Journal of Systems Architecture 178 (2026) 103906

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

CFSECC: Cyclic feedback scheduling for energy-constrained real-time applications in heterogeneous distributed systems

Yuxi Chen [a], Wufei Wu [a,*], Wei Li [a], Jiaxin Zeng [a], Keqin Li [b]

[a] School of Information Engineering, Nanchang University, China
[b] Department of Computer Science, State University of New York, New Paltz, NY 12561, USA

ARTICLE INFO

Keywords:
Cyclic feedback
Energy consumption limitation
Heterogeneous distributed embedded systems
Parallel computing
Task scheduling

ABSTRACT

Heterogeneous distributed systems can significantly improve the computational efficiency of large-scale tasks; however, task scheduling optimization faces the dual challenges of energy consumption and scheduling time constraints. Current state-of-the-art scheduling algorithms often fail to fully utilize the multi-round scheduling potential of the system, making it difficult to simultaneously optimize completion time and energy consumption under different energy allocation schemes. To address this issue, this paper proposes a cyclic feedback scheduling optimization strategy and designs a reasonable termination condition for the cyclic strategy based on theoretical derivation. Furthermore, combining these two iterative strategies and three different energy allocation schemes, we design three efficient and practical cyclic feedback algorithms: the α-Cyclic Iterative Algorithm, the β-Reverse Cyclic Algorithm, and the γ-Reverse Cyclic Algorithm. Experimental results show that the proposed method can effectively shorten task scheduling time. For example, in a real-world industrial DAG, when the energy availability rate is 0.9, compared with the ESECC, ISAECC, EECC, TSSA, and SEDC algorithms, the scheduling length is reduced by approximately 12.13%, 18.00%, 12.01%, 6.80%, and 19.31%, respectively.

1. Introduction

With the rapid development of cloud computing, artificial intelligence, and embedded systems, both data volume and computational demand have grown dramatically. Heterogeneous distributed multi-core systems are gradually replacing traditional single-core platforms to achieve higher operational efficiency. Typical heterogeneous multi-core systems in the market include ARM's big.LITTLE architecture [1,2], Apple Silicon (M1, M2 series) [3], NVIDIA SoCs (Tegra series) [4], and Qualcomm's Snapdragon series [5].

For systems with periodic and highly repetitive workloads, such as many embedded controllers and batch-processing applications, the execution order of tasks and the energy allocation for each task are often planned before program execution. This scheduling paradigm is known as static scheduling [6,7]. Static schedules are generated at compile-time or load-time, and once execution begins, the task ordering and resource allocation remain essentially fixed.

The key performance indicators for task scheduling are energy consumption and scheduling length (makespan). For compute-intensive applications under static scheduling, pre-computing an optimized schedule before runtime enables us to spend a relatively small amount of simulation effort to save a large amount of computational and energy resources during actual execution. In this work, we aim to fully exploit the multi-round pre-computation capability of static scheduling and introduce a cyclic feedback mechanism [8] to allocate a given energy budget to processors as efficiently as possible, thereby minimizing the makespan subject to energy constraints. To this end, we propose a cyclic feedback scheduling framework, termed Cyclic Feedback Scheduling for Energy-Constrained Real-Time Applications in Heterogeneous Distributed Systems (CFSECC), which iteratively refines energy allocation through static DAG scheduling feedback-based simulation.

1.1. Motivation

With the widespread adoption of heterogeneous distributed computing systems [9–11], the complexity of modern workloads continues to increase, and minimizing the makespan under a given energy constraint has become a crucial yet challenging problem. In large-scale heterogeneous platforms, processors differ significantly in performance and energy characteristics, making it difficult for conventional static scheduling strategies to simultaneously satisfy energy constraints and

* Corresponding author.
E-mail address: wuwufei@ncu.edu.cn (W. Wu).

https://doi.org/10.1016/j.sysarc.2026.103906
Received 8 December 2025; Received in revised form 20 June 2026; Accepted 21 June 2026
Available online 27 June 2026

optimize the execution schedule. Existing energy-constrained scheduling algorithms attempt to minimize makespan through various optimization heuristics; however, they often suffer from imbalanced energy allocation and insufficient iterative refinement of schedules. As a result, they fail to fully exploit the potential of multi-round precomputation in static scheduling, leading to unnecessary performance and energy-efficiency losses.

To address these issues, this paper proposes a family of scheduling optimization algorithms based on cyclic feedback, which fully exploit the surplus energy remaining after each allocation round and redistribute it in a refined manner. Through the cyclic feedback mechanism, our algorithms dynamically adjust the energy assigned to each task over multiple simulation rounds and balance the workload across heterogeneous processors. In certain scenarios, this approach enables joint optimization — simultaneously shortening the makespan and reducing energy consumption — showing strong potential for large-scale static task scheduling in energy-constrained heterogeneous systems.

1.2. Contributions

The objective of this paper is to minimize the makespan in heterogeneous distributed systems under a given energy constraint. The main contributions of this work are summarized as follows.

(1) We propose a cyclic feedback strategy that refines each round of simulation-based scheduling decisions using the results of the previous round. Building on this idea, we introduce the α-loop iteration strategy, which prioritizes allocating surplus energy to processors that have not yet reached their maximum operating frequency, and the β-loop iteration strategy, which prioritizes reducing the surplus energy assigned to fully loaded processors. Through such iterative refinement, the energy distribution progressively approaches an energy-efficient configuration.

(2) We apply the above two cyclic iteration strategies to three distinct initial surplus-energy allocation schemes and, through analysis, determine the most suitable iteration strategy for each case. Based on these combinations, we further develop three concrete algorithms: the α-Cyclic Iterative Algorithm, the β-Reverse Cyclic Algorithm, and the γ-Reverse Cyclic Algorithm, ensuring that all three types of initial energy distributions can be effectively optimized via cyclic feedback. In addition, we design principled termination conditions for the proposed algorithms and provide the corresponding correctness proofs.

