases} (10)-(12), \\ \lambda_{2i-1} \sum_{k\in\mathcal{M}_\mathcal{B}} x_{2i-1,k} + (1-\lambda_{2i-1}) \sum_{k\in\mathcal{M}_\mathcal{L}} x_{2i-1,k} = 1, \\ 0 \le ts_{2i-1} + t_{2i-1} \le D_i, \ \forall i \in \mathcal{N}. \end{cases}$$

Under the given frequency assignment decision $c_{2i-1,l}$, we can determine the value of λ_{2i-1} according to (5). If $\lambda_{2i-1} = 1$, we have $\sum_{k\in\mathcal{M}_\mathcal{B}} x_{2i-1,k} = 1$, which implies that task τ_{2i-1} is allocated to the big cluster $\mathcal{P}_B$, otherwise (i.e., $\lambda_{2i-1} = 0$), we have $\sum_{k\in\mathcal{M}_\mathcal{L}} x_{2i-1,k} = 1$, and thus, task τ_{2i-1} is allocated to the LITTLE cluster $\mathcal{P}_L$.

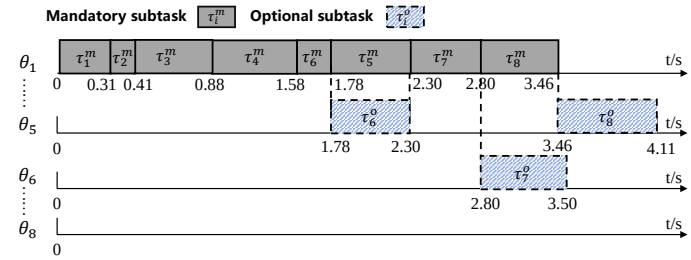
Since ts_{2i-1} is a continuous variable, while $x_{2i-1,k}$ and p_{ij} are the binary variables, SP2 is an MILP problem. Based on the problem structure, a three-step heuristic method is proposed to solve SP2. Firstly, we determine the task sequence to perform task allocation according to the dependencies among the tasks. Then, we determine the start time of the tasks. Finally, we adjust the frequency assignment as the above process may violate the real-time constraints. Based on this idea, the proposed heuristic contains three steps: 1) task layer classification, 2) task allocation and scheduling, and 3) frequency adjustment.

1) Task Layer Classification: Since the tasks are dependent, we introduce a sequence Seq to perform task allocation. To determine Seq, we set a sequence index LC_i for each task τ_i.

a. For each entry task τ_i, we assume that its sequence index is 0. In Fig. 3, we have $LC_1 = 0$ as τ_1 is an entry task.
b. For each non-entry task τ_j, we find all its predecessors, e.g., task τ_i with $q_{i,j} = 1$. We denote the index of predecessor τ_i as LC_i and let $LC_j = \max_{i\in\mathcal{N}}\{LC_i\} + 1$. In Fig. 3, we have $LC_2 = 1$, $LC_3 = 2$, $LC_4 = 3$, $LC_5 = 4$, $LC_6 = 4$, $LC_7 = 5$ and $LC_8 = 5$.
c. We sort all the tasks in ascending order according to their indices, and the tasks with the same index are sorted by their execution time in ascending order. Therefore, we obtain a task sequence Seq. In Fig. 3, if the execution time of τ_5 (τ_8) is less than τ_6 (τ_7), we have $Seq = \{1, 2, 3, 4, 5, 6, 8, 7\}$.



(a) Scheduling of mandatory subtasks.



(b) Scheduling of mandatory and optional subtasks.

Fig. 4. Example of heuristic task deployment method.

2) Task Allocation and Task Scheduling: Based on the task sequence Seq, we perform task allocation to balance the workloads of the processors. By using the sequence Seq, task allocation $x_{i,k}$ and task start time ts_i can be considered concurrently.

a. The allocation of mandatory subtask τ_i^m is influenced by the frequency assignment $c_{2i-1,l}$. According to (5), the value of λ_{2i-1} can be determined under the given $c_{2i-1,l}$. If $\lambda_{2i-1} = 1$, τ_i^m is assigned to the big cluster, otherwise (i.e., $\lambda_{2i-1} = 0$), τ_i^m is assigned to the LITTLE cluster.
b. On the one hand, as we follow the decisions Seq and $\boldsymbol{c}$ to allocate and execute the tasks, the end time of τ_i^m's predecessors is known. Based on the task dependency p_{ij}, the feasible start time of subtask τ_i^m is $\max_{i\in Pre}\{te_i\}$, where Pre is the set of τ_i^m's predecessors, and te_i is the end time of corresponding task. As the DAG example shown in Fig. 3, we have $Pre = \{2, 3\}$ for τ_4^m. If we set $te_2 = 0.41$ s and $te_3 = 0.88$ s, the feasible task start time of τ_4^m is $\max\{te_2, te_3\} = 0.88$ s, as shown in Fig. 4(a).
c. On the other hand, when dealing with τ_i^m, the tasks before τ_i^m in the sequence Seq have been allocated. Therefore, based on the end time of these tasks, the task execution time of each processor, i.e., tp_k, is known. To obtain a longer time interval to execute the optional subtasks, the feasible start time of mandatory subtask τ_i^m can be set to $\max_{k\in\mathcal{M}}\{tp_k\}$. Therefore, taking the feasible start time from the perspectives of tasks and processors into account, τ_i^m is assigned to the processor with the latest feasible start time, i.e., $\max_{k\in\mathcal{M}}\{\max_{i\in Pre}\{te_i\}, tp_k\}$, under the real-time constraints. As the example shown in Fig. 4(a), subtask τ_4^m is allocated to processor θ_1, due to $\max\{\max\{te_2, te_3\} = 0.88, tp_1 = 0.88, tp_2 = 0, \ldots, tp_8 = 0\} = 0.88$ s.

