An Energy Minimization Approach for Periodic Tasks with Data Dependencies in Heterogeneous Systems

JING HUANG, School of Computer Science and Engineering, Hunan University of Science and Technology, Xiangtan, China
JIANG KUAN, School of Computer Science and Engineering, Hunan University of Science and Technology, Xiangtan, China
LIJUN XIAO, College of Computer Science and Electronic Engineering, Hunan University of Science and Technology, Xiangtan, China
WEI LIANG, School of Computer Science and Engineering, Hunan University of Science and Technology, Xiangtan, China
HAIBO ZENG, Virginia Tech, Blacksburg, United States

Heterogeneous systems offer high instruction throughput and cost advantages but face dual challenges of energy efficiency and task scheduling, especially for periodic tasks with data dependencies—such as signal processing and process control—which require precise execution and careful handling of continuous sampling, data transmission, and processing. The Directed Acyclic Graph (DAGs) are commonly used to capture data dependencies among tasks, where nodes and directed edges represent tasks and data dependencies, respectively. This paper focuses on the non-preemptive scheduling problem of such tasks on heterogeneous platforms for time-critical applications, aiming to minimize energy consumption while guaranteeing worst-case deadline constraints. Clearly, this is an NP-hard problem, making it difficult to obtain an optimal solution in polynomial time. Although Dynamic Voltage and Frequency Scaling (DVFS) is a widely used energy-saving technique, its practical effectiveness is limited by potential transient faults, reduced processor lifespan, and overheads caused by switching. First, this paper analyzes priority constraints within DAGs and simplifies the structure by removing unnecessary dependencies to improve execution efficiency. Then, it reduces energy consumption through two sequential approaches: task scheduling optimization and processor frequency adjustment. Accordingly, two energy-efficient algorithms are proposed: the Time and Energy Balanced Scheduling (TEBS) algorithm and the DVFS-Weakly Dependent Energy Saving (DWDES) algorithm. The former reduces makespan and energy consumption purely through scheduling strategies without considering DVFS, while the latter adjusts processor frequencies based on TEBS to further minimize energy use, allowing DVFS only during application switching. Experimental results show that the proposed algorithms achieve significant energy savings compared to state-of-the-art methods while satisfying deadline constraints. Specifically, taking the classic IPPTS algorithm's energy consumption as the baseline, TEBS and DWDES achieve energy savings of 15.58% and 46.31%, respectively.

Additional Key Words and Phrases: Directed acyclic graph, data dependency, energy, heterogeneous systems, periodic tasks, scheduling

Authors' Contact Information: Jing Huang, School of Computer Science and Engineering, Hunan University of Science and Technology, Xiangtan, China; e-mail: huangjing@hnust.edu.cn; Jiang Kuan, School of Computer Science and Engineering, Hunan University of Science and Technology, Xiangtan, China; e-mail: jkuan2022@163.com; Lijun Xiao, College of Computer Science and Electronic Engineering, Hunan University of Science and Technology, Xiangtan, Hunan, China; e-mail: lijunxiao@hnust.edu.cn; Wei Liang, School of Computer Science and Engineering, Hunan University of Science and Technology, Xiangtan, Hunan, China; e-mail: wliang@hnust.edu.cn; Haibo Zeng, Virginia Tech, Blacksburg, Virginia, United States; e-mail: hbzeng@vt.edu.

ACM 1558-3465/2025/11-ART
https://doi.org/10.1145/3777380

1 Introduction

With the rapid development of information technology and intelligent systems, heterogeneous systems have become an important solution to meet the high-performance computing demands. Heterogeneous systems typically consist of different types of computing units (such as CPUs, GPUs, FPGAs, etc.), each with varying computational capabilities and energy efficiency characteristics. Compared to traditional homogeneous systems, heterogeneous platforms can significantly improve system throughput and computational performance by adaptively scheduling tasks and leveraging the strengths of different processing units. However, despite their high computational power, minimizing energy consumption while maintaining high performance remains a significant challenge for the widespread adoption of these systems [1].

In many real-time applications, the system must not only perform complex computations, but also perform tasks efficiently under strict time constraints. This is especially true for applications involving periodic tasks with data dependencies, such as signal processing, process control, and real-time data monitoring, which require continuous sampling and processing of data [2, 3]. These applications require tasks to be completed within predefined time windows, and the tasks often have intricate dependencies that dictate the order in which they must be executed. Directed Acyclic Graphs (DAGs) are commonly used to model these types of applications [4–7], where nodes and edges represent tasks and data dependencies between these tasks, respectively. By modeling tasks and their dependencies with a DAG, the scheduling order, execution times, and task dependencies can be clearly defined, providing effective decision support for real-time scheduling.

However, applying DAG scheduling algorithms on heterogeneous systems presents several challenges [8]. First, since different processing units within the system have varying performance and energy efficiency characteristics, the core challenge is to allocate tasks to the most suitable processors while minimizing overall energy consumption under resource and constraint limitations [9, 10]. Additionally, scheduling periodic tasks with data dependencies must ensure that timing constraints are met, particularly when tasks are interdependent. The scheduling strategy needs to precisely account for task completion order and parallelism while minimizing the communication overhead between processors, reducing energy consumption, and improving overall system efficiency. Although there has been some research into energy efficiency optimization and real-time scheduling in heterogeneous systems, most methods still struggle to balance computational performance with energy efficiency. Traditional scheduling algorithms often focus on optimizing a single objective (such as minimizing execution time or maximizing parallelism), whereas real-world applications typically require the simultaneous optimization of multiple objectives, such as task completion time, system energy efficiency, and inter-task data transfer overhead. This makes the DAG scheduling problem in heterogeneous systems not only computationally complex but also involves the challenge of designing scheduling algorithms that can balance real-time requirements and energy efficiency within a multi-objective optimization framework.

For real-time applications with data dependencies—such as those in avionics and process control—that require continuous data sampling and processing, this paper aims to investigate the non-preemptive scheduling problem of such applications on heterogeneous platforms, striving to reduce energy consumption while meeting worst-case deadline constraints. Our contributions are as follows:

(1) In the DVFS-independent domain, we propose a Time and Energy Balanced Scheduling (TEBS) algorithm. This algorithm assigns dynamic priorities to tasks and preferentially allocates them to processors that yield the lowest completion time and energy consumption while satisfying deadline constraints. The goal is to achieve both reduced makespan and energy use. Experimental results show that TEBS reduces energy consumption by 15.58% compared to the classic IPPTS algorithm.
(2) In the DVFS-weakly dependent domain (where DVFS is applied only during application switching), we present a DVFS-Weakly Dependent Energy Saving (DWDES) algorithm. Building on the TEBS scheduling results, this algorithm iteratively adjusts each processor's frequency by considering processor power and

workload, prioritizing the frequency reduction of the processor with the highest potential energy savings to maximize overall efficiency while satisfying deadline constraints. Experimental results demonstrate that DWDES achieves a 46.31% reduction in energy consumption compared to the classic IPPTS algorithm.

The structure of this paper is as follows. Section 2 reviews related work, while Section 3 defines the models and formulates the problem. Section 4 presents an analysis of precedence constraints. Section 5 introduces the proposed energy-efficiency scheduling algorithms, followed by experimental results in Section 6. Finally, Section 7 concludes the paper.

2 Related Work

In this section, we will review the related work from three aspects: DAG structure optimization, scheduling length optimization and energy-saving optimization.

DAG structure optimization. In many intelligent systems, such as the Autonomous Driving System (ADS) [9, 11], there are complex data dependencies between tasks with different activation rates, which makes it very difficult to analyze the real-time behavior of the system [12, 13]. Some studies have modeled multi-rate tasks with dependency constraints as a single-rate DAG (Directed Acyclic Graph) model. For example, Saidi *et al.* [14] proposed an alternative method for converting similar DAG models from multi-rate to single-rate. Verucchi *et al.* [9] proposed a method for converting a multi-rate DAG task-set with time constraints into a single-rate DAG that optimizes schedulability and end-to-end latency. Zhao *et al.* [5] conducted DAG scheduling and analysis by modeling parallelism and dependency. In [15], an integrated framework is proposed to analyze the schedulability of individual tasks and the end-to-end delay of task chains in a DAG. Various methods have been proposed to optimize the structure of DAG to improve parallelism and execution efficiency. Key techniques include minimizing the number of edges to reduce complexity, and reducing the number of critical paths in a DAG can improve the throughput of the system and reduce latency. Additionally, techniques such as topological sorting and graph partitioning are often used to reconstruct DAGs for better performance. In addition, there are some classic methods for optimizing DAG structure. One notable method is the Floyd-Warshall algorithm [16].

Scheduling length optimization. One of the most classic studies in this field is the Heterogeneous Earliest Finish Time (HEFT) algorithm, which has been widely recognized by researchers for its relatively low time complexity and short schedule length [17]. The task allocation in HEFT is essentially a greedy strategy, which may fall into local optima and miss the global optimal allocation. To address this drawback of HEFT, Djigal et al. [18] proposed the Improved Predict Priority Task Scheduling (IPPTS) algorithm inspired by foresight thinking, which introduces forward-looking planning in both the task priority determination phase and the processor selection phase to shorten the schedule length. In addition, Sun et al. [19] proposed a novel schedulability testing method based on the concept of trivial schedulability for scheduling Directed Acyclic Graph (DAG) tasks in real-time systems, and developed a deep reinforcement learning algorithm to effectively solve the edge generation problem, thereby minimizing the DAG width while ensuring deadline constraints. Chen et al. [20] focused on the energy-constrained workflow scheduling problem in heterogeneous multiprocessor embedded systems. They proposed an improved energy allocation strategy and a novel scheduling algorithm based on an energy difference coefficient, aiming to optimize processor allocation, operating frequency, and task start time while satisfying data dependencies and energy constraints, thereby minimizing the workflow makespan. The main difference between their work and this study lies in the task type: their research addresses traditional aperiodic tasks, whereas this paper focuses on data-dependent periodic tasks, each with strict release times and deadlines. Chen et al. [21] tackled the non-preemptive scheduling problem for data-dependent periodic tasks on heterogeneous multiprocessor platforms. They presented an exact formulation based on mixed-integer linear programming and an efficient list-based offline scheduling algorithm, which improves the scheduling success rate and reduces task completion times and deviation ratios. Although the task type studied in their work is consistent with that

Table 1. Parameters for 7 periodic tasks in Fig. 1

	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6	τ_7
$r_{i,1}$	0	4	0	4	0	5	8
w_i	4	5	5	7	6	4	3
d_i	20	60	20	30	30	20	60
t_i	20	60	20	30	30	20	60

of this paper, the research objectives differ: their study aims to shorten the scheduling length by optimizing the scheduling algorithm, while this work focuses on addressing energy consumption under stringent deadline constraints.

Energy-saving optimization. The energy optimization techniques employed include Dynamic Power Management (DPM) and Dynamic Voltage and Frequency Scaling (DVFS) [22]. DVFS reduces overall energy consumption by lowering the processor's voltage and frequency, achieving a balance between performance and energy efficiency. Meanwhile, DPM minimizes power usage by keeping processors in low-power idle states for as long as possible, ensuring that all real-time tasks meet their respective deadlines. Zhang et al. [23] investigated the joint energy efficiency and reliability management of fixed-priority periodic real-time task sets sharing resources in a standby-sparing system. They proposed a novel algorithm called FPMPSA, which takes into account the overhead of frequency transitions and processor state changes introduced by Dynamic Voltage and Frequency Scaling (DVFS) and Dynamic Power Management (DPM). The algorithm also employs a Rate Monotonic Dual Priority (RM/DPP) scheduling strategy to ensure mutual exclusion when accessing shared resources. Hagras et al. [24] introduced a mechanism called BlueMoon, which reschedules application tasks to extend each task's execution slot while keeping the overall application completion time unchanged. BlueMoon significantly lengthens the execution windows of tasks, thereby effectively reducing the energy consumption of dependent task applications on DVFS-enabled computing platforms.

This study focuses on DVFS technology. In this regard, Huang et al. [25] proposed an energy-efficient scheduling approach with deadline constraints for randomly arriving DAG tasks in multi-ECU embedded systems. Senapati et al. [26] developed an energy-efficient real-time scheduling algorithm for periodic DAGs in heterogeneous systems. These approaches typically achieve energy savings through DVFS; however, frequent use of DVFS can shorten processor lifespan and introduce additional time and energy overheads. To address this, Huang et al. [27] successively proposed the DVFS Non-Dependent Scheduling (DNDS) and DVFS Weakly Dependent Scheduling (DWDS) algorithms for DAG tasks with deadline constraints on heterogeneous computing systems. It is worth noting that DWDS was developed based on DNDS to further reduce energy consumption through weak DVFS dependencies. DNDS adjusts the task allocation strategy based on HEFT to save energy while meeting deadline constraints. However, although DNDS reduces some energy consumption through its scheduling strategy, it also increases the schedule length, which in turn limits the energy-saving potential of DWDS. This indicates that DNDS does not achieve an optimal balance between time and energy. Therefore, this study focuses on addressing this issue.

Carefully modeling multi-rate tasks with dependency constraints as a single-rate DAG is a crucial step, and simultaneously ensuring high parallelism and execution efficiency presents a challenge. In addition, researching task scheduling algorithms that can reduce system energy consumption under the deadline constraints is of great significance for the continuous development of more sustainable and efficient heterogeneous systems.