(3) We conduct extensive experiments on multiple application scenarios, including Fast Fourier Transform (FFT) and Gaussian Elimination (GE) benchmarks, randomly generated DAGs, and a real-world industrial task graph. The results show that, in most cases, the proposed algorithms consistently achieve shorter makespans than state-of-the-art energy-constrained scheduling methods under the same energy budget.

The remainder of this paper is organized as follows. Section 2 reviews related work. Section 3 describes the system model. Section 4 formulates and analyzes the scheduling problem. Section 5 presents the proposed algorithms in detail. Section 6 reports evaluation results on FFT and GE applications, randomly generated DAGs, and a real-world industrial case study. Section 7 concludes the paper.

2. Related work

Early studies on task scheduling focused on single-core architectures, but the emergence of heterogeneous multi-core processors has made the problem substantially more complex. In such environments, task scheduling is NP-complete, and static scheduling must balance two key objectives: minimizing energy consumption and reducing scheduling length (makespan).

Dynamic Voltage and Frequency Scaling (DVFS) [12–14] has been widely adopted to manage energy consumption, enabling processors to lower their power usage during periods of reduced computational demand. Although DVFS is effective in power-sensitive systems, its integration into static scheduling requires a coordinated energy-allocation strategy.

The idea of iterative refinement used in this work is inspired by the general notion of feedback, a principle widely applied in engineering systems to progressively correct deviations and improve system behavior. Although originating from control-related applications such as vehicle suspension and aeroengine distributed control [8,15], its core concept — using execution results to refine subsequent decisions — naturally motivates the cyclic feedback mechanism introduced in our scheduling framework.

For optimizing makespan under a given energy budget, a number of representative algorithms have been proposed. Topcuoglu and Hariri introduced the Heterogeneous Earliest Finish Time (HEFT) algorithm [16], which determines task priorities by computing each task's upward rank and assigning tasks according to their Earliest Finish Time (EFT). Guoqi Xie et al. proposed the MSLECC algorithm [17], which selects processor–frequency combinations that satisfy energy constraints while minimizing the makespan. The ESECC algorithm [18] improves energy utilization by uniformly distributing surplus energy among tasks, whereas the EECC algorithm [19] enhances the energy pre-allocation process through refined modeling. Ye et al. further developed ISAECC [20], adopting a linear-weighted surplus-energy allocation scheme to better reflect task-level energy heterogeneity. More recently, Zhu et al. proposed TSSA [21], which reverses the task graph and distributes surplus energy based on the maximum-frequency execution time of each task. In parallel, SEDC [22] introduces a time-difference coefficient to produce a more fine-grained task priority order and performs an additional redistribution of residual energy.

A summarized comparison of these representative energy-constrained scheduling algorithms — including their advantages and limitations — is provided in Table 1.

In addition to these classical algorithms, several recent studies have further explored energy–time optimization in heterogeneous distributed and fog–cloud environments. Khaledian et al. developed a deadline-aware, energy-efficient workflow scheduling approach for fog–cloud systems, highlighting the importance of cross-layer coordination [23]. Ijaz et al. examined joint optimization of makespan and energy in hierarchical fog–cloud environments, showing the benefit of balancing resource heterogeneity and energy constraints [24]. Li et al. proposed an energy-aware hybrid-cloud scheduler that adapts resource allocation to multi-workflow characteristics [25]. Furthermore, reinforcement learning has been used for precedence-constrained tasks, as Jayanetti et al. demonstrated improved energy–time efficiency using DRL-based schedulers in edge–cloud systems [26]. Although these studies provide valuable perspectives, most rely on meta-heuristics or learning-based strategies rather than deterministic static scheduling.

Despite these advances, existing approaches still exhibit several inherent limitations. Most traditional algorithms perform only a single round of surplus-energy assignment and therefore cannot exploit the multi-round pre-computation capability inherent to static scheduling. As a result, they are unable to correct imbalanced energy allocation, mitigate energy waste caused by full-frequency processors, or redirect surplus energy to locations where it yields the greatest reduction in makespan. This lack of iterative coordination significantly limits their effectiveness under diverse DAG structures, varying processor scalability, and different levels of energy sufficiency.

These limitations highlight the need for a scheduling strategy capable of multi-round refinement and adaptive energy redistribution. Motivated by this insight, we introduce a cyclic feedback scheduling framework that iteratively corrects allocation imbalance and enhances overall energy utilization through coordinated task–energy adjustments.

Recently, intelligent optimization techniques such as reinforcement learning and joint resource configuration have also been investigated for energy-aware scheduling and resource management in distributed

Table 1
Comparison of representative energy-constrained scheduling algorithms.

Year	Algorithm	Advantages	Limitations
2016	**MSLECC** [17] (Minimizing Schedule Length under Energy Consumption Constraint)	Consistently satisfies global energy-consumption constraints, with correctness supported by theoretical analysis and experimental validation.	Suffers from uneven energy allocation, which leads to large differences in task-level energy usage and reduced overall scheduling efficiency.
2017	**ESECC** [18] (Efficient Scheduling under Energy Consumption Constraint)	Introduces an average surplus-energy allocation strategy that effectively enforces the global energy constraint.	Does not adequately account for task heterogeneity, and its energy-allocation strategy remains relatively coarse-grained.
2019	**EECC** [19] (Enhanced Energy-Constrained Scheduling)	Proposes an improved energy pre-allocation method and provides theoretical proof of correctness.	The pre-allocation process does not explicitly consider detailed characteristics of individual tasks.
2020	**ISAECC** [20] (Improved Scheduling Approach for Energy-Constrained Computing)	Employs a weighted energy-allocation scheme based on task energy-demand ranges, improving allocation precision.	Does not fully integrate the structure of the task graph, and its overall scheduling flexibility and adaptability remain limited.
2022	**TSSA** [21] (Task Structure-Aware Scheduling Algorithm)	Takes both the task-graph structure and the average execution time across processors into account, enhancing structural awareness.	Requires both forward and reverse traversals of the task graph, which increases scheduling complexity and overhead.
2023	**SEDC** [22] (Scheduling based on Energy Difference Coefficient)	Uses an energy-difference coefficient to prioritize tasks and allocate energy, leading to improved energy utilization in many cases.	Under high energy budgets, its static allocation scheme may underutilize available resources and limit further makespan reduction.