3) Frequency Adjustment: If the real-time constraints cannot be satisfied, we adjust the frequency assignment $c_{2i-1,l}$ accord-

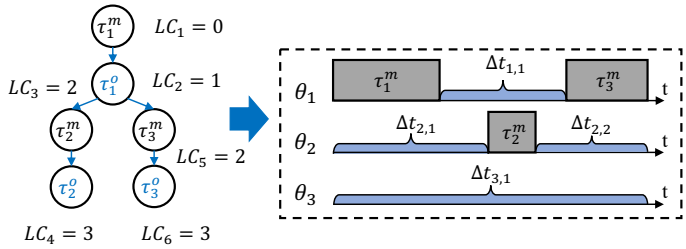


Fig. 5. Four cases to determine the potential optional subtasks.

ingly.

a. We record the index of subtask τ_i^m that violates the real-time constraint, e.g., $te_{2i-1} > D_i$.
b. On the one hand, under the energy constraints, i.e., $E_{total} \leq E_{budget}$, to reduce the end time of subtask τ_i^m, we increase the V/F levels of τ_i^m and its predecessors, e.g., $c_{2i-1,l} = 1 \rightarrow c_{2i-1,l+1} = 1$ and $c_{2j-1,l} = 1 \rightarrow c_{2j-1,l+1} = 1$ ($\forall j \neq i \in \mathcal{N}$, $q_{j,i} = 1$). On the other hand, to balance the energy consumed for task execution, we reduce the V/F levels of τ_i^m's successors, e.g., $c_{2k-1,l} = 1 \rightarrow c_{2k-1,l-1} = 1$ ($\forall k \neq i \in N$, $q_{i,k} = 1$).
c. With the updated V/F assignment, the above task allocation process is performed again for subtask τ_i^m.
d. If the real-time constraints are still not satisfied, the above method is repeated until all the tasks in the sequence Seq have adjusted their V/F levels.

C. Optional Cycle Adjustment

Based on the solutions of the SP1 and SP2, we obtain the deployment results of mandatory subtasks, the remaining energy $E_{optl} = E_{budget} - E_{total}$, and the ith idle time interval of processor θ_k: $\Delta t_{k,i} = t_{k,i}^e - t_{k,i}^s$, where $t_{k,i}^s$ and $t_{k,i}^e$ are the start time and the end time of $\Delta t_{k,i}$, respectively. In this step, we aim to optimize the optional execution cycles, by using the idle time interval $\Delta t_{k,i}$ and the remaining energy E_{optl}, in order to improve the system QoS. The details are as follows.

a. According to the allocation and scheduling of each mandatory subtask τ_i^m, i.e., x_{2i-1}, ts_{2i-1} and te_{2i-1}, we can calculate the idle time interval of each processor, i.e., $\Delta t_{k,i}$. For example, in Fig. 4(a), we have $\Delta t_{1,1} = 0.65$ s and $\Delta t_{5,1} = 4.11$ s.
b. Since (27) and (28) are the time and energy required to execute a task with M_i cycles using the V/F level (V_l, f_l), $TC_l = \frac{1}{\gamma_{i,l} f_l}$ and $EC_l = \frac{P_{sta,l} + P_{dyn,l}}{\gamma_{i,l} f_l}$ are the time and energy required to execute a task with one cycle. Thus, under the given the time $\Delta t_{k,i}$ and energy E_{optl}, we can execute a task with $\frac{\Delta t_{k,i}}{TC_l}$ cycles and $\frac{E_{optl}}{EC_l}$ cycles, respectively. Note that the processors are heterogeneous. If $\theta_k \in \mathcal{P}_{\mathcal{B}}$, the range of V/F level is $\mathcal{L}' = \mathcal{L}_B$, otherwise (i.e., $\theta_k \in \mathcal{P}_{\mathcal{L}}$), $\mathcal{L}' = \mathcal{L}_L$. Based on the type of processor θ_k, the maximum optional cycles are $\Delta o_{\max} = \max_{l \in \mathcal{L}'}\{\min\{\frac{\Delta t_{k,i}}{TC_l}, \frac{E_{optl}}{EC_l}\}\}$, and we assume that the V/F level to achieve $\Delta o_{\max}$ is (V^*, f^*). Substituting (V^*, f^*) into TC_l and EC_l, we obtain TC_{l^*} and EC_{l^*}, i.e., the optimal computation cost to achieve a longer task execution cycles.
c. Based on the task sequence Seq, we determine the potential optional subtasks executed in each idle time interval $\Delta t_{k,i}$.

TABLE III
THE EXAMPLE OF OPTIONAL CYCLE ADJUSTMENT.

Task	O_j (cycles)	$\Delta O_{\max}$ (cycles)	$\frac{te_{fea,j} - ts_{fea,j}}{TC_{l^*}}$	o_j (cycles)
τ_1^o	2.67×10^8	1.65×10^9	$\frac{0.31-0.31}{TC_{12}}$	0
τ_2^o	5.84×10^8		$\frac{0.41-0.41}{TC_{12}}$	0
τ_3^o	4.13×10^8		$\frac{0.88-0.88}{TC_{12}}$	0
τ_4^o	4.94×10^8		$\frac{1.58-1.58}{TC_{12}}$	0
τ_5^o	3.11×10^8		$\frac{2.30-2.30}{TC_{12}}$	0
τ_6^o	4.64×10^8		$\frac{2.30-1.78}{TC_{12}}$	1.24×10^8
τ_8^o	5.63×10^8	1.53×10^9	$\frac{2.30-1.78}{TC_{12}}$	1.55×10^8
τ_7^o	2.74×10^8	1.37×10^9	$ts_{fea,8} = 4.11$ s $te_{fea,8} = D_8 = 3.5$ s	0

Let τ_p^m and τ_q^m denote the nearest dependent tasks allocated to θ_k before and after time interval $\Delta t_{k,i}$, respectively. As the example shown in Fig. 5, the nearest dependent tasks allocated to θ_1 before and after the time interval $\Delta t_{1,1}$ are τ_1^m and τ_3^m. Note that the index of τ_p^m is LC_{2p-1}, while the index of τ_q^m is LC_{2q-1}. In addition, τ_p^m and τ_q^m may not always exist. We have the following four cases:

- If τ_p^m and τ_q^m exist and they are allocated to θ_k, we have $T = \{\tau_j^o | LC_{2p-1} \leq LC_{2j} \leq LC_{2q-1}, \forall j \in \mathcal{N}\}$, where T is the candidate set of optional subtasks executed during $\Delta t_{k,i}$, e.g., for $\Delta t_{1,1}$, as τ_1^m is the predecessor of τ_3^m, the candidate task set $T = \{\tau_j^o | LC_1 = 0 \leq LC_{2j} \leq LC_5 = 2, \forall j \in \mathcal{N}\} = \{\tau_1^o\}$.
- If τ_p^m is allocated to θ_k and τ_q^m doesn't exist, we have $T = \{\tau_j^o | LC_{2j} \geq LC_{2p-1}, \forall j \in \mathcal{N}\}$, e.g., for $\Delta t_{2,2}$, as τ_2^m and τ_3^m are independent, the candidate task set $T = \{\tau_j^o | LC_{2j} \geq LC_3 = 2, \forall j \in \mathcal{N}\} = \{\tau_2^o, \tau_3^o\}$.
- If τ_q^m is allocated to θ_k and τ_p^m doesn't exist, we have $T = \{\tau_j^o | LC_{2j} \leq LC_{2q-1}, \forall j \in \mathcal{N}\}$, e.g., for $\Delta t_{2,1}$, as τ_1^m is the predecessor of τ_2^m, the candidate task set $T = \{\tau_j^o | LC_{2j} \leq LC_3 = 2, \forall j \in \mathcal{N}\} = \{\tau_1^o\}$.
- If τ_p^m and τ_q^m are not exist, we assume that T contains all the optional subtasks, e.g., for $\Delta t_{3,1}$, the candidate task set is $T = \{\tau_1^o, \tau_2^o, \tau_3^o\}$.

Since no task migration is performed among the processors in the same cluster, we remove the optional subtask τ_j^o from the set T if τ_j^o and its mandatory subtask τ_j^m are executed on different processors in the same cluster. Based on the above method, for the idle interval $\Delta t_{5,1}$ in Fig. 4(a), the feasible task set $T = \{\tau_1^o, \tau_2^o, \tau_3^o, \tau_4^o, \tau_5^o, \tau_6^o, \tau_7^o, \tau_8^o\}$. As the tasks in the set T are dependent, we follow the task sequence Seq to determine the execution cycles of each optional subtask τ_j^o in the set T.

d. Let $ts_{fea,j} = \max\{t_{k,i}^s, \{te_s\}\}$ denote the feasible start time of τ_j^o in $\Delta t_{k,i}$, where $\{te_s\}$ is the end time set of τ_j^o's predecessors. In addition, let $te_{fea,j} = \min\{t_{k,i}^e, D_j, \{ts_t\}\}$ denote the feasible end time of τ_j^o in $\Delta t_{k,i}$, where $\{ts_t\}$ is the start time set of τ_j^o's successors. Note that only when the feasible start time $ts_{fea,j}$ is smaller than the feasible end time $te_{fea,j}$, i.e., $ts_{fea,j} \leq te_{fea,j}$, the optional subtask τ_j^o exists. Taking the optimal V/F to achieve longer task execution cycles into account, i.e., (V^*, f^*), the number of execution cycles can be executed during the time slot $te_{fea,j} - ts_{fea,j}$ is $\frac{te_{fea,j} - ts_{fea,j}}{TC_{l^*}}$. On this basis, with the

consideration of the maximum execution cycles O_j of τ_j and the upper bound $\Delta o_{\max}$ of execution cycles in $\Delta t_{k,i}$, the number of execution cycles for optional subtask τ_j^o can be set to $o_j = \min\{O_j, \Delta o_{\max}, \frac{te_{fea,j}-ts_{fea,j}}{TC_{l^*}}\}$. As the example in Fig. 4(b), during the idle time interval $\Delta t_{5,1}$, we have $\Delta o_{\max} = \frac{4.11-0}{TC_{12}} = 1.65 \times 10^9$ (cycles). Based on the method mentioned above, we can obtain the cycles of τ_6^o and τ_8^o, and these tasks are allocated to θ_5, the adjustment process of optional cycles is shown in Table III.

e. We apply the above method for each $\Delta t_{i,k}$ and the adjustment of optional subtasks ends until $E_{optl} = 0$ or $o_i = O_i$ ($\forall i \in \mathcal{N}$). Therefore, the number of execution cycles for τ_7^o is $\min\{O_7, \Delta o_{\max}, \frac{3.40-2.80}{TC_{12}}\} = 1.69 \times 10^8$ (cycles) and τ_7^o is allocated to θ_6.

Based on the above sub-problems, the frequency allocation variable $c_{i,l}$, the task allocation variable $x_{i,k}$, the task start time ts_i, and the optional execution cycles o_i can be determined. Thus, we can obtain the task deployment solution to maximize system QoS under real-time and energy constraints.

D. Time Complexity

To solve the task deployment problem, the proposed heuristic method divides it into three sub-problems: SP1, SP2, and SP3, and solves these sub-problems in sequence. For SP1, the time complexity is $O(N(l_B + l_L))$. In SP2, we have three stages. Firstly, the recursive manner is adopted to decide the task layer. As the height of the top-down recursive tree is $log(N)$, the time complexity is $O(Nlog(N))$. Then, we use the greedy method to determine task allocation and scheduling. During this process, the outer loop i runs N times, while the inner loop k runs M times. Therefore, the time complexity is $O(NM)$. Finally, during the frequency adjustment process, as the maximum number of adjustments is N, the time complexity is $O(N)$. For SP3, we consider the worst case, where the maximum number of idle time intervals is $M + N$. Therefore, the calculation of feasible optional cycles is invoked $O(M+N)$ times. As this calculation takes $O(N)$ times, the time complexity is $O(N(M + N))$. To sum up, the total time complexity of the heuristic method is $O(N(l_B + l_L) + Nlog(N) + NM + N + N(M + N))$. Note that N denotes the total number of tasks, M denotes the number of processors, and $l_B + l_L$ denotes the number of V/F levels. In addition, we have $N \gg M$ and $N \gg l_B + l_L$. Therefore, the total time complexity is $O(Nlog(N) + N^2)$.