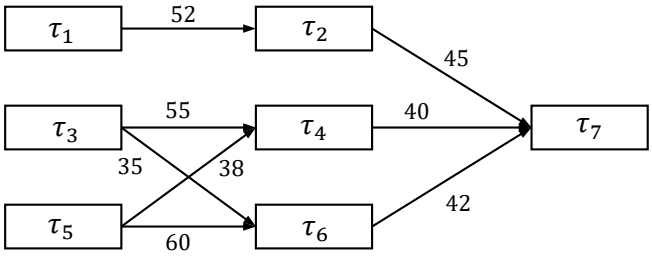


Fig. 1. A relationship graph contains 7 periodic tasks and 8 dependencies.

Table 2. The parameters of simulation environment

Notation	Parameter
$G = (T, E)$	A directed acyclic graph application
$T = \{\tau_1, \tau_1, \ldots, \tau_m\}$	The set of periodic tasks
$E = \{e_{i,j} \mid \tau_i, \tau_j \in T\}$	The dependency relationships between tasks
$HP = lcm(t_1, t_2, \ldots, t_m)$	The hyper-period of tasks
t_i	The job release period of task τ_i
w_i	The computation workload (CPU-cycles) of task τ_i
$J_{i,k}$	The k-th job released by task τ_i
$r_{i,k}$	The release or arrival time of job $J_{i,k}$
$d_{i,k}$	The related deadline of job $J_{i,k}$
$c_{i,j}$	The communication cost between tasks τ_i and τ_j
D	The deadline of Job-DAG F
F	The job directed acyclic graph (Job-DAG)
$\boldsymbol{J}$	The set of released jobs within the hyper-period HP
$\boldsymbol{J}_{\text{entry}}$	The set of all entry jobs J_{entry}
$\boldsymbol{J}_{\text{exit}}$	The set of all exit jobs J_{exit}
$U = \{u_1, u_2, \ldots, u_{\lvert U \rvert}\}$	The processor set of a system
$F_p = \{f_{p,1}, f_{p,2}, \ldots, f_{p,\lvert F_p \rvert}\}$	The available frequency set of processor u_p
$pre(J_{i,k})$	The predecessor job set of job $J_{i,k}$
$suc(J_{i,k})$	The successor job set of job $J_{i,k}$
$AT(J_{i,k})$	The arrival time of job $J_{i,k}$
$ST_p(J_{i,k})$	The start time of job $J_{i,k}$ on processor u_p
$FT_p(J_{i,k})$	The finish time of job $J_{i,k}$ on processor u_p
$M(F)$	The makespan of the Job-DAG F
$P_{p,s}, P_{p,\text{ind}}, P_p$	The static, frequency-independent and total power dissipation of processor u_p
E_{static}	The static energy consumption of the system
$E_{p,d}(J_{i,k})$	The dynamic energy consumption completed job $J_{i,k}$ on processor u_p
E_{total}	The total energy consumption required to execute all jobs

3 Models and Problem Statement

3.1 Task and Job Model

In this work, we consider a real-time system, where m periodic tasks $T = \{\tau_1, \tau_1, \ldots, \tau_m\}$ are scheduled. A Directed Acyclic Graph $G = (T, E)$ is used to capture the data dependency of m tasks, where $E = \{e_{i,j} \mid \tau_i, \tau_j \in T\}$ denotes the data dependency relationships between the these tasks. For any two tasks τ_i and τ_j, $1 \leq i, j \leq m$, there are two relationships:

(1) $\exists\ e_{i,j} \notin E$ indicates that there is no data dependency between tasks τ_i and τ_j. This means that tasks τ_i and τ_j can be executed in parallel.
(2) $\exists\ e_{i,j} \in E$ indicates that task τ_i is a predecessor of task τ_j, and task τ_j is a successor of task τ_i. In this case, the tasks must be executed sequentially. The communication data size for transmitting the processing results from task τ_i to τ_j is defined as $data_{i,j}$.

Each task $\tau_i \in T$, which releases jobs according to its period, is characterized by a tuple

$$\tau_i = (r_{i,1}, w_i, d_i, t_i), \tag{1}$$

where $r_{i,1}$ is the release time of the first job of task τ_i, w_i represents the computation workload (CPU-cycles), d_i is the related deadline, and t_i is the period, with $d_i \leq t_i$. An example of a relationship graph depicting 7 periodic tasks with their dependencies is given in Fig. 1. The parameters of these periodic tasks in Fig. 1 are shown in Table 1.

For the job model, $\boldsymbol{J}_i = \{J_{i,1}, J_{i,2}, \ldots, J_{i,HP/t_i}\}$ is denoted to represent the jobs activated by task τ_i within a hyper-period $HP = lcm(t_1, t_2, \ldots, t_m)$. If task τ_i is a predecessor or successor of task τ_j, i.e., $\exists\ e_{i,j} \in E$, job $J_{i,k} \in \boldsymbol{J}_i$ is very likely to be a predecessor or successor of job $J_{j,k'} \in \boldsymbol{J}_j$. For convenience, we denote the sets of predecessors and successors of $J_{i,k}$ as $pre(J_{i,k})$ and $suc(J_{i,k})$, respectively. A job with no predecessors is referred to as an entry job J_{entry}. Similarly, a job with no successors is referred to as an exit job J_{exit}. We use $\boldsymbol{J}_{\text{entry}}$ and $\boldsymbol{J}_{\text{exit}}$ to represent the set of all entry and exit jobs, respectively.

$$\begin{cases} \boldsymbol{J}_{\text{entry}} = \{J_{i,k} \mid J_{i,k} \in \boldsymbol{J},\ |pre(J_{i,k})| = 0\}, \\ \boldsymbol{J}_{\text{exit}} = \{J_{i,k} \mid J_{i,k} \in \boldsymbol{J},\ |suc(J_{i,k})| = 0\}. \end{cases} \tag{2}$$

For any periodic task τ_i, we denote the k-th job generated by task τ_i as

$$J_{i,k} = (r_{i,k}, w_i, d_{i,k}), 1 \leq k \leq HP/t_i, \tag{3}$$

where $r_{i,k}$, w_i, and $d_{i,k}$ represent the release time, computation workload, and deadline of job $J_{i,k}$, respectively. To represent job dependencies, we introduce a job directed acyclic graph (Job-DAG)

$$F = (\boldsymbol{J}, E_{job}, D), \tag{4}$$

where $\boldsymbol{J} = \{\boldsymbol{J}_1 \cup \boldsymbol{J}_2 \cup \cdots \cup \boldsymbol{J}_m\}$ denotes the set of all jobs, E_{job} is the set of directed edges indicating the dependencies between jobs, and D represents the deadline of the Job-DAG F. In this work, the communication data size for transmitting the processing results from jobs $J_{i,k} \in \boldsymbol{J}_i$ to $J_{j,k'} \in \boldsymbol{J}_j$ is equal to $data_{i,j}$. Key notations used in this paper are summarized in Table 2.

3.2 Processor Model

This work considers a heterogeneous multiprocessor system $U = \{u_1, u_2, \ldots, u_{|U|}\}$, where $|U|$ is the size of the set U. Without loss of generality, each processor $u_p \in U$ is assumed to be supporting the DVFS, and has a set of available frequencies $F_p = \{f_{p,1}, f_{p,2}, \ldots, f_{p,|F_p|}\}$, in which $f_{p,1}$ and $f_{p,|F_p|}$ stand for the minimal and maximal values, respectively. Due to the heterogeneity property, each processor has a different processing capability for each task. An $m \times |U|$ matrix $V = \{v_{i,p} \mid 1 \leq i \leq m, 1 \leq p \leq |U|\}$ represents the processing capabilities

of processors operating at their maximum frequencies. Each element $v_{i,p} \in V$ denotes the execution speed at which task τ_i would be executed on processor u_p, and is directly proportional to the processor's frequency $f_p \in F_p$, namely, $v_{i,p} \propto f_p$. Therefore, the execution time of task τ_i on processor u_p operating at frequency f_p can be calculated as $w_i/(v_{i,p} \times f_p)$. For convenience, we normalize the frequencies such that $f_{p,|F_p|} = 1$. For easy understanding, we present an example of a system comprising three heterogeneous processors, with each processor's processing capability and frequency detailed in Table 3.

The processors are fully connected via point-to-point communication links, and the set $B = \{b_{p,p'} \mid u_p, u_{p'} \in U\}$ denotes the available bandwidths between processors. In general, the communication cost between any two tasks depends not only on the amount of data transferred but also on the bandwidth between processors on which the tasks are executed. If two tasks are assigned to the same processor, their data transmission time is assumed to be zero. When tasks τ_i and $\tau_j \in suc(\tau_i)$ are executed on distinct processors u_p and $u_{p'}$, respectively. Given $data_{i,j}$ and $b_{p,p'}$, the communication cost $c_{i,j}$ between tasks τ_i and τ_j can be determined as

$$c_{i,j} = \begin{cases} 0 & \text{if } u_p = u_{p'}, \\ L_p + data_{i,j}/b_{p,p'} & \text{otherwise.} \end{cases} \tag{5}$$

where L_p represents the communication startup cost of processor u_p. For the convenience of explanation, when transferring a data of size *data* from one processor to another processor, the communication cost c is expressed as the average communication cost, which is given by $c = \overline{L} + data/\overline{B}$, where $\overline{L}$ represents the average communication start-up time across all processors, $\overline{L} = 0$, and $\overline{B}$ denotes the average transfer rate of the links connecting the processors, $\overline{B} = 20$.

3.3 Job Execution Model

According to the job model, there may be multiple entry and exit jobs. In this work, the minimum start (or maximum finish) time of the entry (or exit) jobs is defined as the start (or finish) time of Job-DAG F. Note that each job $J_{i,k} \in \boldsymbol{J}$ must meet the following four conditions to start execution on processor $u_p \in U$:

(1) The job $J_{i,k}$ has been released at time $r_{i,1} + (k-1) \times t_i$;
(2) All predecessors of job $J_{i,k}$ have been completed;
(3) The processor u_p has available resources, that is, it is free;
(4) For any two consecutive jobs generated by a task, the earlier job must complete before the later one begins.

Some common attributes are defined as follows.

Definition 3.1. $u(J_{i,k})$ denotes the processor assigned to job $J_{i,k}$.

Definition 3.2. $AT(J_{i,k})$ represents the arrival time of job $J_{i,k}$, which can be obtained by adding an offset of $k-1$ periods to $r_{i,1}$, namely,

$$AT(J_{i,k}) = r_{i,1} + (k-1) \times t_i. \tag{6}$$

Definition 3.3. $ST_p(J_{i,k})$ represents the start time of job $J_{i,k}$ on processor u_p, and is given by

$$ST_p(J_{i,k}) = \max\left\{ AT(J_{i,k}),\ free(u_p),\ \max_{J_{j,k'} \in pre(J_{i,k})} \left\{ FT_{u(J_{j,k'})}(J_{j,k'}) + c_{j,i} \right\} \right\}. \tag{7}$$

If job $J_{i,k}$ is J_{entry}, $ST(J_{entry}) = \max\left\{AT(J_{entry}),\ free(u_p)\right\}$. $free(u_p)$ is the free time of processor u_p, and its initial value is zero.

Definition 3.4. $FT_p(J_{i,k})$ denotes the finish time of job $J_{i,k}$ executed on processor u_p with operating frequency $f_p \in F_p$, and is defined as

$$FT_p(J_{i,k}) = ST_p(J_{i,k}) + w_i/(v_{i,p} \times f_p). \tag{8}$$

Table 3. Parameters of the processors used to execute the tasks are shown in Fig. 1

	v_p	$P_{p,s}$	$P_{p,\text{ind}}$	C_p	α_p	$\{f_{p,1}, f_{p,2}, \ldots, f_{p,\lvert F_p\rvert}\}$
u_1	1.1	0.01	0.05	1.0	2.5	{0.1, 0.11, ..., 1.0}
u_2	1.2	0.01	0.03	0.9	2.6	{0.1, 0.11, ..., 1.0}
u_3	1.3	0.01	0.04	0.8	2.7	{0.1, 0.11, ..., 1.0}

Definition 3.5. $M(F)$ is defined as the makespan or scheduling length for the Job-DAG F, measured from the minimum start time of the entry jobs to the maximum finish time of the exit jobs, and is given by

$$M(F) = \max_{J_{exit} \in \boldsymbol{J}_{\text{exit}}} \left\{ FT(J_{exit}) \right\} - \min_{J_{entry} \in \boldsymbol{J}_{\text{entry}}} \left\{ AT(J_{entry}) \right\}. \tag{9}$$

3.4 Energy Model

During the active operation of a DVFS-available processor $u_p \in U$, its power includes *static power* ($P_{p,s}$) and *dynamic power* ($P_{p,d}$), where the latter includes *frequency-dependent* and *frequency-independent* ($P_{p,\text{ind}}$) [27–29]. When processor u_p operates at a frequency $f_p \in F_p$, its total power dissipation is given by the following equation:

$$P_p = P_{p,s} + P_{p,\text{ind}} + C_p \times (f_p)^{\alpha_p}, \tag{10}$$

where C_p represents the effective switching capacitance and α_p is the dynamic power exponent, both of which are constants specific to each processor. For easy understanding, the power parameters of the given system example are shown in Table 3.