Table 2
Key notations and descriptions.

Notation	Description
$rank_u(n_i)$	Upward rank of task n_i
$EST(n_i, u_k, f_{k,h})$	Earliest start time of n_i on u_k at frequency $f_{k,h}$
$EFT(n_i, u_k, f_{k,h})$	Earliest finish time of n_i on u_k at frequency $f_{k,h}$
$w_{i,k}$	WCET of n_i on u_k at maximum frequency
$c_{i,j}$	Worst-case communication delay between n_i and n_j
G	Application DAG
P_{ind}	Frequency-independent dynamic power term
P_d	Frequency-dependent dynamic power term
C_{ef}	Effective switching capacitance
m	Dynamic power exponent
f_{ee}	Minimum-energy (energy-efficient) operating frequency
$SL(G)$	Schedule length (makespan) of G
$E_{\text{given}}(G)$	Total allowable energy for G
$E^{(n)}_{\text{expected}}(n_i)$	Energy assigned to n_i in the nth simulation round

and edge environments [27–29]. For example, latency- and reliability-aware task offloading under energy harvesting [27], joint client selection and resource configuration for energy-efficient federated learning [28], and DRL-based request dispatching in serverless edge systems [29] have demonstrated the potential of adaptive learning-based strategies.

However, the problem addressed in this work focuses on static DAG scheduling under strict energy constraints, where deterministic feasibility and structural predictability must be guaranteed. In contrast to these dynamic or learning-based approaches, the proposed CFSECC framework emphasizes deterministic cyclic optimization within a fixed energy budget.

3. System model

In this section, we introduce the task structure, the energy-consumption model, and the energy-allocation methods. Table 2 summarizes the main notations used throughout this paper and their definitions.

3.1. Task structure

To better constrain the execution order of tasks in scheduling, we model each parallel application as a Directed Acyclic Graph (DAG)

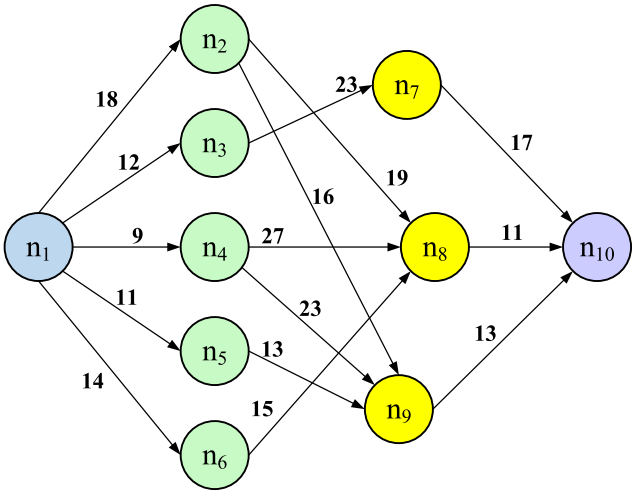


Fig. 1. An example DAG with 10 tasks.

Table 3
Execution times of tasks n_1 to n_{10} at maximum frequency.

Task	n_1	n_2	n_3	n_4	n_5	n_6	n_7	n_8	n_9	n_{10}
u_1	14	13	11	13	12	13	7	5	18	21
u_2	16	19	13	8	13	16	15	11	12	7
u_3	9	18	19	17	10	9	11	14	20	16

[30–32]. The processor set is denoted by $U = \{u_i\}$, where $1 \le i \le |U|$ and $|U|$ is the number of processors.

We define $G = \{N, W, M, C\}$, where N is the set of nodes in G, and each node corresponds to a task whose execution time may differ across processors. W is an $|N| \times |U|$ matrix and records the execution time of each task when processors run at their maximum frequencies. M is the set of communication edges between tasks. $\text{pred}(n_i)$ and $\text{succ}(n_i)$ denote the sets of predecessor and successor tasks of n_i, respectively. An illustrative 10-task DAG is shown in Fig. 1, and the corresponding execution-time matrix W is given in Table 3.

To further evaluate the scheduling algorithms, we also employ Fast Fourier Transform (FFT) and Gaussian Elimination (GE) applications

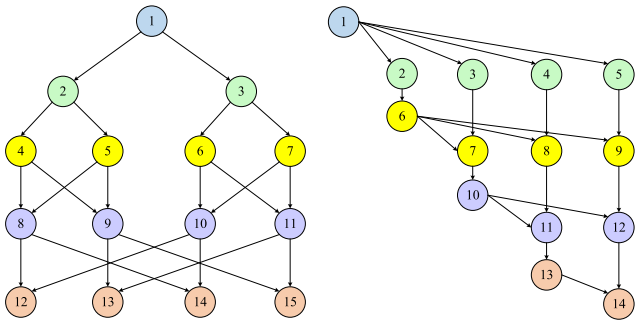


Fig. 2. Task-graph structures of FFT and GE benchmarks.

as benchmark DAGs [33–35]. Their classical task-graph structures are illustrated in Fig. 2. The parameter ρ denotes the problem size (e.g., matrix dimension). For FFT and GE, the total number of task nodes can be computed as

$$|N| = 2\rho - 1 + \rho \log_2 \rho \quad \text{(FFT)}, \qquad |N| = \frac{\rho^2 + \rho - 2}{2} \quad \text{(GE)}.$$

In addition, the parameter ρ is required to satisfy $\rho = 2^t$, where t is an integer.

Next, we recall several important concepts used in list-based task scheduling.