V. Simulation Results

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

1) Simulation Setup: We consider the AMP platform inspired by the ARM DynamIQ big.LITTLE with per-core DVFS technology as a case study. The power parameters are adopted from [34] and [28]. The number of processors $M_B = 4$ and $M_L = 4$, and the number of V/F levels $l_B = 9$ and $l_L = 5$. The execution efficient factor $\gamma_{i,l}$ is within the range $[0.4, 1]$ [36], which demonstrates the heterogeneity among the processors. For each task τ_i, the cycles of the mandatory subtask τ_i^m and the maximum cycles of the optional subtask τ_i^o, i.e., M_i and O_i, are assumed to be in the range of $[4 \times 10^7, 6 \times 10^8]$ [20], which is given by the practical applications, such as M-JPEG2000. For the QoS function, we use a T2-coder's linear QoS model $f_i(o_i) = 0.0313o_i - 9.296$ [29]. Under the given task number N, we randomly generate a DAG to describe the task dependency. The parameters regarding the IC tasks and the AMP platform are summarized in Table IV. β and δ are the tuned parameters. $D_{\max}$ denotes the time required to execute the tasks on the critical path (CPT) of DAG with the minimum frequency, while $D_{\min}$ denotes the time required to execute task τ_i with the maximum frequency. In addition, we set $H = D_{\max}$ as all the tasks should be executed within the scheduling horizon H. Note that the values of these parameters do not affect the structure of the task deployment problem. Thus, the proposed method is still applicable to other system parameters.

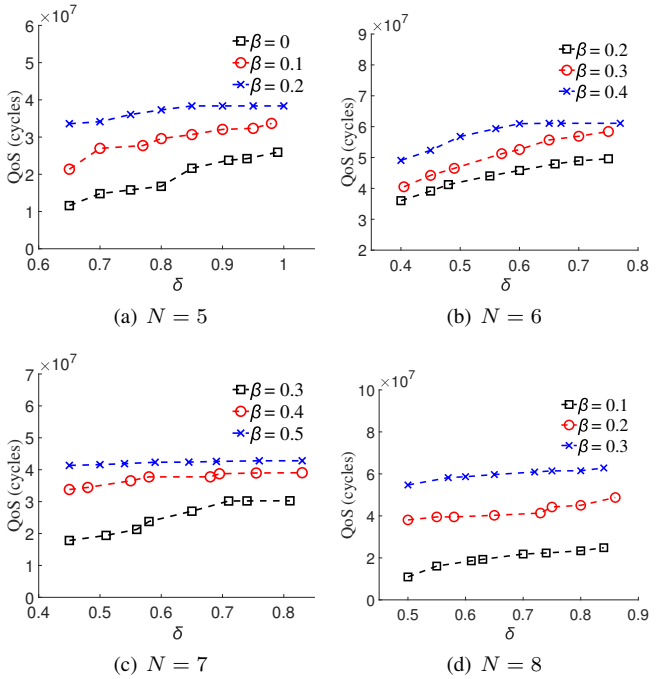
TABLE IV
Experimental set-up

Task Model	
$M_i, O_i \in [4 \times 10^7, 6 \times 10^8]$	$\gamma_{i,l} \in [0.4, 1]$
$D_{\min} = \min_{l \in \mathcal{L}}\{\frac{M_i+O_i}{\gamma_{i,l} f_l}\}$	$D_{\max} = \max_{l \in \mathcal{L}}\{\sum_{i \in \mathcal{CPT}} \frac{M_i+O_i}{\gamma_{i,l} f_l}\}$
$H = D_{\max}$	$f_i(o_i) = 0.0313o_i - 9.296$
$D_i = (D_{\max} - D_{\min})\beta + \delta D_{\min}$	
System Model	
$E_{\min} = \min_{l \in \mathcal{L}}\{\sum_{i \in \mathcal{N}} \frac{M_i+O_i}{\gamma_{i,l} f_l}(P_{sta,l} + P_{dyn,l})\}$	
$E_{\max} = \max_{l \in \mathcal{L}}\{\sum_{i \in \mathcal{N}} \frac{M_i+O_i}{\gamma_{i,l} f_l}(P_{sta,l} + P_{dyn,l})\}$	
$E_{buget} = (E_{\max} - E_{\min})\beta + E_{\min} + (M_B + M_L)HP_{on}$	

Next, we first evaluate the influence of system parameters on task deployment results. Then, we compare the system performance (i.e., system QoS, problem schedulability, and computation time) of the proposed optimal (OPT), the heuristic (HEU) methods, and other state-of-the-art deployment schemes: i) task deployment without task migration (WTM) [24], ii) task deployment without DVFS (WDV) [20], i.e., each processor has a fixed voltage/frequency, and iii) task deployment with fixed task allocation (WTA) [5], i.e., task-to-processor allocation is fixed. The simulations are performed on a workstation with 32 processors and 64 GB RAM, and the algorithms are implemented in Matlab 2019a with optimization solver Gurobi 9.

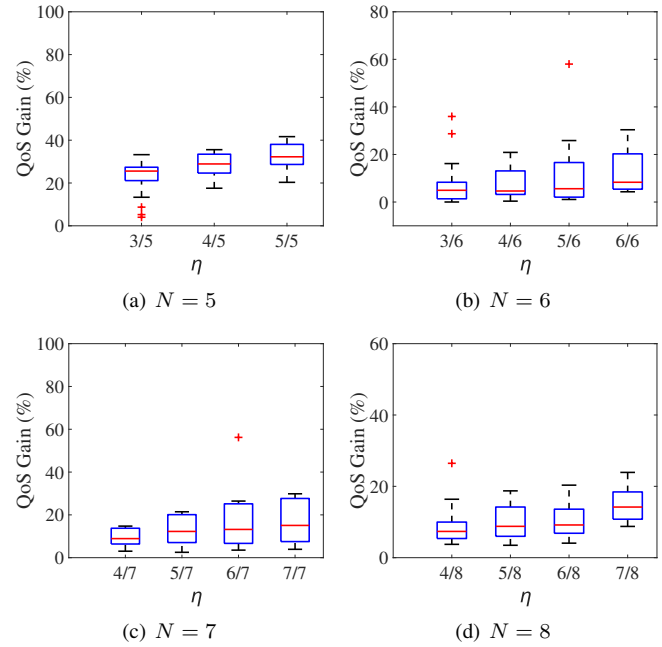
2) Simulation Results: Fig. 6 compares the system QoS achieved by the OPT method under different task numbers N and system parameters β and δ. Note that β and δ will change energy supply E_{buget} and task deadline D_i, and thus, influence the system QoS. We set the requirements regarding time and energy in the intermediate level, and thus, we have $\beta \in [0, 0.5]$ and $\delta \in [0.4, 1]$. In addition, the execution efficient factors are set to $\gamma_{i,l} = 0.6$ ($l \in \mathcal{P}_L$) and $\gamma_{i,l} = 1$ ($l \in \mathcal{P}_B$) [11]. Fig. 6 shows that the system QoS increases with β, δ, and N. Moreover, under the same task number N, with β and δ increasing, the growth rate of system QoS decreases. This is because the larger the values of β, δ, and N, the more optional subtasks are executed, and thus, a higher system QoS can be achieved. However, under the fixed task number N, although the processors can execute more optional cycles with β and δ increasing, the QoS improvement becomes limited, since the range of optional cycles o_i is bounded by $0 \leq o_i \leq O_i$.