The energy consumption of a processor is the product of power and time. After all jobs are executed, the static energy consumption required by all processor is given by

$$E_{\text{static}} = \sum_{p=1}^{|U|} P_{p,s} M(F). \tag{11}$$

The dynamic energy consumption required to complete job $J_{i,k}$ on processor u_p with operating frequency f_p is expressed as

$$E_{p,d}(J_{i,k}) = \left(P_{p,\text{ind}} + C_p (f_p)^{\alpha_p} \right) \times \left(FT_p(J_{i,k}) - ST_p(J_{i,k}) \right). \tag{12}$$

In summary, the total energy consumption E_{total} required to execute all jobs in $\boldsymbol{J}$ is determined by

$$E_{\text{total}} = E_{\text{static}} + \sum_{J_{i,k} \in \boldsymbol{J}} E_{p,d}(J_{i,k}). \tag{13}$$

3.5 Problem Statement

Based on the above models, the problem is formulated as follows: Given a fully connected platform comprising a set of heterogeneous processors $U = \{u_1, u_2, \ldots, u_{|U|}\}$ and a collection of m non-preemptive and data-dependent tasks $T = \{\tau_1, \tau_1, \ldots, \tau_m\}$, the objective is to develop a scheduling strategy that minimizes total energy consumption while ensuring that all jobs released by tasks within a hyper-period $HP = lcm(t_1, t_2, \ldots, t_m)$ meet their data dependencies and deadline constraints.

$$Minimize \quad E_{\text{total}}, \tag{14}$$

subject to

$$\begin{cases} M(F) \le D; \\ ST(J_{i,k}) \ge AT(J_{i,k}), 1 \le k \le HP/t_i, 1 \le i \le m; \\ ST(J_{i,k}) \ge FT(J_{i,k-1}), \text{ if } k > 1; \\ FT(J_{i,k}) \le AT(J_{i,k}) + d_{i,k}; \\ f_p \in F_p = \{f_{p,1}, f_{p,2}, \ldots, f_{p,|F_p|}\}, 1 \le p \le |U|. \end{cases}$$

4 Precedence Constraint Analysis

In this section, we analyze the precedence constraint for all jobs $\boldsymbol{J}$. This also directly determines the structure of the Job-DAG F. In order to ensure task $\tau_j \in suc(\tau_i)$ receives all data produced by task τ_i without any data being lost or duplicated, the following conditions for each job $J_{i,k} \in \boldsymbol{J}$ must be satisfied.

(1) Satisfy the reachability of each job to the unique exit job;
(2) Each job $J_{i,k} \in \boldsymbol{J}_i$ has at least one successor $J_{j,k'}$ released from module J_j, that is,

$$|suc(J_{i,k}) \cap \boldsymbol{J}_j| \ge 1; \tag{15}$$

(3) Each job $J_{j,k'} \in \boldsymbol{J}_j$ has at least one predecessor $J_{i,k}$ released from module J_i, namely,

$$|pre(J_{j,k'}) \cap \boldsymbol{J}_i| \ge 1. \tag{16}$$

Therefore, the value range of $|E_{job}|$ can be summarized as

$$\sum_{e_{i,j} \in E} \max\{HP/t_i, HP/t_j\} \le |E_{job}| \le \sum_{e_{i,j} \in E} HP/t_i \times HP/t_j. \tag{17}$$

In DAG task scheduling, parallelism refers to the number of tasks that can be started simultaneously without violating dependencies. Therefore, a DAG with fewer dependencies has a relatively high parallelism. The example of converting the DAG G in Fig. 1 into a Job-DAG F is shown in Fig. 2, which contains 15 jobs and 34 dependencies. Obviously, this structure of the Job-DAG is very complex and is not conducive to the efficient execution. Here, a reasonable optimization of the DAG structure is essential for improving the parallelism and execution efficiency of jobs. Chen et al. [30] investigated the periodicity of real-time schedules for dependent periodic tasks and proposed an analysis method for precedence constraints.

To make the job have fewer dependencies, we set $|suc(J_{i,k}) \cap \boldsymbol{J}_j| = 1$ and $|pre(J_{j,k'}) \cap \boldsymbol{J}_i| = 1$, as shown in Eqs. (15) and (16). Additionally, we ensure that each job $J_{j,k'}$ $(1 \le k' \le HP/t_j)$ receives exactly t_j/t_i data produced

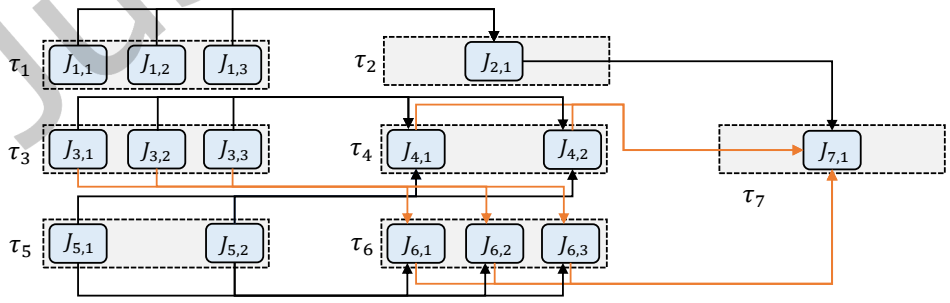


Fig. 2. The Job-DAG, corresponding to the DAG in Fig. 1, consists of 15 jobs and 34 dependencies.

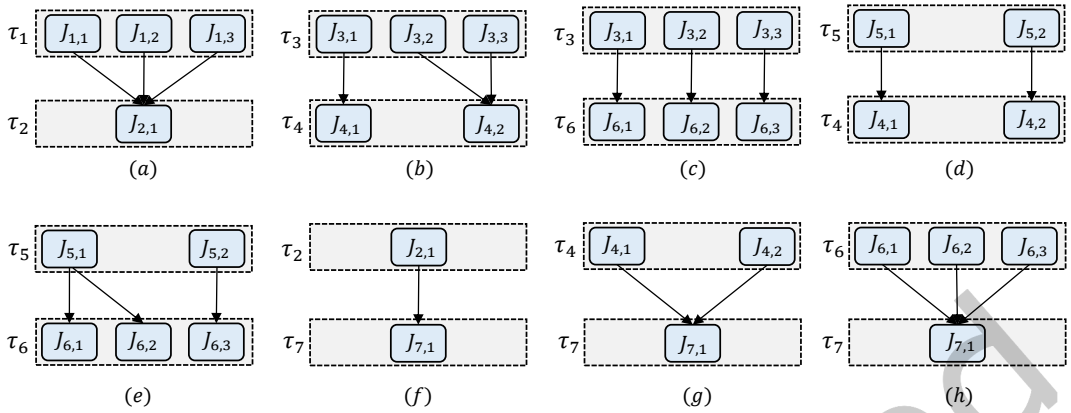


Fig. 3. 8 cases of removing dependencies between jobs.

by the jobs from $J_{i,(k'-1)\times t_j/t_i+1}$ to $J_{i,k'\times t_j/t_i}$. Therefore, the predecessor set of each job $J_{j,k'} \in \boldsymbol{J}$ can be updated to

$$pre(J_{j,k'}) = \left\{ J_{i,k} \mid (k'-1) \times t_j/t_i + 1 \le k \le k' \times t_j/t_i, \tau_i \in pre(\tau_j) \right\}. \tag{18}$$

Based on Eq. (18), the successor set of each job $J_{j,k'} \in \boldsymbol{J}$ is $suc(J_{j,k'}) = \left\{ J_{i,k} \mid J_{j,k'} \in pre(J_{i,k}) \right\}$.

Example description. According to Fig. 1, the periods $t_1, t_2, \ldots, t_m \in \{20, 30, 60\}$. The hyper-period $HP = lcm\{t_1, t_2, \ldots, t_m\} = 60$. Next, we simplify the dependencies between jobs $\boldsymbol{J}_1$ and $\boldsymbol{J}_2$ ($e_{1,2} \in E$), as well as between jobs $\boldsymbol{J}_3$ and $\boldsymbol{J}_4$ ($e_{3,4} \in E$), as follows.

(1) For the edge $e_{1,2} \in E$, $t_1 = 20$ and $t_2 = 60$. When $k' = 1$, $(1-1) \times 60/20 + 1 \le k \le 1 \times 60/20$, making the predecessors of job $J_{2,1}$ the jobs $J_{1,1}$, $J_{1,2}$, and $J_{1,3}$, as illustrated in Fig. 3(a).
(2) For the edge $e_{3,4} \in E$, with $t_3 = 20$ and $t_4 = 30$, the analysis is as follows. When $k' = 1$, $(1-1) \times 30/20 + 1 \le k \le 1 \times 30/20$, making the predecessor of job $J_{4,1}$ the job $J_{3,1}$. When $k' = 2$, $(2-1) \times 30/20 + 1 \le k \le 2 \times 30/20$, making the predecessors of job $J_{4,2}$ the jobs $J_{3,2}$ and $J_{3,3}$. This is illustrated in Fig. 3(b).

The dependencies between other jobs can be handled similarly. Finally, we obtain the structure diagrams after simplifying the edges $e_{3,6}$, $e_{5,4}$, $e_{5,6}$, $e_{2,7}$, $e_{4,7}$, and $e_{6,7}$, as shown in Fig. 3(c), (d), (e), (f), (g), and (h), respectively. The simplified Job-DAG is shown in Fig. 4, which contains 20 dependencies. Compared with the unsimplified Job-DAG in Fig. 2, 14 dependencies are removed.

5 Energy-Efficient Scheduling

Based on the precedence constraint analysis in Section 4, we have determined the dependencies between jobs, that is, constructed a certain Job-DAG $F = (\boldsymbol{J}, E_{job}, D)$. In this section, we will study the energy-efficient scheduling strategy without violating resource, precedence, and deadline constraints. This section is mainly divided into two stages: DVFS independent scheduling design and DVFS weakly dependent energy-saving design. The former focuses on job priority calculation and processor allocation to reduce time and energy consumption costs, while the latter primarily aims to further reduce energy consumption by minimizing the processor's operating frequency as much as possible, all while meeting the deadline constraints.

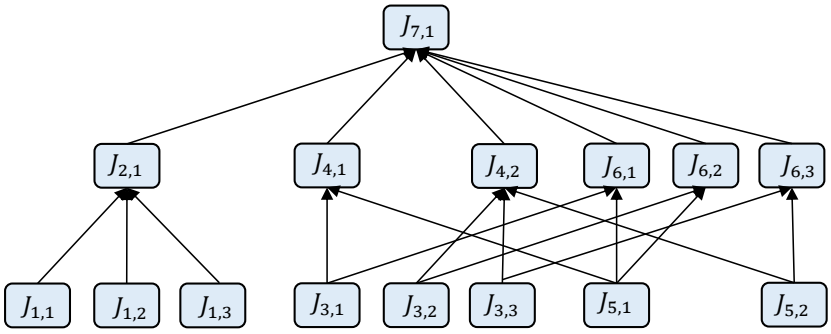


Fig. 4. The simplified Job-DAG, consists of 15 jobs and 20 dependencies.

5.1 DVFS-independent Scheduling Design

Time and energy are two crucial performance metrics in multi-processor heterogeneous systems. The less time and energy consumption required to complete a Job-DAG, the higher the system performance. A scheduling strategy usually includes two primary processes: determining the scheduling order and allocating processors. The former indicates the execution order of jobs, and the latter determines on which processor the job is executed. This section will explain these two processes in separate subsections.

5.1.1 Job Priority Calculation. The job priority determines the scheduling order of jobs, significantly influencing overall scheduling outcomes [31]. Existing research typically assigns a static priority to each job, which has the advantage of allowing the scheduling orders of all jobs to be predetermined, making it more suitable for fixed jobs. As described in [4, 17], the static priority $rank_u(J_{i,k})$ for each job $J_{i,k}$ is calculated recursively from the exit job to the entry job. However, in our job model, each job has a specified release time and a related deadline. This means that each job must be released before it can be executed, even if it has a high priority. Consequently, static priority allocation methods such as $rank_u(J_{i,k})$ are not applicable to our job model. Therefore, it is essential to develop a dynamic priority assignment strategy.

The priority of a job reflects its urgency to be processed, which is typically measured by how close it is to its deadline. The closer the finish time of job $J_{i,k}$ is to its deadline, the more urgent it is to execute job $J_{i,k}$. According to the release time and relative deadline of job $J_{i,k}$, the deadline of job $J_{i,k}$ can be defined as

$$D(J_{i,k}) = AT(J_{i,k}) + d_{i,k}, 1 \leq k \leq HP/t_i. \tag{19}$$

In addition, the calculation of the finish time of job $J_{i,k}$ requires obtaining its start time in advance. Note that the start time of job $J_{i,k}$ depends on the finish times of its predecessor jobs $pre(J_{i,k})$. Therefore, we only consider the priority calculation of jobs whose predecessors have completed. If these jobs have been released, they are described as ***ready jobs*** and added to a set $\boldsymbol{Q}$. To ensure that job $J_{i,k}$ in $\boldsymbol{Q}$ gains higher priority as its deadline approaches, we define its dynamic priority as follows.