Upward rank value: In task scheduling, the upward rank is used to determine task priorities. Tasks with larger upward ranks are regarded as more critical and are therefore scheduled earlier. The upward rank of task n_i is defined as

$$\text{rank}_u(n_i) = \overline{w_i} + \max_{n_j \in \text{succ}(n_i)} \left\{ c_{i,j} + \text{rank}_u(n_j) \right\}, \tag{1}$$

where $\overline{w_i}$ denotes the average execution time of n_i over all processors:

$$\overline{w_i} = \frac{\sum_{k=1}^{|U|} w_{i,k}}{|U|}. \tag{2}$$

EST (Earliest Start Time): The value $\text{EST}(n_i, u_k, f_{k,h})$ is used as a criterion for task assignment and represents the earliest time when n_i can start on processor u_k at frequency $f_{k,h}$:

$$\text{EST}(n_i, u_k, f_{k,h}) = \max \left\{ \text{avail}[k], \max_{n_x \in \text{pred}(n_i)} \left\{ \text{AFT}(n_x) + c'_{x,i} \right\} \right\}, \tag{3}$$

where $\text{AFT}(n_x)$ is the actual finish time of task n_x and $c'_{x,i}$ is the communication delay between n_x and n_i. When n_x and n_i are assigned to the same processor, $c'_{x,i} = 0$.

EFT (Earliest Finish Time): The value $\text{EFT}(n_i, u_k, f_{k,h})$ is another standard used for task assignment and represents the earliest completion time of n_i on u_k at frequency $f_{k,h}$:

$$\text{EFT}(n_i, u_k, f_{k,h}) = \text{EST}(n_i, u_k, f_{k,h}) + w_{i,k} \times \frac{f_{k,\max}}{f_{k,h}}. \tag{4}$$

Scheduling length (SL): The scheduling length is an important metric to evaluate a scheduling scheme, representing the total time required to complete all tasks in G:

$$SL(G) = \min_{u_k \in U} \left\{ \min_{f_{k,h} \in [f_{k,\text{low}}, f_{k,\max}]} \left\{ \text{EFT}(n_{\text{exit}}, u_k, f_{k,h}) \right\} \right\}. \tag{5}$$

Equivalently, the scheduling length can be regarded as the earliest completion time of the exit node n_{exit} after all its predecessors are finished.

3.2. Energy consumption model

The supply voltage and operating frequency of CMOS circuits are approximately linearly related, which allows adjusting the frequency to optimize energy usage. In this work, we focus on dynamic power, which is the dominant contributor to energy consumption. For a DVFS-enabled processor operating at frequency f, the power consumption is modeled as

$$P(f) = P_s + h(P_{ind} + P_d) = P_s + h\left(P_{ind} + C_{ef} f^m\right), \tag{6}$$

where P_s denotes the static power, and dynamic power consists of two components, P_{ind} and P_d. Here, P_{ind} is the frequency-independent part (a constant), whereas P_d is the frequency-dependent part. The variable h indicates the processor state: $h = 1$ when the processor is active and $h = 0$ when it is powered down. C_{ef} is the effective switching capacitance and m is the dynamic power exponent.

Because static power is typically uncontrollable and relatively small in magnitude, we ignore it in the following and focus only on dynamic power.

When the operating frequency is reduced, the execution time of a task increases; hence, a lower frequency does not necessarily imply lower energy consumption. The energy-efficient frequency f_{ee} that minimizes dynamic energy can be derived as

$$f_{ee} = \sqrt[m]{\frac{P_{ind}}{(m-1)C_{ef}}}. \tag{7}$$

For a processor whose supported frequency range spans from f_{min} to f_{max}, the feasible operating frequency interval is restricted to $[f_{low}, f_{max}]$, where f_{low} is given by

$$f_{low} = \max\{f_{min}, f_{ee}\}. \tag{8}$$

When the processor is active, its operating frequency f must lie within this interval. The energy consumption of task n_i executed on processor u_k at frequency $f_{k,h}$ is then

$$E(n_i, u_k, f_{k,h}) = \left(P_{k,ind} + C_{k,ef} f_{k,h}^{m_k}\right) \times \frac{w_{i,k} f_{k,\max}}{f_{k,h}}, \tag{9}$$

where $w_{i,k}$ is the execution time of n_i on u_k at maximum frequency $f_{k,\max}$.

In this paper, we adopt a simplified model and ignore energy consumption introduced by inter-task communication. Under this assumption, the energy constraint for an application G is expressed as

$$E(G) = \sum_{i=1}^{|N|} E(n_i, u_{\text{pr}(i)}, f_{\text{pr}(i),hz(i)}) \le E_{\text{given}}(G), \tag{10}$$

where $u_{\text{pr}(i)}$ and $f_{\text{pr}(i),hz(i)}$ denote the processor and operating frequency assigned to task n_i, respectively, and satisfy

$$f_{\text{pr}(i),low} \le f_{\text{pr}(i),hz(i)} \le f_{\text{pr}(i),\max}, \tag{11}$$

for all $1 \le i \le |N|$ with $u_{\text{pr}(i)} \in U$.

We further assume that the minimum and maximum energy consumption of each task can be obtained over the available frequencies. They are denoted by $E_{\min}(n_i)$ and $E_{\max}(n_i)$, respectively:

$$E_{\min}(n_i) = \min_{u_k \in U} E(n_i, u_k, f_{k,\text{low}}), \tag{12}$$

$$E_{\max}(n_i) = \max_{u_k \in U} E(n_i, u_k, f_{k,\max}). \tag{13}$$

Thus, the total minimum and maximum energy of application G are

$$E_{\min}(G) = \sum_{i=1}^{|N|} E_{\min}(n_i), \tag{14}$$

$$E_{\max}(G) = \sum_{i=1}^{|N|} E_{\max}(n_i). \tag{15}$$

3.3. Energy allocation methods

Cyclic feedback-based energy allocation is the core of this work. To clarify the allocation process, we first introduce several key concepts used in the energy-allocation procedure.