Fig. 7 evaluates the influence of DAG structure on system



(a) $N = 5$ (b) $N = 6$ (c) $N = 7$ (d) $N = 8$

Fig. 6. The system QoS achieved by OPT method with N, δ, and β varying.



(a) $N = 5$ (b) $N = 6$ (c) $N = 7$ (d) $N = 8$

Fig. 7. The change of QoS gain (OPT vs WTM) with η and N varying.

QoS. Let $N_{\mathcal{CPT}}$ denote the number of the tasks on the critical path, and we introduce a task parallelism factor $\eta = N_{\mathcal{CPT}}/N$. Since N is the number of total tasks, the larger the value of η, the lower the parallelism level of task DAG. The box plot of 'OPT vs WTM' shows the statistical property of data set $\{\frac{f_{OPT}(N,\eta)-f_{WTM}(N,\eta)}{f_{OPT}(N,\eta)}\}$ for all the tuned N and η parameters, where $f_{OPT}(N,\eta)$ and $f_{WTM}(N,\eta)$ are the system QoS of OPT and WTM under the given N and η parameters. On each box, the central mark indicates the median. The bottom and the top edges indicate the 25th and the 75th percentiles, respectively. The whiskers extend to the most extreme data points that are not considered outliers, and the outliers are plotted individually by the '+' symbol. From Fig. 7, we observe that the QoS gain increases with η. When DAG has a higher parallelism task execution level (i.e., η is small), each processor has more idle time intervals to execute the optional subtasks. Therefore, the differences in system QoS achieved by OPT and WTM methods are the smallest observed. However, when the parallelism level of task execution is low (i.e., η is large), the task deployment process of OPT has higher gains. This is because OPT is more flexible due to the task migration, allowing one subtask to be executed on different (heterogeneous) processors. Thus, we can better use system resources to increase QoS under time and energy constraints.

Fig. 8 evaluates the influence of processor heterogeneity on the system QoS. As the processors in the same cluster are homogeneous, we introduce a heterogenous factor γ_L/γ_B, where $\gamma_L = \gamma_{i,l}$ $(l \in \mathcal{P}_L)$ and $\gamma_B = \gamma_{i,l}$ $(l \in \mathcal{P}_B)$. Thus, γ_L/γ_B represents the ratio of execution efficiency factor between the processors in the LITTLE and big clusters. When the V/F level (V_l, f_l) is used to execute a task with M_i cycles, the task execution time and energy are $\frac{M_i}{\gamma_{i,l} f_l}$ and $\frac{M_i}{\gamma_{i,l} f_l}(P_{sta,l} + P_{dyn,l})$, respectively. Hence, if we fix the parameters M_i, f_l, $P_{sta,l}$ and $P_{dyn,l}$, the larger the value of γ_L/γ_B, the smaller the gap regarding task execution between processors of LITTLE and big clusters. We set $N \in [5, 8]$, $\gamma_L \in [0.5, 1]$ and $\gamma_B = 1$. The box plot of 'OPT vs WTM' shows the statistical property of data set $\{\frac{f_{OPT}(N,\gamma_L/\gamma_B)-f_{WTM}(N,\gamma_L/\gamma_B)}{f_{OPT}(N,\gamma_L/\gamma_B)}\}$ for all the tuned N and γ_L/γ_B parameters, where $f_{OPT}(N,\gamma_L/\gamma_B)$ and $f_{WTM}(N,\gamma_L/\gamma_B)$ are the system QoS of OPT and WTM under the given N and γ_L/γ_B parameters. From Fig. 8, we observe that when the differences of processors in big and LITTLE clusters are large (i.e., γ_L/γ_B is low), task migration is more efficient, and thus, OPT can achieve a higher system QoS. On the other hand, with γ_L/γ_B increasing (i.e., γ_L/γ_B is close to 1), the heterogeneity of the processors in big and LITTLE clusters is reduced. Since task execution time $\frac{M_i}{\gamma_{i,l} f_l}$ and energy $\frac{M_i}{\gamma_{i,l} f_l}(P_{sta,l} + P_{dyn,l})$ decrease, we can execute more optional cycles to improve system QoS, and thus, OPT can achieve a higher system QoS.

Fig. 9 compares the system QoS and the problem feasibility of OPT and WTM, as both task deployment problems are solved by the optimal methods. The system QoS and problem feasibility are given by $\sum_{i \in \mathcal{N}} f_i(o_i)$ and n_{fea}/n, respectively, where n_{fea} is the number of feasible solutions found during n simulations. We run the simulations $n = 30$ times, under the time factors $\beta = 0.4$ and $\delta = 0.4$, execution efficient factors $\gamma_{i,l} = 1$ $(\forall l \in \mathcal{P}_B)$ and $\gamma_{i,l} = 0.6$ $(\forall l \in \mathcal{P}_L)$, and task number $N \in [5, 19]$. From Fig. 9 we observe that, compared with WTM, OPT achieves a higher QoS (31.2% on average and up to 112.8%) and higher problem feasibility (72.54% on average). This is because, with task migration, a set of additional variables μ_{2i-1} and μ_{2i} (i.e., the proportions of a task τ_i executed on the processors of different clusters) are introduced into the task deployment problem, which makes this problem easier to solve, under the same constraints. The values of μ_{2i-1} and μ_{2i} are fixed during task deployment without task migration. Note that the multi-core system is time- and energy-constrained, and the processors in big and LITTLE clusters have