Definition 5.1. The priority of job $J_{i,k} \in \boldsymbol{Q}$ is defined as the time interval from the average finish time of this job to its deadline $D(J_{i,k})$, and is given by

$$P(J_{i,k}) = \underbrace{\frac{1}{|U|} \times \sum_{p=1}^{|U|} FT_p(J_{i,k})}_{\text{The average finish time}} - \underbrace{\Big(AT(J_{i,k}) + d_{i,k}\Big)}_{\text{The job deadline}}, J_{i,k} \in \boldsymbol{Q}. \tag{20}$$

After each ready job is scheduled, the system will update the elements in set $\boldsymbol{Q}$ in real time. Each time, the priority of each job $J_{i,k}$ in $\boldsymbol{Q}$ is calculated based on Eq. (20), and the job $J_{i,k}$ with the highest priority is then popped for execution. Considering that the processor's free time is variable, the finish times of all jobs in $\boldsymbol{Q}$ will also change, thereby achieving dynamic calculation of job priority. We introduce the ready job set $\boldsymbol{Q}$, all of which are in the ready state, meaning that the job has been released and its predecessor jobs have been completed. No matter which job is selected from $\boldsymbol{Q}$ for scheduling, it will not violate the dependency constraints between jobs. This also holds true for Definition 5.1.

5.1.2 Processor Assignment . Energy-efficient computing is usually achieved by using DVFS to balance time performance and energy consumption. However, in many critical systems, frequent use of DVFS is not allowed. Therefore, we introduce a scheduling strategy independent of DVFS to achieve energy saving. To ensure fair comparison and eliminate the scale effect between execution time and energy consumption, both metrics are standardized to a dimensionless form using min-max normalization. This allows for a balanced multi-objective trade-off and facilitates subsequent optimization. We normalize time and energy as follows.

Definition 5.2. **Normalized Finish Time (NFT)** represents the ratio of the finish time of job $J_{i,k}$ to the deadline D of the Job-DAG $F = (\boldsymbol{J}, E_{job}, D)$, namely,

$$NFT_p(J_{i,k}) = \frac{1}{D} \times FT_p(J_{i,k}). \tag{21}$$

Definition 5.3. **Normalized Energy Consumption (NEC)** is the ratio of the energy consumption of job $J_{i,k}$ to the average dynamic energy consumption $\overline{EC_d}$ when all jobs are executed on a single processor, that is,

$$NEC_p(J_{i,k}) = \frac{1}{\overline{EC_d}} \times \Big(P_{p,d} \times \Big(FT_p(J_{i,k}) - ST_p(J_{i,k})\Big) + E_{\text{current},d}\Big), \tag{22}$$

where

$$\overline{EC_d} = \frac{1}{|U|} \times \sum\nolimits_{p=1}^{|U|} \sum\nolimits_{J_{i,k} \in \boldsymbol{J}} E_{p,d}(J_{i,k}), \tag{23}$$

$$E_{\text{current},d} = E_{\text{current},d} + P_{u(J_{i,k}),d} \times \Big(FT_{u(J_{i,k})}(J_{i,k}) - ST_{u(J_{i,k})}(J_{i,k})\Big). \tag{24}$$

The initial value of $E_{\text{current},d}$ is 0. Eq. (24) is an accumulation method. The processor $u(J_{i,k})$ for job $J_{i,k}$ is obtained by eq. (26).

Processor assignment strategy. Based on Definitions 5.2 and 5.3, we employ the way of weighted sum of *NFT* and *NEC* to mapping jobs to processors, calculated as follows.

$$\varphi_p(J_{i,k}) = \lambda \times NFT_p(J_{i,k}) + (1 - \lambda) \times NEC_p(J_{i,k}), 0 \le \lambda \le 1, \tag{25}$$

where the parameter λ serves as a balancing factor between time and energy. When $\lambda = 1$, the processor is selected solely based on the earliest finish time, and when $\lambda = 0$, it is selected purely based on the lowest energy consumption. Obviously, the value of λ determines both the finish time and energy overheads. Therefore, a proper value for λ requires to be solved. A natural intuition for λ is that as its value decreases, the finish time increases while the energy consumption decreases. This intuition generally holds, but fluctuations can arise due to the nonlinear relationship between finish time and energy consumption. To obtain a proper λ, we use an iterative method that decreases λ from 1 to 0 with a step $\Delta\lambda$. For a given λ, the $\varphi_p(J_{i,k})$ value varies across different processors. Since both finish time and energy consumption should be minimized, for a job $J_{i,k}$, we naturally assign it to the processor $u(J_{i,k}) \in U$ that yields the minimum $\varphi(J_{i,k})$ value, namely,

$$\varphi_{u(J_{i,k})}(J_{i,k}) = \min\Big\{\varphi_p(J_{i,k}) \mid u_p \in U\Big\}, u(J_{i,k}) \in U. \tag{26}$$

Algorithm 1 The Time and Energy Balance Scheduling (TEBS) Algorithm

Input: $F = (\boldsymbol{J}, E_{job}, D)$, U, $\Delta\lambda$, $HP = lcm(t_1, t_2, \ldots, t_m)$
Output: E_{total}, $M(F)$
1: $\lambda' = \lambda = 1$, $M(F) = 0$, $\boldsymbol{\varphi}' = \infty$;
2: Create a set $\boldsymbol{Q}$ to save the ready jobs;
3: **while** $\lambda \geq 0$ **do**
4: Set tab = True;
5: **while** $\boldsymbol{Q}$ is not empty **do**
6: Calculate $P(J_{i,k})$ for each job $J_{i,k}$ in $\boldsymbol{Q}$ by Eq. (20);
7: Pop job $J_{i,k}$ with the highest priority from $\boldsymbol{Q}$;
8: Assign job $J_{i,k}$ to processor $u(J_{i,k})$ with the minimal $\varphi_{u(J_{i,k})}(J_{i,k})$ by Eq. (25);
9: Update the set $\boldsymbol{Q}$; // *add the new ready jobs to* $\boldsymbol{Q}$
10: **if** $FT(J_{i,k}) \geq D(J_{i,k})$ **then**
11: tab = False;
12: **end if**
13: **end while**
14: Obtain the $M(F)$ and $E_{\text{total},d}$, and calculate $\boldsymbol{\varphi}$ by Eq (27), respectively;
15: **if** $\boldsymbol{\varphi}' \geq \boldsymbol{\varphi}$ **and** $M(F) \leq D$ **and** tab == True **then**
16: $\boldsymbol{\varphi}' = \boldsymbol{\varphi}$, $\lambda' = \lambda$;
17: **else**
18: break;
19: **end if**
20: $\lambda = \lambda - \Delta\lambda$;
21: **end while**
22: Repeat lines 5-14 based on λ';

5.1.3 The TEBS Algorithm. The objective of this section is to optimize both time and energy costs by devising a DVFS-independent scheduling strategy. Reducing time costs helps achieve rational resource allocation, thereby enhancing resource utilization and parallel processing efficiency. Meanwhile, reducing energy consumption helps cut operational costs and boost system efficiency. In this section, we adopt the sum of NFT and NEC as the optimization objective, denoted by $\boldsymbol{\varphi}$.

$$\boldsymbol{\varphi} = \frac{M(F)}{D} + \frac{E_{\text{total},d}}{\overline{EC}_d}. \tag{27}$$

Therefore, the DVFS-independent scheduling problem ($\mathbf{P}_1$) investigated in this section can be formally stated as

$$\mathbf{P}_1 : \ Minimize \quad \boldsymbol{\varphi}, \tag{28}$$

subject to

$$\begin{cases} M(F) \leq D; \\ FT(J_{i,k}) \leq D(J_{i,k}); \\ ST(J_{i,k}) \geq AT(J_{i,k}). \end{cases}$$

Finally, we take the value of λ that yields the minimal value of Eq. (28) as the proper value of λ. Note that the Job-DAG $F = (\boldsymbol{J}, E_{job}, D)$ is requested to be completed within the deadline D, and each job $J_{i,k} \in \boldsymbol{J}$ is required to be completed within its deadline $D(J_{i,k})$. As mentioned above, we introduce the proposed TEBS (Time and Energy Balance Scheduling) approach in Algorithm 1 to achieve the DVFS-independent energy-efficient scheduling. In

Table 4. The schedule generated by each step of the TEBS for the simplified Job-DAG in Fig. 4

Step	Job	All ready jobs in $\boldsymbol{Q}$	$u(J_{i,k})$	$ST(J_{i,k})$	$FT(J_{i,k})$	$E_{p,d}(J_{i,k})$	$E_{\text{total},d}$
1	$J_{3,1}$	$\{J_{1,1}, J_{3,1}, J_{5,1}\}$	u_3	0	3.846	3.231	3.231
2	$J_{1,1}$	$\{J_{1,1}, J_{3,2}, J_{5,1}\}$	u_2	0	3.333	3.1	6.331
3	$J_{3,2}$	$\{J_{1,2}, J_{3,2}, J_{5,1}\}$	u_3	20	23.846	3.231	9.562
4	$J_{1,2}$	$\{J_{1,2}, J_{3,3}, J_{5,1}\}$	u_2	20	23.333	3.1	12.662
5	$J_{5,1}$	$\{J_{1,3}, J_{3,3}, J_{5,1}\}$	u_1	0	5.455	5.727	18.389
6	$J_{6,1}$	$\{J_{1,3}, J_{3,3}, J_{4,1}, J_{5,2}, J_{6,1}\}$	u_1	7.205	10.841	3.818	22.207
7	$J_{4,1}$	$\{J_{1,3}, J_{3,3}, J_{4,1}, J_{5,2}\}$	u_1	10.841	17.205	6.682	28.889
8	$J_{3,3}$	$\{J_{1,3}, J_{3,3}, J_{5,2}\}$	u_3	40	43.846	3.231	32.12
9	$J_{6,2}$	$\{J_{1,3}, J_{5,2}, J_{6,2}\}$	u_2	25.596	28.929	3.1	35.22
10	$J_{1,3}$	$\{J_{1,3}, J_{5,2}\}$	u_2	40	43.333	3.1	38.32
11	$J_{2,1}$	$\{J_{2,1}, J_{5,2}\}$	u_2	43.333	47.5	3.875	42.195
12	$J_{5,2}$	$\{J_{5,2}\}$	u_1	30	35.455	5.727	47.922
13	$J_{4,2}$	$\{J_{4,2}, J_{6,3}\}$	u_3	43.846	49.231	4.523	52.445
14	$J_{6,3}$	$\{J_{6,3}\}$	u_1	45.596	49.233	3.818	56.263
15	$J_{7,1}$	$\{J_{7,1}\}$	u_3	51.333	53.64	1.938	58.202

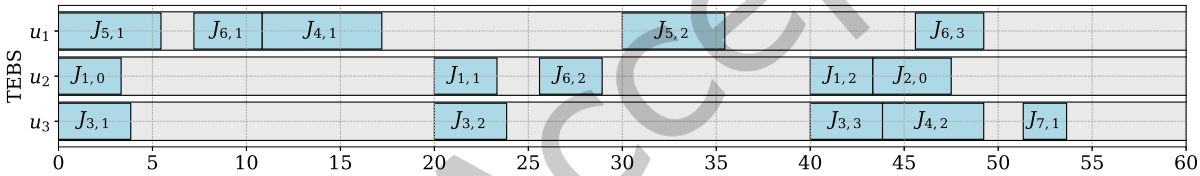


Fig. 5. Gantt chart show the scheduling results of the simplified Job-DAG in Fig. 4 by the TEBS algorithm.

line 2, create a set $\boldsymbol{Q}$ to save the entry jobs in $\boldsymbol{J}_{\text{entry}}$. Since the size of the set $\boldsymbol{Q}$ changes in real time, we set its upper bound to

$$|\boldsymbol{J}| = \sum_{i=1}^{m} HP/t_i, \tag{29}$$

(i.e., $|Q| \leq |\boldsymbol{J}|$). Lines 4 to 9 describe the scheduling process: each time, the highest-priority job $J_{i,k}$ is selected from the set $\boldsymbol{Q}$ and assigned to the processor $u_p \in U$ with the minimal $\varphi_p(J_{i,k})$ for execution. The **if** condition in line 12 is used to store the scheduling result with the minimal $\boldsymbol{\varphi}$ that meets the deadline constraint, along with the corresponding value of λ.

Time complexities analysis. Both lines 3 and 5 list two **while** cycle, which are executed $1/\Delta\lambda$ and $|\boldsymbol{J}|$ times, respectively. Both lines 5 and 6 traverse all elements in $\boldsymbol{Q}$, with the number of traversals being $|Q|$. Line 6 needs to traverse all processors to find the minimal $\varphi(J_{i,k})$ for job $J_{i,k}$, which requires $|U|$ times. Line 8 needs to iterate all jobs twice to update $\boldsymbol{Q}$, resulting in $|\boldsymbol{J}|^2$ executions. Considering $1/\Delta\lambda$ is a constant, the time complexity of the TEBS is $O(|\boldsymbol{J}||U| + |\boldsymbol{J}|^3)$.

Example 1. This example demonstrates how Algorithm 1, namely the TEBS algorithm, schedules the simplified Job-DAG in Fig. 4. We set $\Delta\lambda = 0.01$, and the TEBS ultimately yields $\lambda' = 0.05$, with a makespan of 53.64 and a total dynamic energy consumption of 58.202. The TEBS iteratively selects jobs to optimize energy consumption while meeting deadline requirements. Table 4 provides a detailed breakdown of the scheduling process, consisting of 15

steps. Each step involves selecting the highest priority job from the ready job set $\boldsymbol{Q}$, determining the processor $u(J_{i,k})$, calculating the start time $ST(J_{i,k})$, finish time $FT(J_{i,k})$, dynamic energy consumption $E_{p,d}(J_{i,k})$, and the system's current total dynamic energy consumption $E_{\text{total},d}$. For instance, in Step 1, job $J_{3,1}$ is selected from $\boldsymbol{Q}$, which includes $J_{1,1}, J_{3,1}, J_{5,1}$. It is assigned to processor $u(J_{3,1}) = u_3$, starting at time 0 and ending at time 3.846, consuming 3.231 units of energy, contributing to the total energy consumption of 3.231 for this step. To visualize the scheduling process, Fig. 5 illustrates the scheduling results of the TEBS using a Gantt chart. The chart visually represents the start and finish times of each job across different processors. Each bar in the chart corresponds to a job, with its length indicating the duration of the job and its position showing processor allocation.