(1) **Initial allocated energy** $E_{\text{given}}(G)$: This denotes the total energy budget that the system is allowed to consume for application G.

(2) **Actual energy** $E^{(n)}_{\text{actual}}(G)$: This denotes the actual energy consumed by G in the nth simulation round:

$$E^{(n)}_{\text{actual}}(G) = \sum_{i=1}^{|N|} E^{(n)}_{\text{actual}}(n_i). \tag{16}$$

(3) **Available energy** $E_{\text{available}}(G)$: This is defined as the difference between the initial energy budget and the total minimum energy $E_{\text{min}}(G)$, i.e.,

$$E_{\text{available}}(G) = E_{\text{given}}(G) - E_{\text{min}}(G), \tag{17}$$

which represents the total energy margin that can be redistributed among tasks.

(4) **Surplus energy** $E^{(n)}_{\text{surplus}}(G)$: This denotes the remaining energy that has not been utilized after the nth round of simulation scheduling:

$$E^{(n)}_{\text{surplus}}(G) = E_{\text{given}}(G) - E^{(n)}_{\text{actual}}(G). \tag{18}$$

(5) **Expected energy** $E^{(n)}_{\text{expected}}(n_i)$: This is the target energy assigned to task n_i in the nth simulation round, produced by the corresponding energy-allocation policy.

(6) **Average energy** $E_{\text{ave}}(n_i)$: This denotes the average predicted energy of n_i:

$$E_{\text{ave}}(n_i) = \frac{E_{\text{max}}(n_i) + E_{\text{min}}(n_i)}{2}. \tag{19}$$

(7) **Average time** $T_{\text{ave}}(n_i)$: This represents the average execution time of n_i when running at maximum frequencies across all processors:

$$T_{\text{ave}}(n_i) = \frac{1}{|U|} \sum_{k=1}^{|U|} \left(\text{EFT}(n_i, u_k, f_{k,\text{max}}) - \text{EST}(n_i, u_k, f_{k,\text{max}}) \right). \tag{20}$$

To efficiently distribute the available energy $E_{\text{available}}(G)$, we consider three representative allocation methods. Let $\Delta E_{\text{available}}(G)$ denote the portion of available energy that is to be allocated in the current distribution round.

(1) **Equal distribution method:** The available energy is evenly assigned to all tasks:

$$E_{\text{expected}}(n_i) = E_{\text{min}}(n_i) + \frac{\Delta E_{\text{available}}(G)}{|N|}. \tag{21}$$

(2) **Energy-based linear weighted allocation:** The available energy is distributed according to each task's average energy demand:

$$E_{\text{expected}}(n_i) = E_{\text{min}}(n_i) + \Delta E_{\text{available}}(G) \times W_{\text{energy}}(n_i), \tag{22}$$

where the energy weight $W_{\text{energy}}(n_i)$ is

$$W_{\text{energy}}(n_i) = \frac{E_{\text{ave}}(n_i)}{\sum_{j=1}^{|N|} E_{\text{ave}}(n_j)}.$$

(3) **Time-based linear weighted allocation:** The available energy is distributed according to each task's average execution time:

$$E_{\text{expected}}(n_i) = E_{\text{min}}(n_i) + \Delta E_{\text{available}}(G) \times W_{\text{time}}(n_i), \tag{23}$$

where the time weight $W_{\text{time}}(n_i)$ is

$$W_{\text{time}}(n_i) = \frac{T_{\text{ave}}(n_i)}{\sum_{j=1}^{|N|} T_{\text{ave}}(n_j)}.$$

Each of these three allocation methods is suitable for different scenarios. Regardless of the specific method, the surplus energy $E^{(n)}_{\text{surplus}}(G)$ should ideally be driven as close to zero as possible, in order to avoid energy waste and enhance overall system performance.

After choosing a particular energy-allocation scheme, the pre-allocated energy for each task n_i is computed as

$$E_{\text{given}}(n_i) = \min\left(E_{\text{pre}}(n_i), E_{\text{max}}(n_i)\right), \tag{24}$$

where $E_{\text{pre}}(n_i)$ is derived in a backward manner as follows:

$$E_{\text{pre}}(n_j) = \min\left(E_{\text{expected}}(n_j), E_{\text{max}}(n_j)\right), \tag{25}$$

$$E_{\text{pre}}(n_i) = E_{\text{given}}(G) - \sum_{j=i+1}^{|N|} E_{\text{pre}}(n_j) - \sum_{k=1}^{k<i} E_{\text{actual}}(n_k), \tag{26}$$

where n_j denotes nodes that appear after n_i in the pre-allocation order and $E_{\text{pre}}(n_j)$ is their pre-allocated energy, while n_k denotes nodes that appear before n_i and $E_{\text{actual}}(n_k)$ is the actual energy consumed by n_k in the previous simulation round.

3.4. Optimization objective

Based on the above system model, the energy-constrained task scheduling problem studied in this paper can be formulated as follows.

Given a task graph $G = (V, E)$ and a total energy constraint $E_{\text{given}}(G)$, the objective is to minimize the scheduling length $SL(G)$ under the energy bound:

$$\begin{aligned} &\min \quad SL(G) \\ &\text{s.t.} \quad E(G) = \sum_{t_i \in V} E(t_i, u_i, f_i) \le E_{\text{given}}(G) \end{aligned} \tag{27}$$

where $E(t_i, u_i, f_i)$ is computed according to Eq. (9), and $SL(G)$ is defined in Eq. (5).

4. Analysis of cyclic feedback mechanism

4.1. Cycling analysis

Due to the operating characteristics of DVFS [13], the schedule length of a single task on a fixed processor decreases as its energy consumption increases. Therefore, under a given total energy budget, we aim to utilize the available energy as efficiently as possible to reduce the overall schedule length. However, once a task running on a particular processor has reached the maximum frequency supported by that processor, allocating additional energy to it can no longer increase the frequency, and thus cannot further shorten its execution time.