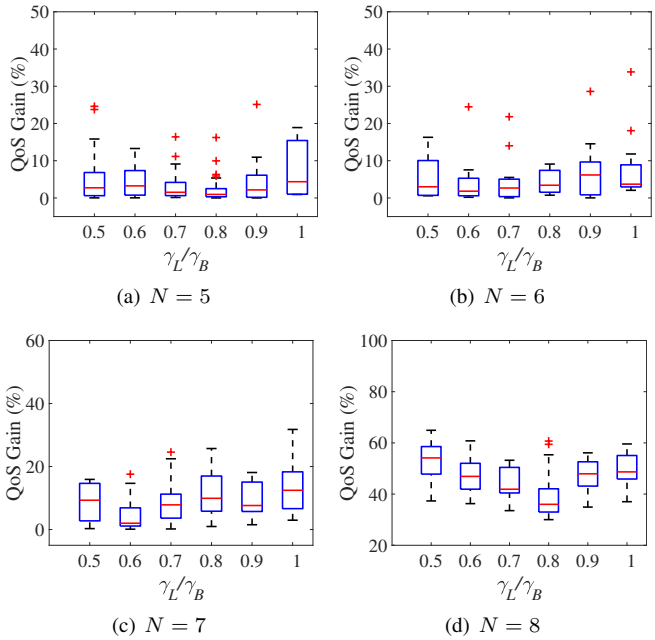


Fig. 8. The change of QoS gain (OPT vs WTM) with γ_L/γ_B and N varying.

different performances, e.g., computation time and energy. Thus, separating a task into two parts and assigning them to different clusters can better balance energy consumption and execution time, satisfying the problem constraints.

Fig. 10 compares the system QoS and the computation time of OPT, HEU, WDV, and WTA based on both randomly generated DAGs and the DAGs from real applications, such as Gaussian Elimination (GE) [37], Fast Fourier Transform (FFT) [37], Laplace equation (LE) [38], and Montage Workflows (MW) [39]. We set $\gamma_{i,l} = 1$ ($\forall l \in \mathcal{P}_B$), $\gamma_{i,l} = 0.6$ ($\forall l \in \mathcal{P}_L$), and $\eta \in (0, 1]$. For randomly generated DAGs, $N \in [5, 100]$; while for GE, FFT, LE, and MW, the applications can be modeled as DAGs with 14, 15, 16, and 24 tasks, respectively. In Fig. 10(a) and Fig. 10(b), the QoS gain between OPT and HEU is defined as $\frac{f_{OPT}(N)-f_{HEU}(N)}{f_{OPT}(N)}$, where $f_{OPT}(N)$ and $f_{HEU}(N)$ are the system QoS achieved by OPT and HEU under the given N parameter, respectively. Note that WDV and WTA are based on multiple-step heuristic methods. Fig. 10(a) and Fig. 10(b) show that our method (HEU) outperforms WDV and WTA in terms of QoS improvement (22.89% and 27.62% on average, including randomly generated DAGs and DAGs for GE, FFT, LE, and MW), as DVFS and task allocation are taken into account and optimized in the HEU method. On the other hand, with task number N increasing, more variables and constraints are added to the task deployment method. Therefore, the computation time to find the optimal solution increases as well. Fig. 10(c) shows that, compared with the OPT method, the HEU method has a negligible time (within 0.5 seconds) and achieves about 53.9% (on average) performance of the OPT method. Hence, our HEU approach is suitable for applications with larger and more complex task sets, which can enhance the adaptivity and scalability of the task deployment process.

VI. Conclusion

This paper addresses the problem of dependent IC task deployment on the AMP platforms to maximize system QoS.

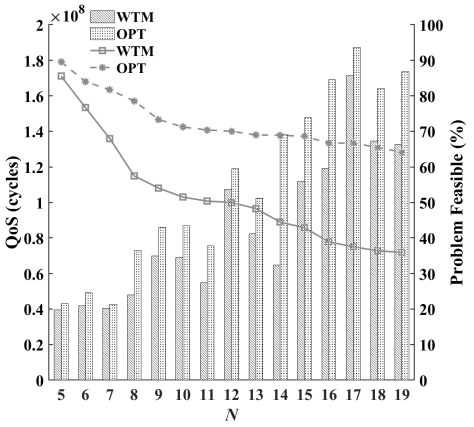


Fig. 9. QoS and problem feasibility comparisons of OPT with WTM.

The task allocation, scheduling, migration, and frequency assignment are optimized concurrently under real-time and energy supply constraints. The joint-design task deployment problem is formulated as an MINLP problem and then is equivalently transformed into an MILP problem to find the optimal solution. To improve the scalability of the proposed method, we design a novel three-step heuristic with low computation time to solve this problem. This method decomposes the joint-design problem into three sub-problems with fewer variables and constraints. The sub-problems regarding the assignment of task execution frequency, the allocation and scheduling of tasks, and the adjustment of optional subtasks are solved in sequence. The simulation results show that the proposed heuristic can obtain an acceptable solution with a negligible computation time. In addition, the proposed methods outperform other task deployment methods in terms of the improvement of system QoS and the usage of system resources.

References

[1] S. Mittal, "A survey of techniques for architecting and managing asymmetric multicore processors," *ACM Computing Surveys*, vol. 48, no. 3, pp. 1–38, 2016.

[2] A. K. Singh, M. Shafique, A. Kumar, and J. Henkel, "Mapping on multi/many-core systems: survey of current and emerging trends," in *Proc. ACM Design Automation Conference*, 2013, pp. 1–10.

[3] S. Saha, X. Zhai, S. Ehsan, S. Majeed, and K. McDonald-Maier, "RASA: Reliability-aware scheduling approach for FPGA-based resilient embedded systems in extreme environments," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 52, no. 6, pp. 3885–3899, 2022.

[4] S. Yoo, "An empirical validation of power-performance scaling: DVFS vs. multi-core scaling in big.LITTLE processor," *IEICE Electronics Express*, vol. 12, no. 8, pp. 1–9, 2015.