5.2 DVFS-weakly Dependent Energy-saving Design

5.2.1 Available Frequency Analysis. While DVFS-independent scheduling can reduce energy consumption, its effectiveness is limited. To achieve greater energy savings, lowering processor frequency is necessary. Since frequent use of DVFS should be avoid, we propose a DVFS-weakly dependent approach. The energy consumption of a processor depends on its power and the job load it handles. According to [27], when $|\boldsymbol{J}|$ jobs assigned to the same processor u_p must be completed within a given time, the energy consumption is minimized if all jobs are executed at the same frequency. However, this minimum allowed frequency of processor u_p needs to be analyzed in detail before adjusting the processor operating frequency f_p. The following three cases are divided.

(1) The processor u_p provides the minimum available frequency, namely, $f_{p,1} \leq f_p$.
(2) According to the inherent power parameters of the processor u_p, there is a minimum dynamic energy-efficient frequency $f_{p,ee}$. The specific calculation process can be found in Ref. [27].

$$f_{p,ee} = \sqrt[\alpha_p]{P_{ind,p}/(C_p \times (\alpha_p - 1))} \leq f_p.$$

(3) Considering that each job $J_{i,k}$ has a related deadline $d_{i,k}$, when reducing the operating frequency of processor u_p, it is necessary to ensure that all jobs assigned to this processor can be completed within their related deadlines.

$$f_{p,dl} = \max_{J_{i,k} \in \boldsymbol{J} \& u(J_{i,k}) = u_p} \left\{ \frac{w_i}{v_{i,k} \times d_{i,k}} \right\} \leq f_p.$$

5.2.2 The DWDES Algorithm. Reducing processor power within the acceptable time range can significantly save energy. However, changing the frequency of a processor will inevitably affect the completion time of jobs on other processors due to job dependencies. Therefore, careful selection of processors for power reduction is required to maximize energy savings while meeting time requirements. The research objective of this section is to reduce the overall energy consumption by adjusting the operating frequencies of processors, without involving the scheduling of tasks. Therefore, the DVFS-weakly Dependent Energy-saving problem ($\mathbf{P}_2$) investigated in this section can be formally stated as

$$\mathbf{P}_2: \ \textit{Minimize} \quad E_{\text{total},d}, \tag{30}$$

subject to

$$\begin{cases} M(F) \leq D; \\ FT(J_{i,k}) \leq D(J_{i,k}); \\ ST(J_{i,k}) \geq AT(J_{i,k}); \\ \max\{f_{p,1}, f_{p,ee}, f_{p,dl}\} \leq f_p. \end{cases}$$

To this end, we define an indicator as follows,

$$\Delta E_p = E_{p,d}(f_{p,l}) - E_{p,d}(f_{p,l-1}), 1 \leq l \leq |F_p|. \tag{31}$$

Algorithm 2 The DVFS-Weakly Dependent Energy-Saving (DWDES) Algorithm

Input: $F = (\boldsymbol{J}, E_{job}, D), U, HP = lcm(t_1, t_2, \ldots, t_m)$
Output: E_{total}

```
1:  Call the TEBS to get its job mapping and the makespan M(F);
2:  while M(F) ≤ D do
3:      MaxΔE = 0;
4:      for each processor u_p in the set U do
5:          Calculate E_{p,d}(f_{p,l}) by Eq. (12);
6:          Reduce the frequency of processor u_p by one level, and calculate E_{p,d}(f_{p,l−1}) by Eq. (12);
7:          Recalculate M(F) based on the TEBS's job mapping;
8:          if M(F) ≤ D then
9:              ΔE_p = E_{p,d}(f_{p,l}) − E_{p,d}(f_{p,l−1});
10:             MaxΔE = (ΔE_p ≥ MaxΔE) ? ΔE_p : MaxΔE;
11:         end if
12:         Increase the frequency of processor u_p by one level;
13:     end for
14:     if MaxΔE ≠ 0 then
15:         Select the processor with maximal MaxΔE to decrease its frequency by one level;
16:         Recalculate M(F) based on the TEBS's job mapping;
17:     else
18:         bresk;
19:     end if
20: end while
```

Eq. (31) measures the energy saved when a processor u_p decreases its frequency from level l to level $l-1$. We can decrease the frequency of processors iteratively. Each iteration selects a processor that achieves the maximum ΔE_p to decrease its frequency by one level, until the deadline is violated. After the frequency of all processors is determined, DVFS is used only when switching applications.

Therefore, we call the approach as the DVFS-Weakly Dependent Energy-Saving (DWDES) algorithm, and detail it in Algorithm 2. Line 1 calls the TEBS algorithm to obtain the job assignment and the initial makespan $M(F)$, providing the foundational information for energy consumption optimization. Lines 2-18 enter a **while** loop that iteratively adjusts the processor frequencies, continuing until the makespan $M(F)$ surpasses the deadline D. Line 3 initializes $Max\Delta E$ to 0 at the commencement of each loop iteration for tracking the maximum achievable reduction in energy consumption. Lines 4-13 involve calculating the energy consumption $E_{p,d}(f_{p,l})$ for each processor u_p at its current frequency $f_{p,l}$, followed by determining the new energy consumption $E_{p,d}(f_{p,l-1})$ post frequency reduction, and recalculating the makespan $M(F)$. Should $M(F)$ remain less than or equal to D, the energy consumption difference ΔE_p is computed and $Max\Delta E$ is updated accordingly, before the frequency is reverted. Lines 14-17 stipulate that if $Max\Delta E$ is non-zero, the processor exhibiting the greatest energy consumption difference is selected to have its frequency decreased, followed by a recalculation of $M(F)$.

Time complexities analysis. The **while** loop spanning lines 2 to 18 iterates, with the worst-case scenario involving a number of iterations equal to $|F_p| \times |U|$. Within each iteration, the DWDES algorithm performs frequency reduction operations on each processor, incurring a time complexity of $O(|U|)$. Recalculating the makespan carries a time complexity of $O(|\boldsymbol{J}|)$. Consequently, the total time complexity for operations on all processors is $O(|U| \times |\boldsymbol{J}|)$. The time complexity from lines 14 to 17 is $O(|U| + |\boldsymbol{J}|)$. Therefore, the overall time complexity of the DWDES algorithm is $O(|F_p| \times |U|^2 \times |\boldsymbol{J}|)$.

Table 5. The schedule generated by each step of the DWDES for the simplified Job-DAG in Fig. 4

Step	Job	$u(J_{i,k})$	$ST(J_{i,k})$	$FT(J_{i,k})$	f_p	$E^{\text{old}}_{p,d}(J_{i,k})$	$E^{\text{new}}_{p,d}(J_{i,k})$	$E_{\text{total},d}$
1	$J_{3,1}$	u_3	0	4.049	0.95	3.231	2.982	2.982
2	$J_{1,1}$	u_2	0	6.803	0.49	3.1	1.162	4.144
3	$J_{3,2}$	u_3	20	24.049	0.95	3.231	2.982	7.126
4	$J_{1,2}$	u_2	20	26.803	0.49	3.1	1.162	8.288
5	$J_{5,1}$	u_1	0	9.917	0.55	5.727	2.721	11.009
6	$J_{6,1}$	u_1	11.667	18.279	0.55	3.818	1.814	12.823
7	$J_{4,1}$	u_1	18.279	29.849	0.55	6.682	3.174	15.997
8	$J_{3,3}$	u_3	40	44.049	0.95	3.231	2.982	18.979
9	$J_{6,2}$	u_2	26.803	33.605	0.49	3.1	1.162	20.141
10	$J_{1,3}$	u_2	40	46.803	0.49	3.1	1.162	21.303
11	$J_{2,1}$	u_2	46.803	55.306	0.49	3.875	1.453	22.756
12	$J_{5,2}$	u_1	30	39.917	0.55	5.727	2.721	25.477
13	$J_{4,2}$	u_3	44.049	49.717	0.95	4.523	4.175	29.652
14	$J_{6,3}$	u_1	45.799	52.41	0.55	3.818	1.814	31.465
15	$J_{7,1}$	u_3	57.556	59.985	0.95	1.938	1.789	33.255

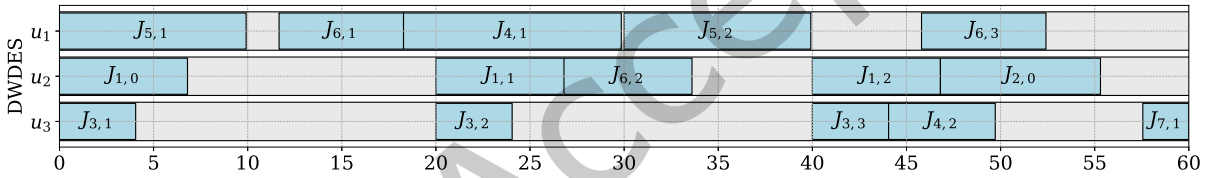


Fig. 6. Gantt chart show the scheduling results of the simplified Job-DAG in Fig. 4 by the DWDES algorithm.

Example 2. Based on **Example 1**, we successfully achieved a significant reduction in energy consumption using the DWDES algorithm, decreasing from the original 58.202 (obtained from the TEBS algorithm) to the current 33.255. So, the energy saving rate is approximately 42.92%. Without compromising the deadlines, we reduced the working frequencies of the processors to 0.55, 0.49, and 0.95, respectively. These results not only demonstrate the theoretical feasibility of our approach but also provide an empirical foundation for future energy optimization research. Table 5 provides The schedule generated by each step of the DWDES algorithm for the simplified Job-DAG in Fig. 4. To visualize the scheduling process, Fig. 6 illustrates the scheduling results of the DWDES algorithm using a Gantt chart.

6 Experimental Results and Discussion

We first perform precedence dependency analysis and simplify the Job-DAG $F = (\boldsymbol{J}, E_{job}, D)$ structure, and then demonstrate the effectiveness of the energy-saving algorithms. This section presents the performance evaluation of our proposed algorithms TEBS and DWDES, including three comparative algorithms, namely IPPTS [18], DNDS [27], and DWDS [27]. All experiments are simulated using Python on a computer with an 11th Gen Intel(R) Core(TM) i7-11800H CPU at 2.30 GHz and 32.0 GB of RAM. We first show the parameter settings of the heterogeneous platform.

Table 6. The parameters of simulation environment

Notation	Parameter	Value
$\|U\|$	The number of processors in a system	{4, 8, 12, 16, 20}
m	The number of modules	10 to 20
w	The task's computational workload	1 to 10 cycles
$data$	The size of data transferred	2 to 8 Bytes
B	The bus bandwidth of in a system	20 kbit/s
$P_{p,s}$	The static power of processor u_p	0.01 to 0.02 W
$P_{p,\text{ind}}$	The frequency-independent dynamic power of processor u_p	0.04 to 0.08 W
f_p	The frequency of processor u_p	0.1 to 1.0 GHz
C	The effective switching capacitance	0.8 to 1.3
α	The dynamic power exponent	2.5 to 3.0
v	The execution capability of processor	1.0 to 2.0 Hz

6.1 Platform Configuration

Considering that different multi-processor heterogeneous systems may be equipped with various computing platforms, our simulation experiments will cover heterogeneous platforms with 4, 8, 12, 16, and 20 processors, i.e., $|U| \in \{4, 8, 12, 16, 20\}$. All processors in one platform are fully connected. When transferring a data of size $data$ from one processor to another processor, the communication cost c is expressed as the average communication cost, which is given by $c = \overline{L} + data/\overline{b}$, where $\overline{L}$ represents the average communication start-up time across all processors, $\overline{L} = 0$ s, and $\overline{b}$ denotes the average transfer rate of the links connecting the processors in the target system, $\overline{b} = 20$ kbit/s. The processor supports the DVFS, with its execution capability being proportional to its frequency. The execution capability of each processor is randomly selected between 1.0 Hz and 2.0 Hz, namely, $1.0 \leq v_p \leq 2.0$ Hz. All frequencies $f_{p,1}, f_{p,2}, \ldots,$ and $f_{p,|F_p|}$ of each processor $u_p \in U$ are discrete with a step size of $\Delta f = 0.01$ GHz, where the maximum frequency is normalized to $f_{p,|F_p|} = 1.0$ GHz. The processor power values are randomly generated within specific ranges as follows: $P_{p,s} = [0.01, 0.02]$ W, $P_{p,\text{ind}} \in [0.04, 0.08]$ W, $C_p \in [0.8, 1.3]$, and $\alpha_p \in [2.5, 3.0]$. The above parameter settings are listed in Table 6.

6.2 Workload Settings

We consider two sets of graphs as workflow: randomly generated application graphs and real-world application graphs (*Gaussian Elimination* [32] and *Fast Fourier Transform* [33]).

6.2.1 Randomly Generated Application Graphs. We use randomly synthesized weighted DAGs as workflows, which are generated using an open-source DAG generation program in [34]. The following parameters are considered to generate random DAGs.