To maximize the overall energy utilization of the system, it is necessary to allocate as much energy as possible to the processors while avoiding excessive allocation to those already operating at the maximum frequency. In heterogeneous multicore systems, tasks exhibit different computation characteristics on different processors, which further increases the complexity of energy allocation during scheduling. Assuming that the maximum operating frequency of all processors is normalized to 1.0, Table 5 shows the final results of a simulated scheduling scenario.

From Table 5, it can be observed that, after distributing surplus energy, tasks such as n_3 and n_{10} already operate at full frequency. As discussed above, additional energy assigned to these tasks is effectively wasted from the perspective of schedule-length reduction. In contrast, tasks $n_1, n_2, n_4, n_5, n_6, n_7, n_8$, and n_9 still have headroom for frequency scaling and thus remain improvable. In this situation, we should, under the global energy constraint, prioritize allocating more energy to tasks that have not yet reached their maximum frequency.

In scenarios where the energy demands of different tasks are relatively similar, it is more appropriate to distribute the surplus energy using an equal allocation strategy, as illustrated in Fig. 3. In contrast,

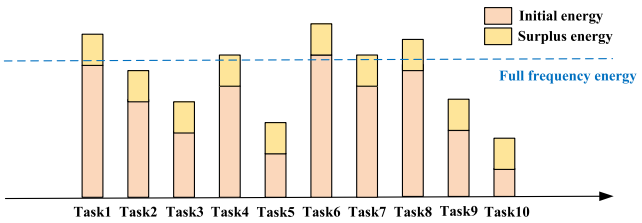


Fig. 3. Even distribution of surplus energy among tasks.

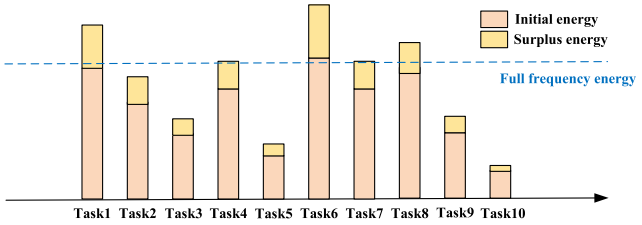


Fig. 4. Linearly weighted distribution of surplus energy among tasks.

when the energy requirements of tasks differ significantly, allocating the surplus energy via a linear weighting scheme may yield better results.

Common linear weighting strategies include energy-based linear weighting [20] and schedule-length- or time-based linear weighting [21]. Each has its own advantages under different application scenarios. For the same example, the effect of distributing surplus energy using a linear weighting strategy is shown in Fig. 4.

The above discussion describes only the initial energy-distribution phase. A key advantage of static scheduling is that it allows the scheduler to repeatedly simulate the scheduling procedure and then select the best configuration for actual execution. Based on the preceding analysis of processor frequencies, we obtain two complementary ideas for cyclic iteration:

1. Increase the energy allocation for tasks that have not yet reached full frequency as much as possible, even if this requires reducing the energy assigned to other tasks.
2. Decrease the energy allocation for tasks already running at full frequency as much as possible, and reassign the released energy to the remaining tasks.

For these two design philosophies, we next provide more detailed formulations, which form the foundation for the proposed cyclic feedback strategies.

4.2. Formula explanation

Our goal is to fully exploit the allocated energy budget so that the decision in each simulation round builds on all previous results and gradually approaches the desired target allocation and schedule length.

In the energy-based linear weighted allocation method, we introduce a dynamic adjustment parameter α, which is updated in each round and serves as a global scaling factor for the current round's energy distribution.

Specifically, the adjustment parameter in the nth round is defined as

$$\alpha_n = \frac{E_{\text{given}}(G)}{E^{(n)}_{\text{actual}}(G)}, \tag{28}$$

where $E^{(n)}_{\text{actual}}(G)$ is the total energy actually consumed in the nth round.

In the nth allocation round, to incorporate the effects of all preceding rounds, the expected energy for task n_i is written with a cumulative scaling factor as

$$E^{(n)}_{\text{expected}}(n_i) = E_{\min}(n_i) + \Delta E_{\text{available}}(G) \times W_{\text{energy}}(n_i) \times A_n \tag{29}$$

where $\Delta E_{\text{available}}(G)$ denotes the portion of available energy to be (re)allocated in the current round.

To facilitate the analysis, we denote the cumulative scaling factor as

$$A_n \triangleq \prod_{j=1}^{n} \alpha_j = \alpha_1 \times \alpha_2 \times \cdots \times \alpha_n, \tag{30}$$

which can be updated recursively as

$$A_n = A_{n-1} \cdot \alpha_n, \quad A_0 = 1. \tag{31}$$

Thus, the expected allocation $E^{(n)}_{\text{expected}}(n_i)$ in each round depends only on the current-round scaling factor α_n and the current cumulative state A_{n-1}, rather than explicitly on the entire history of previous rounds. This recursive form ensures that the cyclic feedback process can be interpreted as a finite-dimensional deterministic state-transition system.

For the equal-distribution and time-based linear weighted schemes, we follow a similar idea by introducing another dynamic adjustment parameter β, which is updated in each round and controls how aggressively we compress the energy assigned to tasks already running at full frequency.

The adjustment parameter in the nth round is defined as

$$\beta_n = \frac{E^{(n)}_{\text{actual}}(G)}{E_{\text{given}}(G)}. \tag{32}$$

In the nth round, the expected energy allocation for the equal distribution and time-based linear weighting schemes is updated as follows:

$$E^{(n)}_{\text{expected}}(n_i) = E_{\min}(n_i) + \frac{\Delta E_{\text{available}}(G)}{|N|} \times \prod_{j=1}^{n} \beta_j, \tag{33}$$

$$E^{(n)}_{\text{expected}}(n_i) = E_{\min}(n_i) + \Delta E_{\text{available}}(G) \times W_{\text{time}}(n_i) \times \prod_{j=1}^{n} \beta_j. \tag{34}$$

Here,

$$\prod_{j=1}^{n} \beta_j = \beta_1 \times \beta_2 \times \cdots \times \beta_n \tag{35}$$

represents the cumulative effect of the β factors. With this construction, the expected allocation in each round automatically reflects how much of the original budget has been used so far, and gradually drives the system toward a stable allocation that respects the global energy constraint while improving the schedule length.