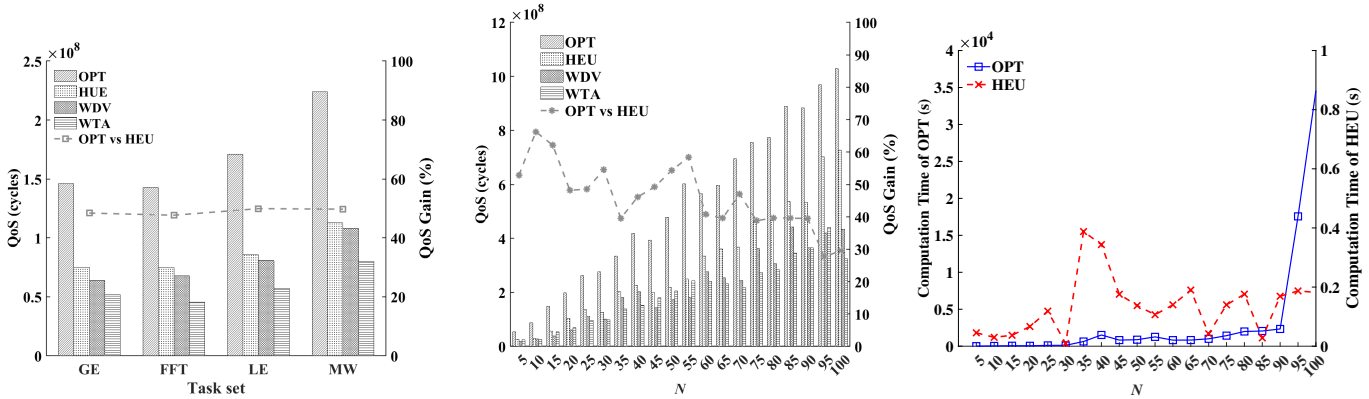
[5] H. Yu, Y. Ha, and B. Veeravalli, "Quality-driven dynamic scheduling for real-time adaptive applications on multiprocessor systems," *IEEE Transactions on Computers*, vol. 62, no. 10, pp. 2026–2040, 2012.

[6] V. K. Chippa, D. Mohapatra, K. Roy, S. T. Chakradhar, and A. Raghunathan, "Scalable effort hardware design," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 22, no. 9, pp. 2004–2016, 2014.

[7] S. K. Roy, R. Devaraj, A. Sarkar, and D. Senapati, "SLAQA: Quality-level aware scheduling of task graphs on heterogeneous distributed systems," *ACM Transactions on Embedded Computing Systems*, vol. 20, no. 5, pp. 1–31, 2021.

[8] J. W. S. Liu, W. K. Shih, K. J. Lin, R. Bettati, and J. Y. Chung, "Imprecise computations," *Proceedings of the IEEE*, vol. 82, no. 1, pp. 83–94, 1994.

[9] G. Chen, K. Huang, and A. Knoll, "Energy optimization for real-time multiprocessor system-on-chip with optimal DVFS and DPM combination," *ACM Transactions on Embedded Computing Systems*, vol. 13, no. 3, pp. 1–21, 2014.



(a) QoS of four real-world task sets. (b) QoS of different task deployment methods. (c) Computation time of OPT and HEU methods.

Fig. 10. QoS and computation time comparisons of different task deployment methods.

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

[11] D. Li and J. Wu, "Minimizing energy consumption for frame-based tasks on heterogeneous multiprocessor platforms," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 3, pp. 810–823, 2015.

[12] S. Abd Ishak, H. Wu, and U. U. Tariq, "Energy-aware task scheduling on heterogeneous NoC-based MPSoCs," in *Proc. IEEE International Conference on Computer Design*, 2017, pp. 165–168.

[13] Y. Li, J. Niu, M. Atiquzzaman, and X. Long, "Energy-aware scheduling on heterogeneous multi-core systems with guaranteed probability," *Journal of Parallel and Distributed Computing*, vol. 103, pp. 64–76, 2017.

[14] K. Siddesha and G. Jayaramaiah, "Energy-aware task scheduling approach using DVFS and particle swarm optimization for heterogeneous multicore processors," in *Emerging Research in Computing, Information, Communication and Applications*, 2022, pp. 943–955.

[15] G. Zeng, Y. Matsubara, H. Tomiyama, and H. Takada, "Energy-aware task migration for multiprocessor real-time systems," *Future Generation Computer Systems*, vol. 56, pp. 220–228, 2016.

[16] D. Rupanetti and H. Salamy, "Energy-aware task migration through ant-colony optimization for multiprocessors," in *Proc. IEEE Annual Ubiquitous Computing, Electronics & Mobile Communication Conference*, 2021, pp. 0901–0907.

[17] S. Moulik, A. Sarkar, and H. K. Kapoor, "Energy aware frame based fair scheduling," *Sustainable Computing: Informatics and Systems*, vol. 18, pp. 66–77, 2018.

[18] H. S. Chwa, J. Seo, H. Yoo, J. Lee, and I. Shin, "Energy and feasibility optimal global scheduling framework on big.LITTLE platforms," in *Proc. IEEE Real-Time Scheduling Open Problems Seminar*, 2015, pp. 1–11.

[19] K. H. Kim, "Reward-based allocation of cluster and grid resources for imprecise computation model-based applications," *International Journal of Web and Grid Services*, vol. 9, no. 2, pp. 146–171, 2013.

[20] J. Zhou, J. Yan, T. Wei, M. Chen, and X. S. Hu, "Energy-adaptive scheduling of imprecise computation tasks for QoS optimization in real-Time MPSoC systems," in *Proc. IEEE/ACM Design, Automation and Test in Europe*, 2017, pp. 1402–1407.

[21] I. Méndez-Díaz, J. Orozco, R. Santos, and P. Zabala, "Energy-aware scheduling mandatory/optional tasks in multicore real-time systems," *International Transactions in Operational Research*, vol. 24, no. 1-2, pp. 173–198, 2017.

[22] L. Mo, A. Kritikakou, and O. Sentieys, "Energy-quality-time optimized task mapping on DVFS-enabled multicores," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2428–2439, 2018.

[23] R. Ravindran, C. M. Krishna, I. Koren, and Z. Koren, "Scheduling imprecise task graphs for real-time applications," *International Journal of Embedded Systems*, vol. 6, no. 1, pp. 73–85, 2014.

[24] S. Chakraborty, S. Saha, M. Själander, and K. Mcdonald-Maier, "Prepare: Power-aware approximate real-time task scheduling for energy-adaptive QoS maximization," *ACM Transactions on Embedded Computing Systems*, vol. 20, no. 5s, pp. 1–25, 2021.