(1) *Number of tasks*: The number of tasks in the DAG is randomly selected from the range $[10, 20]$. This ensures a diverse range of task sizes, capturing both smaller and medium-scale task graphs to enhance the generality and applicability of the experimental results.
(2) *Dependencies between tasks.* To ensure the comprehensiveness and objectivity of the experiments, we use a dependency generation tool from the open-source project in [34] to simulate the DAG $\boldsymbol{G}$ representing the task dependencies. In a DAG, the number of incoming edges to a task is referred to as its *in-degree*, while the number of outgoing edges is its *out-degree*. For this project, the in-degree and out-degree of each

module are selected from the set $\{1, 2, 3, 4, 5\}$, and the probability of adding each dependency is chosen from the set $\{0.3, 0.4, 0.5, 0.6\}$.

(3) *Task and edge properties.* In a DAG, each task τ_i will active a job with computational workload w_i at intervals t_i time, where $w_i \in [1, 10]$ cycles and $t_i \in \{5, 10, 25, 50\}$ s. The size of data transferred between tasks τ_i and τ_j is set to $data_{i,j} \in [2, 8]$ Bytes. Here, $1 \leq i, j \leq m$.

(4) *Deadline Constraints.* We set the deadline of each DAG to be the hyper-period HP. To explore the performance changes of the algorithm under different period spans, 5 different period spans are introduced: $\theta \in \{1.1, 1.2, 1.3, 1.4, 1.5\}$. Furthermore, five different DAG deadlines are introduced, namely, $Deadline = HP = lcm(t_1, t_2, \ldots, t_m)$, where $t_i = \theta \times t_i$.

6.2.2 Real-world Application Graphs. Besides the random DAGs, we also evaluate the performance of the algorithms with two real-world Graphs: *fast Fourier transform* [33] and *Gaussian elimination* [35]. For the Fast Fourier Transform (FFT) graphs, the total number of tasks is determined by the matrix size ρ, and is given by

$$\text{FFT: } m = \rho \times \log_2 \rho + 2 \times \rho - 1, \tag{32}$$

where ρ is a constant, and its value must be a power of 2. When $\rho = 4$, $m = 15$. Another commonly used graph type is the Gaussian Elimination (GE) graphs, which, like Fast Fourier Transform graphs, follows a fixed structure. For Gaussian Elimination graphs, the total number of tasks is determined by the matrix size ρ, and is defined as follows:

$$\text{GE: } m = (\rho^2 + \rho - 2)/2. \tag{33}$$

When $\rho = 5$, $m = 14$. The task properties, edge properties, and deadline constraints of the FFT and GE DAGs are aligned with those of the random DAGs. The specific structures of the FFT and GE DAGs, corresponding to $\rho = 4$ and $\rho = 5$, are illustrated in Fig. 7(a) and (b), respectively.

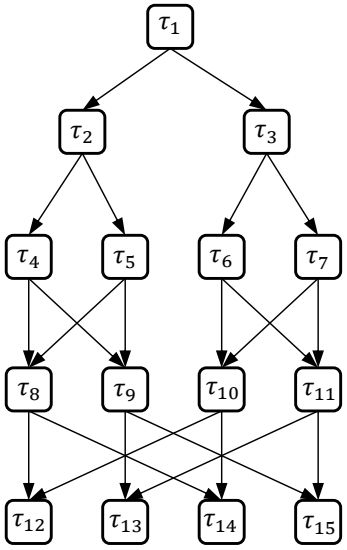
6.3 Comparison Metrics

This section introduces three comparison metrics used in the experiments, as follow:

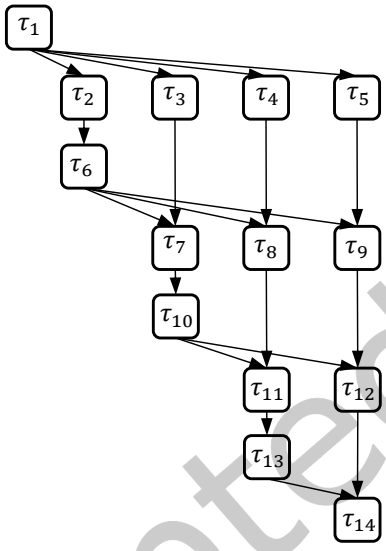
(1) *Average makespan (AM).* The objective of this study is to reduce system energy consumption while meeting job-level deadlines. In fact, there exists a proportional relationship between f and E, as well as between f and $1/makespan$, while ensuring a balance of computing power between processors. This implies that a smaller makespan will have greater opportunities for frequency adjustment, thus offering more room for energy savings. TEBS we propose, along with the comparative IPPTS and DNDS, does not employ DVFS. The algorithms DWDES and DWDS are based on TEBS and DNDS, respectively, and utilize DVFS technology for energy optimization. Therefore, it is necessary to compare the specific makespan and energy consumption of IPPTS, TEBS, and DNDS to highlight the comprehensive advantages of our proposed TEBS in terms of both makespan and energy consumption, i.e., it has both lower energy consumption and makespan. Considering that a large number (N_g) of DAGs with different attributes are used, we calculate the average value of the N_g different $M(F_i)$ as follows.

$$AM = \frac{1}{N_g} \times \sum_{i=1}^{N_g} M(F_i). \tag{34}$$

where F_i represents the i-th DAG to be scheduled. For instance, after 1,000 simulations, 1,000 different makespans can be obtained for each algorithm. Then, we calculate the average value of the 1,000 different makespans.



(a) Example of the fast Fourier transform DAG with $m = 15$.



(b) Example of the Gaussian elimination DAG with $m = 14$.

Fig. 7. Examples include the FFT and GE DAGs, with m-values of 15 and 14, respectively.

(2) *Average energy consumption (AEC).* This metric is the primary research objective of this work. we calculate the average value of the N_g different $E_{total}(F_i)$ as follows,

$$AEC = \frac{1}{N_g} \times \sum_{i=1}^{N_g} E_{total}(F_i). \tag{35}$$

(3) *Average energy-saving rate (AER).* We use the energy consumption of IPPTS, denoted as E_{IPPTS}, as the baseline. The energy-saving rate of the DAG F_i can be calculated as

$$rate(F_i) = \frac{E_{IPPTS} - E_{total}(F_i)}{E_{IPPTS}} \times 100\%. \tag{36}$$

Numerous simulation experiments are conducted to evaluate the effectiveness of the IPPTS, DNDS, TEBS, DWDS, and DWDES algorithms across different number of processors and deadline constraints, including average energy consumption, average makespan, and average energy-saving rate. When evaluating different numbers of processors, θ is set to 1.2. Conversely, when analyzing varying deadline constraints, the number of processors $|U|$ is fixed at 8.

6.4 Performance Results for Random DAGs

6.4.1 The Results Comparison for the AEC. We examine the *AEC* of the IPPTS, DNDS, TEBS, DWDS, and DWDES algorithms under varying deadlines, with the results presented in Fig. 8(a). The findings indicate that DNDS and TEBS consistently consume less energy compared to IPPTS. Additionally, we observe that the impact of deadlines on energy consumption is slight. Although DNDS and TEBS attempt to save energy, they do not utilize DVFS, limiting the extent of energy savings. To further reduce energy consumption, we propose the DWDES algorithm, which builds on TEBS. The existing state-of-the-art algorithm DWDS builds on DNDS. Both DWDES and DWDS

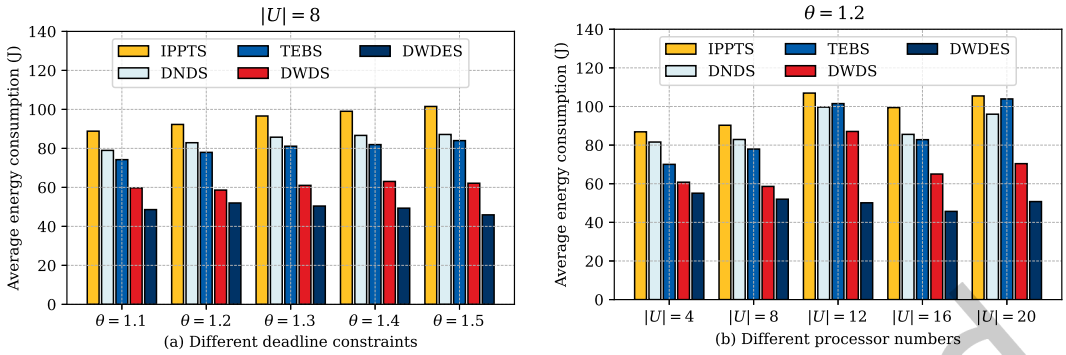


Fig. 8. The *AEC* results under different deadlines and processor numbers for random DAGs.

are DVFS-weakly dependent algorithms, using DVFS only when switching applications. Calculations show that DWDS achieves a 23.62% energy saving over DNDS, while DWDES achieves a 40.24% energy saving over TEBS. Comparing DWDES with DWDS directly, DWDES can save more energy than DWDS. In the experiment shown in Fig. 8(b), we investigate the *AEC* of the algorithms IPPTS, DNDS, TEBS, DWDS, and DWDES across different number of processors. Their *AEC* values are 95.65 J, 84.28 J, 79.84 J, 60.92 J, and 49.25 J, respectively. The energy consumption of DNDS is similar to that of TEBS. DWDES has the lowest energy consumption, reducing energy consumption by 46.40 J, 34.73 J, 15.81 J, and 11.37 J compared to IPPTS, DNDS, TEBS, and DWDS, respectively.

We have compared our algorithms with existing state-of-the-art algorithms under various deadlines and across different processor numbers, respectively. All results demonstrate that DWDES offers greater energy efficiency while meeting the deadlines.

6.4.2 The Results Comparison for the AM. In the experiment shown in Fig. 9, we mainly consider the effectiveness of scheduling algorithms IPPTS, DNDS and TEBS without using DVFS in terms of *AM*. Since DWDS and DWDES minimize energy consumption under deadline constraints using DVFS, their average makespans are close to the *Deadline*. In Fig. 9(a), the *AM*s of IPPTS, TEBS and DNDS are 50.01 s, 55.02 s, and 62.85 s, respectively. We can conclude that TEBS delivers a lower *AM* than DNDS, although it has a slightly higher *AM* compared to IPPTS. As the *Deadline* increases, the advantage of TEBS in terms of *AM* becomes more pronounced when compared to DNDS in Fig. 9(a). This is because DNDS prioritizes minimizing energy consumption within the deadline, which initially increases the makespan while saving energy. In Fig. 9(b), the *AM* of TEBS under different processor

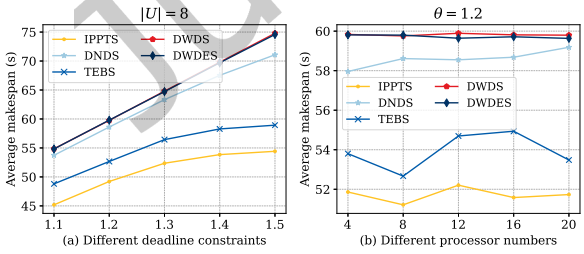


Fig. 9. The *AM* results under different deadlines and processor numbers for random DAGs.

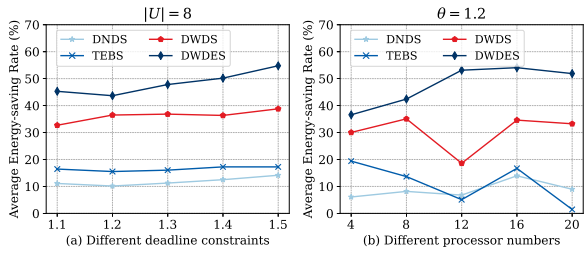


Fig. 10. The *AER* results under different deadlines and processor numbers for random DAGs.

counts is also significantly lower than DNDS, and the *AM*s of IPPTS, TEBS and DNDS are 51.71 s, 53.91 s, and 58.59 s, respectively.

Although IPPTS has the smallest makespan among the five algorithms, its energy consumption is the highest, combining the experiments in Fig. 8. The TEBS's makespan is close to that of IPPTS, but its energy consumption is lower than IPPTS. The TEBS's energy consumption is closer to that of DNDS, but it has a shorter makespan. Therefore, based on this experiment, we conclude that TEBS outperforms IPPTS in terms of *AEC* and surpasses DNDS in *AM*.

6.4.3 The Results Comparison for the AER. In the experiments in Fig. 10, we are using the *AEC* of IPPTS as the baseline to explore the average energy-saving rate of algorithms DNDS, TEBS, DWDS and DWDES under different deadlines and number of processors. In Fig. 10(a), the *AER*s of algorithms DNDS, TEBS, DWDS and DWDES are 11.83%, 16.50%, 36.23% and 48.34% respectively. In summary, when considering the DVFS, the *AER* of DWDES is higher than that of DWDS. Conversely, when the DVFS is not taken into account, TEBS achieves a higher *AER* than DNDS. As the deadline increases, the *AER*s of algorithms DNDS, TEBS, and DWDS remain relatively constant, while DWDES exhibits a slight increase. In Fig. 10(b), the *AER*s of algorithms DNDS, TEBS, DWDS and DWDES are 8.78%, 11.26%, 30.30% and 47.60%, respectively.

Simulation results show that, without considering DVFS, TEBS achieves a higher average energy-saving rate than DNDS. Furthermore, when DVFS is taken into account, DWDES outperforms DWDS in terms of average energy-saving rate.