4.3. Cycle termination conditions

Finite-State Interpretation of the Cyclic Feedback Process.

For a fixed application G, the task graph structure, the processor set, and the supported frequency levels of each processor are all finite. Since each task is assigned to a processor–frequency pair selected from a finite candidate set, and the scheduling procedure is deterministic, the cyclic feedback strategy can be interpreted as a state-transition process over a finite state space.

In this context, each state can be represented by the tuple $\left(E_{\text{actual}}(G), SL(G), A_n\right)$, where A_n denotes the cumulative scaling factor defined in (30). Since A_n can be updated recursively as $A_n = A_{n-1}\alpha_n$ with $A_0 = 1$, the evolution of the α-cycle (and similarly the β-cycle) induces a deterministic mapping from the current state to the next state.

After specifying the cyclic update rules, we now discuss how to determine the termination conditions. Because the α-cycle and β-cycle

encode different design philosophies, their stopping criteria are also designed differently.

Intuitively, if at some round the overall pair [total energy, total schedule length] reappears exactly as in a previous round, then subsequent iterations will enter a strictly repeating cycle and cannot generate new schedules. The following lemma formalizes this observation under a deterministic scheduling policy.

Lemma 1. *Given a fixed application G, a fixed energy-allocation policy, and a deterministic scheduling algorithm, any given* [total energy, total schedule length] *pair corresponds to a unique scheduling strategy.*

Proof. Assume, for the sake of contradiction, that there exist two different scheduling strategies S_1 and S_2 under the same initial energy-consumption constraint, such that they produce the same pair of values:

$$\big(E(G,S_1), SL(G,S_1)\big) = \big(E(G,S_2), SL(G,S_2)\big),$$

while $S_1 \neq S_2$.

For a fixed application G and a fixed energy-allocation policy, the scheduling algorithm (e.g., HEFT-style with a given priority rule) is deterministic: once the per-task energy allocation and the task-order priorities are fixed, the assignment of tasks to processors and their start/finish times are uniquely determined. A different schedule S_1 versus S_2 necessarily implies at least one task with a different processor-frequency assignment or a different start time, which in turn changes either the individual energy or the completion time of that task, and thus changes the aggregate pair $\big(E(G), SL(G)\big)$. This contradicts the assumption that S_1 and S_2 yield exactly the same [total energy, total schedule length].

Since the state space is finite and the state-transition rule is deterministic, the sequence of states generated by the cyclic feedback process must eventually revisit a previously encountered state. By the pigeonhole principle, repetition is inevitable, which justifies the cycle-detection termination mechanism.

Therefore, under a deterministic scheduling policy, each [total energy, total schedule length] pair corresponds to a unique scheduling strategy. □

Based on Lemma 1, we can further analyze how cycles arise in the α-loop strategy and design a natural termination condition.

Theorem 1. *In the α-cycle strategy, if the* [total energy, total schedule length] *pairs in the kth and mth rounds ($m > k$) are identical, then the scheduling outcome of the $(m+1)$-th round is identical to that of the $(k+1)$-th round.*

Proof. Suppose that in rounds k and m we have

$$E^{(k)}_{\text{actual}}(G) = E^{(m)}_{\text{actual}}(G), \quad SL^{(k)}(G) = SL^{(m)}(G),$$

with $m > k$. By definition of α_n in (28), the respective scaling factors for rounds k and m satisfy

$$\alpha_k = \frac{E_{\text{given}}(G)}{E^{(k)}_{\text{actual}}(G)} = \frac{E_{\text{given}}(G)}{E^{(m)}_{\text{actual}}(G)} = \alpha_m.$$

Because the energy-allocation policy and scheduling algorithm are deterministic, the same pair $\big(E^{(k)}_{\text{actual}}(G), SL^{(k)}(G)\big) = \big(E^{(m)}_{\text{actual}}(G), SL^{(m)}(G)\big)$ implies, by Lemma 1, that the underlying schedules at rounds k and m are identical. Consequently, the per-task energy allocation patterns used to construct $E^{(k+1)}_{\text{expected}}(n_i)$ and $E^{(m+1)}_{\text{expected}}(n_i)$ are also identical.

Since the cumulative scaling factor has been denoted by A_n and updated recursively as $A_n = A_{n-1}\alpha_n$, the expected-energy expression in each round depends only on the current state $\big(E^{(n)}_{\text{actual}}(G), SL^{(n)}(G), A_n\big)$. Once $\alpha_k = \alpha_m$ and the schedules in rounds k and m are identical, we also have $A_k = A_m$. Therefore, the update rule receives identical state inputs in rounds $(k+1)$ and $(m+1)$, which implies identical per-task expected allocations.

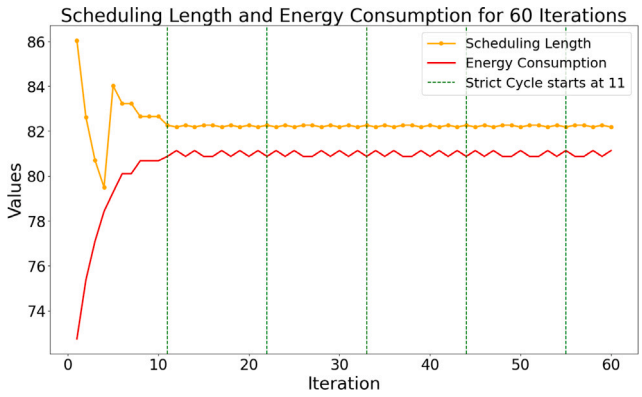


Fig. 5. α-loop iterative strategy over 60 iterations.