[25] T. Wei, J. Zhou, K. Cao, P. Cong, M. Chen, G. Zhang, X. S. Hu, and J. Yan, "Cost-constrained QoS optimization for approximate computation real-Time tasks in heterogeneous MPSoCs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 9, pp. 1733–1746, 2017.

[26] G. L. Stavrinides and H. D. Karatza, "An energy-efficient, QoS-aware and cost-effective scheduling approach for real-time workflow applications in cloud computing systems utilizing DVFS and approximate computations," *Future Generation Computer Systems*, vol. 96, pp. 216–226, 2019.

[27] L. Mo, A. Kritikakou, and O. Sentieys, "Approximation-aware task deployment on asymmetric multicore processors," in *Proc. IEEE/ACM Design, Automation and Test in Europe*, 2019, pp. 1513–1518.

[28] Y. Qin, G. Zeng, R. Kurachi, Y. Matsubara, and H. Takada, "Execution-variance-aware task allocation for energy minimization on the big. little architecture," *Sustainable Computing: Informatics and Systems*, vol. 22, pp. 155–166, 2019.

[29] H. Yu, B. Veeravalli, Y. Ha, and S. Luo, "Dynamic scheduling of imprecise-computation tasks on real-time embedded multiprocessors," in *Proc. IEEE International Conference on Computational Science and Engineering*, 2013, pp. 770–777.

[30] L. A. Cortés, P. Eles, and Z. Peng, "Quasi-static assignment of voltages and optional cycles in imprecise-computation systems with energy considerations," *IEEE transactions on very large scale integration systems*, vol. 14, no. 10, pp. 1117–1129, 2006.

[31] R. Jejurikar, C. Pereira, and R. Gupta, "Leakage aware dynamic voltage scaling for real-time embedded systems," in *Proc. Design Automation Conference*, 2004, pp. 275—280.

[32] S. M. Martin, K. Flautner, T. Mudge, and D. Blaauw, "Combined dynamic voltage scaling and adaptive body biasing for lower power microprocessors under dynamic workloads," in *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2002, pp. 721—725.

[33] W. Wang and P. Mishra, "Leakage-aware energy minimization using dynamic voltage scaling and cache reconfiguration in real-time systems," in *Proc. International Conference on VLSI Design*, 2010, pp. 357–362.

[34] S. Deniziak and L. Ciopiński, "Synthesis of self-adaptable energy aware software for heterogeneous multicore embedded systems," *Microelectronics Reliability*, vol. 123, p. 114184, 2021.

[35] S. Burer and A. N. Letchford, "Non-convex mixed-integer nonlinear programming: A survey," *Surveys in Operations Research and Management Science*, vol. 17, no. 2, pp. 97–106, 2012.

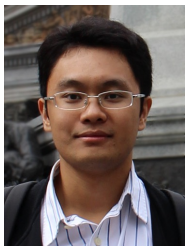
[36] H. Huang, V. Chaturvedi, G. Quan, J. Fan, and M. Qiu, "Throughput maximization for periodic real-time systems under the maximal temperature constraint," *ACM Transactions on Embedded Computing Systems*, vol. 13, no. 2, pp. 363–368, 2014.

[37] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE transactions on parallel and distributed systems*, vol. 13, no. 3, pp. 260–274, 2002.

[38] M. Wu and D. Gajski, "Hypertool: A programming aid for message-passing systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 1, no. 3, pp. 330–343, 1990.

[39] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good *et al.*, "Pegasus: A framework for mapping complex scientific workflows onto distributed systems," *Scientific Programming*, vol. 13, no. 3, pp. 219–237, 2005.

Xinmei Li received her B.S. degree in automation engineering from Sichuan University, Chengdu, China, in 2021. She is currently working towards the M.S. degree in Control Science and Engineering at Southeast University, Nanjing, China. Her current research interests include embedded and real-time systems, and network-on-chip.



Lei Mo (S'13–M'17) is currently an associate professor with the School of Automation, Southeast University, Nanjing, China. He received the B.S. degree from the College of Telecom Engineering and Information Engineering, Lanzhou University of Technology, Lanzhou, China, in 2007, and the Ph.D. degree from the College of Automation Science and Engineering, South China University of Technology, Guangzhou, China, in 2013. From 2013 to 2015, he was a research fellow with the Department of Control Science and Engineering, Zhejiang University, China. From 2015 to 2017, he was a research fellow with INRIA Nancy–Grand Est, France. From 2017 to 2019, he was a research fellow with INRIA Rennes–Bretagne Atlantique, France. His current research interests include networked estimation and control in wireless sensor and actuator networks, cyber-physical systems, task mapping and resource allocation in embedded systems. He serves as an Associate Editor for KSII Transactions on Internet and Information Systems, and International Journal of Ad Hoc and Ubiquitous Computing, and a TPC Member for several international conferences.

Angeliki Kritikakou is currently an Associate Professor at University of Rennes 1 and IRISA - INRIA Rennes research center. She received her Ph.D. in 2013 from the Department of Electrical and Computer Engineering at University of Patras, Greece and in collaboration with IMEC Research Center, Belgium. She worked for one year as a Postdoctoral Research Fellow at the Department of Modelling and Information Processing (DTIM) at ONERA in collaboration with Laboratory of Analysis and Architecture of Systems (LAAS) and the University of Toulouse, France. Her research interests include embedded systems, real-time systems, mixed-critical systems, hardware/software co-design, mapping methodologies, design space exploration methodologies, memory management methodologies, low power design, and fault tolerance.

Olivier Sentieys (M'94) is a Professor at the University of Rennes holding an INRIA Research Chair on Energy-Efficient Computing Systems. He is leading the Cairn team common to INRIA (French research institute dedicated to computational sciences) and IRISA Laboratory. He is also the head of the Computer Architecture department of IRISA. From 2012 to 2017, he was on secondment at INRIA as a Senior Research Director. His research interests are in the area of computer architectures, embedded systems and signal processing, with a focus on system-level design, energy-efficiency, reconfigurable systems, hardware acceleration, approximate computing, and power management of energy harvesting sensor networks. He authored or co-authored more than 250 journal or conference papers, holds 6 patents, and served in the technical committees of several international IEEE/ACM/IFIP conferences.