6.5 Performance Results for FFT DAGs

6.5.1 The Results Comparison for the AEC. In the experiment of Fig. 11(a), the energy consumption performance of different scheduling algorithms for FFT DAGs shows significant differences. Among all the algorithms, IPPTS has the highest *AEC*, reaching 142.17 J, which is considerably higher than the others. The *AEC* of DNDS and TEBS are 121.68 J and 113.49 J, respectively, achieving energy savings of approximately 17.5% and 23.1% compared to IPPTS. On the other hand, DWDS and DWDES have the lowest *AEC* at 92.47 J and 72.95 J, respectively, saving 37.3% and 50.6% energy compared to IPPTS. Particularly, DWDES demonstrates outstanding energy-saving performance. Further observations reveal that IPPTS is highly sensitive to the processor load parameter θ, with energy consumption significantly increasing as θ grows. In contrast, DWDS and DWDES exhibit minimal variation in energy consumption with changes in θ, reflecting strong energy-saving robustness. This indicates

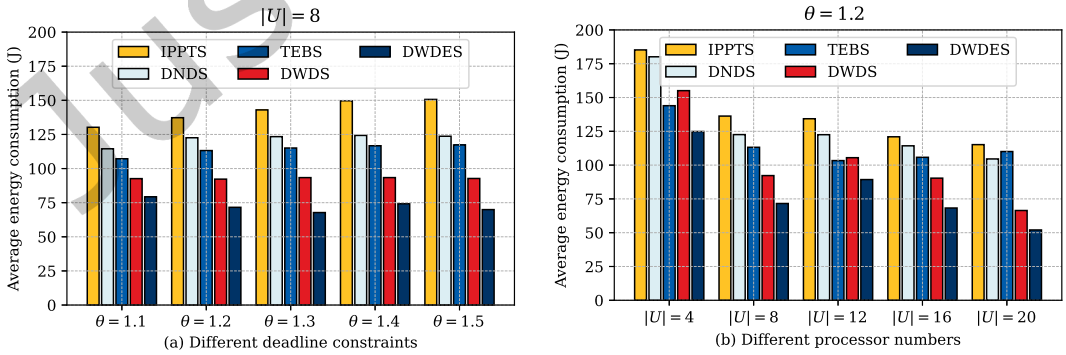


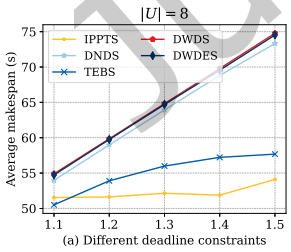
Fig. 11. The *AEC* results under different deadlines and processor numbers for Fast Fourier Transform DAGs.

that in energy-sensitive scenarios, DWDES and DWDS are the preferred scheduling algorithms, with DWDES standing out in terms of energy savings and stability.

The experimental results of Fig. 11(b) demonstrate significant differences in energy consumption performance across scheduling algorithms under varying numbers of processors $|U|$, with distinct trends observed as processor size changes. Specifically, IPPTS exhibits the highest *AEC* at 138.37 J, with a peak of 185.22 J at $|U| = 4$. As the processor size increases, its energy consumption gradually decreases to 115.12 J at $|U| = 20$, yet it consistently remains the highest among all algorithms. In comparison, DNDS and TEBS demonstrate moderate energy-saving capabilities, with *AEC* of 128.81 J and 115.25 J, achieving energy savings of 12.1% and 19.9%, respectively, compared to IPPTS. Notably, TEBS achieves its lowest energy consumption of 103.34 J at $|U| = 12$, though its energy consumption slightly rebounds at larger processor sizes ($|U| = 16, 20$), indicating weaker stability in large-scale processor scenarios. In contrast, DWDS and DWDES demonstrate significantly better and more stable energy-saving performance. DWDS achieves an *AEC* of 101.92 J, decreasing sharply from 155.08 J at $|U| = 4$ to 66.42 J at $|U| = 20$, representing a 30.6% reduction in energy consumption compared to IPPTS. Particularly, DWDES achieves the lowest *AEC* at 81.21 J, consistently maintaining the lowest energy consumption across all processor sizes, dropping from 124.86 J at $|U| = 4$ to 52.03 J at $|U| = 20$. This represents an energy-saving rate of 53.7% compared to IPPTS, showcasing exceptional energy efficiency and robustness to processor size variations.

In terms of trends, the energy consumption of all algorithms decreases with increasing $|U|$, but the rate of decline and final energy consumption differ significantly. IPPTS shows the slowest decline and the weakest sensitivity, indicating limited adaptability to increasing processor sizes. DNDS and TEBS exhibit faster declines but weaker stability for larger processor sizes. DWDS and DWDES, on the other hand, demonstrate strong energy-saving capabilities and robustness, particularly when $|U| \geq 16$, where their energy consumption remains consistently lower than that of other algorithms. Overall, DWDS and DWDES are the most suitable scheduling algorithms for energy-sensitive scenarios, with DWDES standing out for its lowest *AEC*, best energy-saving performance, and adaptability to processor size variations.

6.5.2 The Results Comparison for the AM. In the experiment shown in Fig. 12, we primarily focus on the *AM* of scheduling algorithms IPPTS, DNDS, TEBS, DWDS, and DWDES under different deadline constraints and processor counts. As DWDS and DWDES utilize DVFS to minimize energy consumption while meeting deadline constraints, their *AM*s are closer to the *Deadline*. In Fig. 12(a), the *AM*s of IPPTS, DNDS, TEBS, DWDS, and DWDES are 52.25 s, 63.80 s, 55.05 s, 64.82 s, and 64.65 s, respectively. From this, we observe that TEBS and IPPTS have similar the *AM*, with TEBS slightly outperforming IPPTS, while DNDS has a higher the *AM* due to its focus on energy consumption optimization. As the *Deadline* increases, the *AM* of TEBS improves further relative to DNDS, highlighting the advantage of TEBS in scheduling tasks more efficiently without compromising too much on energy consumption. In Fig. 12(b), we see that as the processor count increases, the *AM* of TEBS remains



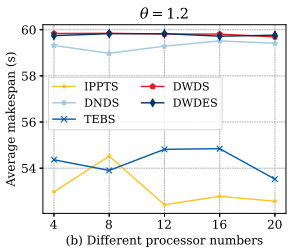
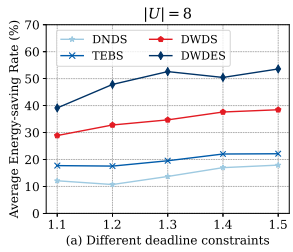


Fig. 12. The *AM* results under different deadlines and processor numbers for FFT DAGs.



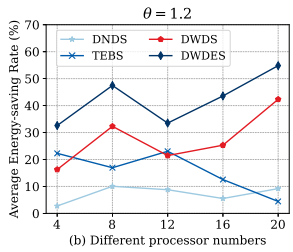


Fig. 13. The *AER* results under different deadlines and processor numbers for FFT DAGs.

significantly lower than that of DNDS, with values of 53.05 s, 54.29 s, and 59.30 s for IPPTS, TEBS, and DNDS, respectively. This reinforces the notion that TEBS perfor better with a higher number of processors, making it a more scalable algorithm compared to DNDS.

By combining the results from both Figs. 12(a) and (b), it is clear that TEBS consistently outperfor DNDS in ter of *AM*, especially under longer deadlines and varying processor counts. Therefore, based on this experiment, we conclude that TEBS delivers a better balance between scheduling efficiency and energy consumption, outperforming both IPPTS and DNDS in terms of *AM* across different scenarios.

6.5.3 The Results Comparison for the AER. In the experiment shown in Fig. 13, we use the *AEC* of IPPTS as the baseline to explore the average energy-saving rate (*AER*) of algorithms DNDS, TEBS, DWDS, and DWDES under different deadlines and processor counts. In Fig. 13(a), the *AER*s of DNDS, TEBS, DWDS, and DWDES are 14.27%, 19.79%, 34.48%, and 48.71%, respectively. From the results, it is evident that when DVFS is considered, DWDES achieves a significantly higher *AER* than DWDS, while TEBS outperforms DNDS when DVFS is not used. As the deadline increases, the *AER*s of DNDS, TEBS, and DWDS remain relatively stable, whereas the *AER* of DWDES shows a slight increase. In Fig. 13(b), the *AER*s of DNDS, TEBS, DWDS, and DWDES are 7.26%, 15.85%, 27.52%, and 42.38%, respectively. These results show that, without considering DVFS, TEBS achieves a higher energy-saving rate than DNDS. However, when DVFS is considered, DWDES outperforms DNDS, TEBS, and DWDS in terms of energy-saving rate.

In summary, when DVFS is considered, DWDES exhibits the best performance in terms of energy-saving rate, significantly outperforming the other algorithms. Conversely, when DVFS is not used, TEBS achieves a higher energy-saving rate than DNDS.

6.6 Performance Results for GE DAGs

6.6.1 The Results Comparison for the AEC. In the experiment, we explored the average energy consumption (*AEC*) of the algorithms under different deadlines and processor counts, with the results presented in Fig. 14(a) and (b). The data shows that as the θ value (*Deadline*) increases, the energy consumption of all algorithms rises. For example, when $\theta = 1.1$, the energy consumption of IPPTS, DNDS, TEBS, DWDS, and DWDES is 75.89 J, 69.57 J, 63.84 J, 50.19 J, and 47.84 J, respectively, whereas when $\theta = 1.5$, the energy consumption increases to 81.37 J, 69.24 J, 67.43 J, 53.48 J, and 35.36 J. This trend indicates that as the deadline lengthens, the algorithms require more computational resources, resulting in higher energy consumption. Notably, DWDES consistently demonstrates the lowest energy consumption across all deadlines, showing the best energy-saving performance.

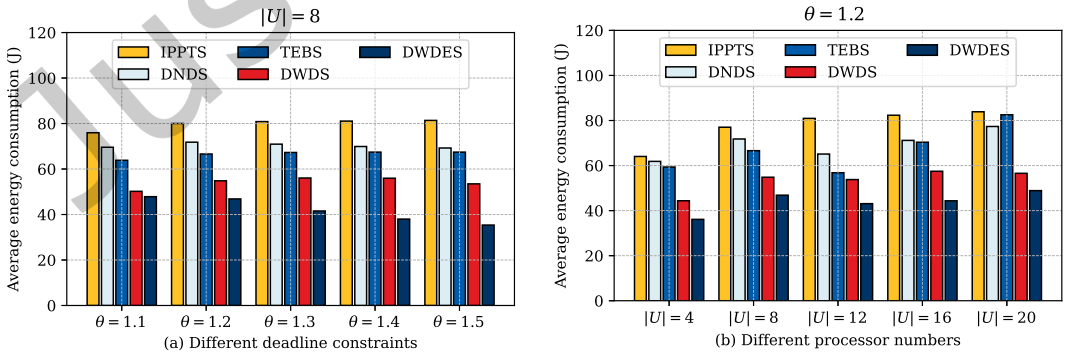
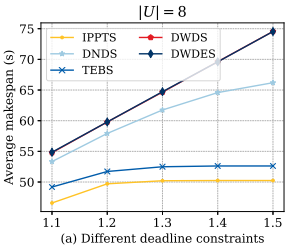


Fig. 14. The *AEC* results under different deadlines and processor numbers for Gaussian Elimination DAGs.



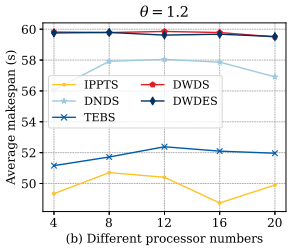
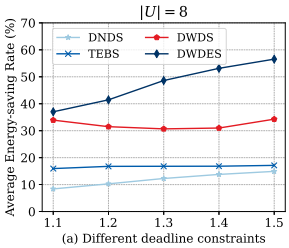


Fig. 15. The *AM* results under different deadlines and processor numbers for EG DAGs.



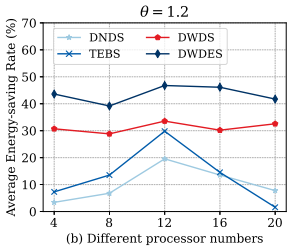


Fig. 16. The *AER* results under different deadlines and processor numbers for EG DAGs.

Further analysis of energy consumption under varying processor counts reveals a decreasing trend in energy consumption as the number of processors increases. From the data in Fig. 14(b), for example, when $|U| = 4$, the energy consumption of IPPTS, DNDS, TEBS, DWDS, and DWDES is 64.94 J, 61.85 J, 59.36 J, 44.38 J, and 36.12 J, respectively, while when $|U| = 20$, the energy consumption decreases to 83.66 J, 77.35 J, 80.53 J, 56.56 J, and 48.87 J. This demonstrates that as the number of processors increases, the computational capacity of the algorithms is enhanced, which effectively reduces energy consumption. DWDES consistently exhibits the lowest energy consumption across different processor counts, further validating its advantage in energy efficiency.

6.6.2 The Results Comparison for the AM. In this experiment, we analyze the average makespan (*AM*) of various scheduling algorithms under different deadline constraints and processor counts. First, we examine the results with varying deadlines. As the deadline increases, the *AM* of all algorithms shows distinct trends. In Fig. 15(a), when $\theta = 1.1$ ($Deadline = 55$ s), the *AM* values are as follows: IPPTS (46.57 s), DNDS (53.33 s), TEBS (49.18 s), DWDS (54.79 s), and DWDES (54.86 s). It is observed that TEBS and IPPTS have similar *AM* values, with TEBS slightly outperforming IPPTS, while DNDS exhibits a higher *AM* due to its focus on energy optimization. As the θ value increases to 1.2, 1.3, 1.4, and 1.5, the *AM* of all algorithms increases, with TEBS and IPPTS experiencing smaller increases, while DNDS shows a more significant rise. For instance, at $\theta = 1.5$ ($Deadline = 75$ s), the *AM* values are: TEBS (52.61 s), IPPTS (50.24 s), and DNDS (66.19 s), highlighting TEBS's more efficient scheduling without sacrificing too much on energy savings. Next, we analyze the *AM* with different processor counts in Fig. 15(b). At processor $|U| = 4$, the *AM* values are: IPPTS (49.14 s), DNDS (56.07 s), TEBS (51.16 s), DWDS (59.82 s), and DWDES (59.76 s). As the number of processors increases, the *AM* values of all algorithms stabilize. For example, at processor $|U| = 20$, the values are: IPPTS (49.93 s), DNDS (56.91 s), TEBS (51.96 s), DWDS (59.50 s), and DWDES (59.52 s). This trend indicates that with more processors, all algorithms reach a steady state, particularly TEBS and IPPTS, which maintain high scheduling efficiency even in larger processor environments.

In summary, TEBS consistently outperforms other algorithms in terms of *AM*, particularly under longer deadlines and with more processors. While DNDS shows some advantages in energy efficiency, its relatively higher makes it less favorable compared to TEBS. Therefore, TEBS demonstrates a better balance between scheduling efficiency and energy consumption, offering superior overall performance in the experiments.

6.6.3 The Results Comparison for the AER. In the experiment shown in Fig. 16, we use the *AEC* of IPPTS as the baseline to examine the average energy-saving rate (*AER*) of the DNDS, TEBS, DWDS, and DWDES algorithms under varying deadlines and processor counts. As shown in Fig. 16(a), the *AER* values for DNDS, TEBS, DWDS, and DWDES are 11.92%, 16.69%, 32.26%, and 47.35%, respectively. These results indicate that when DVFS is considered, DWDES achieves a significantly higher *AER* compared to DWDS, while TEBS outperforms DNDS in the absence of DVFS. As the deadline increases, the *AER* values for DNDS, TEBS, and DWDS remain relatively stable, while

the *AER* of DWDES shows a slight increase. In Fig. 16(b), the *AER* values for DNDS, TEBS, DWDS, and DWDES are 10.21%, 13.35%, 31.16%, and 43.47%, respectively. These results demonstrate that, without considering DVFS, TEBS achieves a higher energy-saving rate than DNDS. However, when DVFS is taken into account, DWDES outperforms DNDS, TEBS, and DWDS in terms of energy-saving rate.

Through the Gaussian Elimination DAG experiments, we conclude that our proposed TEBS algorithm can simultaneously minimize energy consumption and makespan, providing DWDES with a larger energy-saving optimization space. Finally, DWDES achieves maximum system energy consumption reduction within the deadline by accurately adjusting the processor frequency.

7 Conclusion

In this paper, we have analyzed the precedence constraints in the Job-DAG $F = (\boldsymbol{J}, E_{job}, D)$ and simplified the structure by removing unnecessary dependencies, improving the efficiency of the Job-DAG. In addition, we have reported a time and energy balance scheduling algorithm (TEBS) designed to achieve DVFS-independent energy-efficient scheduling. Building on TEBS, we have introduced a DVFS-weakly dependent energy-saving algorithm (DWDES) aimed at further reducing energy consumption while adhering to deadline constraints. This type of DVFS-weak dependency is more aligned with real-world applications. DWDES has achieved significant energy savings compared to existing algorithms while successfully meeting deadline constraints. In future research, we plan to address the issues of multi-task modeling and scheduling, exploring new methods for constructing highly efficient task models and scheduling strategies to tackle current challenges.

Acknowledgments

We are grateful to the anonymous reviewers for their insightful comments and suggestions, which improved the overall quality of this work. The work reported in this paper was supported by the Hunan Provincial Natural Science Foundation of China (Grant No. 2023JJ30263) and Scientific Research Fund of Hunan Provincial Education Department (Grant No. 22B0510). The corresponding authors are Kuan Jiang, Lijun Xiao and Haibo Zeng.

References

[1] Alessandra Melani, Marko Bertogna, Vincenzo Bonifaci, Alberto Marchetti-Spaccamela, and Giorgio Buttazzo. 2016. Schedulability analysis of conditional parallel task graphs in multicore systems. *IEEE Trans. Comput.* 66, 2 (2016), 339–353.

[2] Julien Forget, Frédéric Boniol, Emmanuel Grolleau, David Lesens, and Claire Pagetti. 2010. Scheduling dependent periodic tasks without synchronization mechanisms. In *2010 16th IEEE Real-Time and Embedded Technology and Applications Symposium*. IEEE, 301–310.

[3] Wanli Chang, Shuai Zhao, Simon Burton, Haitong Wang, Ting Chen, Nan Chen, and Neil Audsley. 2021. Hardware/software co-synthesis and co-optimization for autonomous systems. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1319–1322.

[4] Shuiguang Deng, Hailiang Zhao, Zhengzhe Xiang, Cheng Zhang, Rong Jiang, Ying Li, Jianwei Yin, Schahram Dustdar, and Albert Y Zomaya. 2021. Dependent function embedding for distributed serverless edge computing. *IEEE Transactions on Parallel and Distributed Systems* 33, 10 (2021), 2346–2357.

[5] Shuai Zhao, Xiaotian Dai, and Iain Bate. 2022. DAG scheduling and analysis on multi-core systems by modelling parallelism and dependency. *IEEE Transactions on Parallel and Distributed Systems* 33, 12 (2022), 4019–4038.

[6] Qingqiang He, Nan Guan, Zhishan Guo, et al. 2019. Intra-task priority assignment in real-time scheduling of DAG tasks on multi-cores. *IEEE Transactions on Parallel and Distributed Systems* 30, 10 (2019), 2283–2295.

[7] Jing Huang, Kuan Jiang, Weijie Wang, Wei Liang, and Wanli Chang. 2025. Construction of DAG Models for Autonomous Systems. In *2025 62nd ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[8] Debabrata Senapati, Arnab Sarkar, and Chandan Karfa. 2021. HMDS: A makespan minimizing DAG scheduler for heterogeneous distributed systems. *ACM Transactions on Embedded Computing Systems (TECS)* 20, 5s (2021), 1–26.

[9] Micaela Verucchi, Mirco Theile, Marco Caccamo, and Marko Bertogna. 2020. Latency-aware generation of single-rate DAGs from multi-rate task sets. In *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 226–238.

[10] Zahaf Houssam-Eddine, Nicola Capodieci, Roberto Cavicchioli, Giuseppe Lipari, and Marko Bertogna. 2020. The hpc-dag task model for heterogeneous real-time systems. *IEEE Trans. Comput.* 70, 10 (2020), 1747–1761.

[11] S Liu, B Yu, N Guan, Z Dong, and B Akesson. 2021. Real-time scheduling and analysis of an autonomous driving system. *Proc. RTSS Ind. Challenge Problem* (2021), 1–47.
[12] Matthias Becker, Dakshina Dasari, Saad Mubeen, Moris Behnam, and Thomas Nolte. 2016. Synthesizing job-level dependencies for automotive multi-rate effect chains. In *2016 IEEE 22nd International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. IEEE, 159–169.
[13] Hyunjong Choi, Mohsen Karimi, and Hyoseung Kim. 2020. Chain-based fixed-priority scheduling of loosely-dependent tasks. In *2020 IEEE 38th International Conference on Computer Design (ICCD)*. IEEE, 631–639.
[14] Salah Eddine Saidi, Nicolas Pernet, and Yves Sorel. 2017. Automatic parallelization of multi-rate fmi-based co-simulation on multi-core. In *TMS/DEVS 2017-Symposium on Theory of Modeling and Simulation*. ACM, Article–No.
[15] Jinghao Sun, Kailu Duan, Xisheng Li, Nan Guan, Zhishan Guo, Qingxu Deng, and Guozhen Tan. 2023. Real-Time Scheduling of Autonomous Driving System with Guaranteed Timing Correctness. In *2023 IEEE 29th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 185–197.
[16] Antonio E Mirino et al. 2017. Best routes selection using Dijkstra and Floyd-Warshall algorithm. In *2017 11th International Conference on Information & Communication Technology and System (ICTS)*. IEEE, 155–158.
[17] Haluk Topcuoglu, Salim Hariri, and Min-You Wu. 2002. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems* 13, 3 (2002), 260–274.
[18] Hamza Djigal, Jun Feng, Jiamin Lu, and Jidong Ge. 2020. IPPTS: an efficient algorithm for scientific workflow scheduling in heterogeneous computing systems. *IEEE Transactions on Parallel and Distributed Systems* 32, 5 (2020), 1057–1071.
[19] Binqi Sun, Mirco Theile, Ziyuan Qin, Daniele Bernardini, Debayan Roy, Andrea Bastoni, and Marco Caccamo. 2024. Edge generation scheduling for DAG tasks using deep reinforcement learning. *IEEE Trans. Comput.* 73, 4 (2024), 1034–1047.
[20] Jinchao Chen, Pengcheng Han, Ying Zhang, Tao You, and Pengyi Zheng. 2023. Scheduling energy consumption-constrained workflows in heterogeneous multi-processor embedded systems. *Journal of Systems Architecture* 142 (2023), 102938.
[21] Jinchao Chen, Yang Wang, Ying Zhang, Yantao Lu, Qing Li, and Qiuhao Shu. 2025. Non-Preemptive Scheduling of Periodic Tasks with Data Dependencies in Heterogeneous Multiprocessor Embedded Systems. *ACM Transactions on Design Automation of Electronic Systems* 30, 2 (2025), 1–25.
[22] Wenbing Yang, Mingqiang Zhao, Jingbo Li, and Xingjun Zhang. 2024. Energy-efficient DAG scheduling with DVFS for cloud data centers. *The Journal of Supercomputing* 80, 10 (2024), 14799–14823.
[23] Yi-Wen Zhang. 2021. Energy-aware fixed priority scheduling with shared resources in standby-sparing systems. *Microprocessors and Microsystems* 87 (2021), 104362.
[24] Tarek Hagras and Gamal A El-Sayed. 2024. Maintaining the completion-time mechanism for Greening tasks scheduling on DVFS-enabled computing platforms. *Cluster Computing* 27, 6 (2024), 7373–7388.
[25] Jing Huang, Hao Sun, Fan Yang, Shouping Gao, and Renfa Li. 2022. Energy optimization for deadline-constrained parallel applications on multi-ECU embedded systems. *Journal of Systems Architecture* 132 (2022), 102739.
[26] Debabrata Senapati, Arnab Sarkar, and Chandan Karfa. 2022. Energy-aware real-time scheduling of multiple periodic DAGs on heterogeneous systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2022).
[27] Jing Huang, Renfa Li, Jiyao An, Haibo Zeng, and Wanli Chang. 2021. A DVFS-weakly dependent energy-efficient scheduling approach for deadline-constrained parallel applications on heterogeneous systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 40, 12 (2021), 2481–2494.
[28] Guoqi Xie, Gang Zeng, Xiongren Xiao, Renfa Li, and Keqin Li. 2017. Energy-efficient scheduling algorithms for real-time parallel applications on heterogeneous distributed embedded systems. *IEEE Transactions on Parallel and Distributed Systems* 28, 12 (2017), 3426–3442.
[29] Guoqi Xie, Gang Zeng, Yuekun Chen, Yang Bai, Zhili Zhou, Renfa Li, and Keqin Li. 2017. Minimizing redundancy to satisfy reliability requirement for a parallel application on heterogeneous service-oriented systems. *IEEE Transactions on Services Computing* 13, 5 (2017), 871–886.
[30] Jinchao Chen, Chenglie Du, Pengcheng Han, and Xiaoyan Du. 2019. Work-in-Progress: Non-preemptive Scheduling of Periodic Tasks with Data Dependency Upon Heterogeneous Multiprocessor Platforms. In *2019 IEEE Real-Time Systems Symposium (RTSS)*. 540–543. DOI: http://dx.doi.org/10.1109/RTSS46320.2019.00059
[31] Peng Chen, Hui Chen, Weichen Liu, Linbo Long, Wanli Chang, and Nan Guan. 2023. DAG-Order: An Order-Based Dynamic DAG Scheduling for Real-Time Networks-on-Chip. *ACM Transactions on Architecture and Code Optimization* 21, 1 (2023), 1–24.
[32] A. K. Amoura, E. Bampis, and Jean Claude Knig. 1998. Scheduling Algorithms for Parallel Gaussian Elimination With Communication Costs. *IEEE Transactions on Parallel and Distributed Systems* (1998).
[33] Y. Chung and S. Ranka. 1992. Applications and performance analysis of a compile-time optimization approach for list scheduling algorithms on distributed memory multiprocessors. (1992).
[34] Wzwtime. 2017. RandomGraphGenerator. [Online]. Available: https://github.com/laser/random-dag-generator-go. (2017).
[35] Wikipedia. 2019. Gaussian elimination. [Online]. Available: https://en.wikipedia.org/wiki/Gaussian/elimination. (2019).

Received 4 January 2025; revised 16 July 2025; accepted 7 October 2025