Therefore, the inputs to the scheduler in rounds $(k+1)$ and $(m+1)$ are the same, and the deterministic nature of the algorithm ensures that the resulting schedules are identical:

$$E^{(k+1)}_{\text{actual}}(G) = E^{(m+1)}_{\text{actual}}(G), \quad SL^{(k+1)}(G) = SL^{(m+1)}(G).$$

Hence, once two identical states occur, the subsequent evolution of the cyclic feedback process is periodic.

This proves the theorem. □

Fig. 5 illustrates the variation of total energy and schedule length over 60 iterations for a representative instance using the α-loop strategy. A relatively low schedule length is already achieved by the fourth iteration. As the number of iterations increases, the energy of each task gradually approaches its maximum allowed value $E_{\max}(n_i)$, so that some tasks can no longer be further accelerated. This restriction propagates to other tasks via the global energy constraint and may cause the overall schedule length to increase in certain rounds.

A closer inspection reveals that, in this example, starting from the 11-th iteration, the results form a repeating pattern with a period of 11 iterations: after several rounds of scaling, the [energy, schedule length] pair appears exactly the same as in a previous period. Motivated by Theorem 1, we record the [energy, schedule length] pair in each iteration and terminate the α-loop once the same pair is detected for the second time. At that point, the schedule with the smallest schedule length among all recorded iterations is selected as the final output.

On the other hand, since the α-loop continuously increases the energy of tasks that have not yet reached full frequency, certain iterations may yield total energy $E^{(n)}_{\text{actual}}(G) > E_{\text{given}}(G)$. Therefore, after the loop terminates, we must perform an additional energy-feasibility check on the candidate schedule with the minimum schedule length to ensure that its total energy does not exceed $E_{\text{given}}(G)$. If the best schedule violates the energy constraint, we successively examine the schedule with the second smallest schedule length, and so on, until a schedule that satisfies the energy constraint is found. This schedule is then selected as the final output.

Similarly, we analyze the convergence behavior of the β-loop strategy and derive its termination condition.

Fig. 6 shows the variation of total energy and schedule length over 80 iterations using the β-loop on a representative instance. A relatively small schedule length is already obtained in the second iteration. As the iteration proceeds, the energy of fully loaded tasks is gradually reduced and approaches their minimum allowed energy $E_{\min}(n_i)$, which limits the possibility of further reallocation and may cause the total schedule length to increase again in later iterations.

Because the β-loop continually compresses the energy allocated to full-frequency tasks and redistributes it to other tasks, each scaling factor β_n satisfies $\beta_n \leq 1$. Together with

$$E^{(n)}_{\text{actual}}(G) = \sum_{i=1}^{|N|} E^{(n)}_{\text{actual}}(n_i),$$

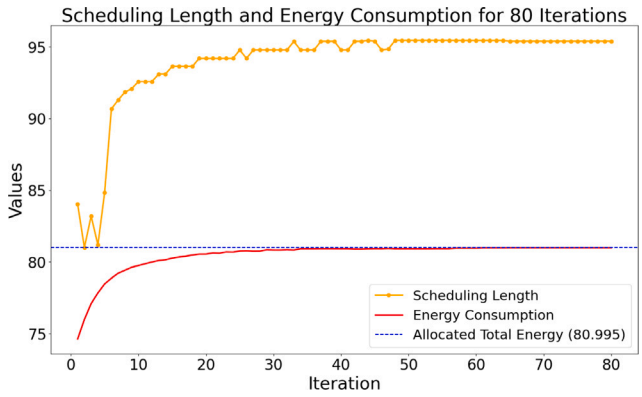


Fig. 6. β-loop iterative strategy over 80 iterations.

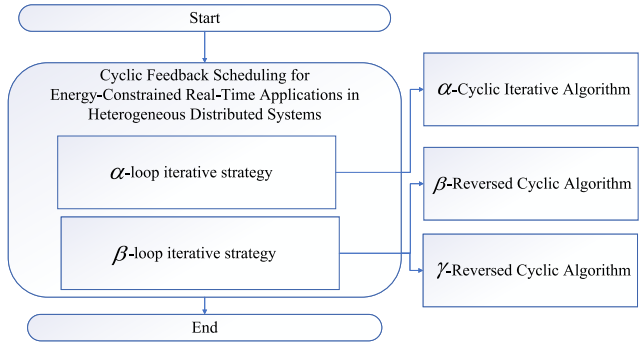


Fig. 7. Overall architecture of the CFSECC framework. The α-loop and β-loop define two cyclic iteration structures. From the α-loop, the α-Cyclic Iterative Algorithm is derived, while the β-loop gives rise to the β-Reversed and γ-Reversed Cyclic Algorithms.

and the backward pre-allocation scheme in (26), it follows that $E^{(n)}_{\text{actual}}(G)$ never exceeds $E_{\text{given}}(G)$ and instead converges monotonically toward this upper bound.

As the number of iterations increases, the change in schedule length between successive iterations becomes smaller and smaller. Therefore, we define the termination condition for the β-loop as follows: the loop stops when either

$$\left|\frac{E^{(n+1)}_{\text{actual}}(G) - E^{(n)}_{\text{actual}}(G)}{E^{(n)}_{\text{actual}}(G)}\right| < \epsilon, \tag{36}$$

for a user-defined threshold ϵ, or when a preset maximum number of iterations (e.g., 50) is reached. After termination, we select from all recorded iterations the schedule with the minimum schedule length as the final result.

5. Proposed cyclic feedback scheduling strategy

To provide an overview of the interaction between the iterative strategies and the three energy-allocation schemes, the overall structure of the CFSECC framework is illustrated in Fig. 7.

Based on the analysis in the previous section, we now present a cyclic feedback scheduling strategy (CFSECC), which iteratively refines the energy allocation in static simulations by exploiting feedback information from previous rounds. Under a given global energy constraint, CFSECC aims to improve energy utilization efficiency and reduce the schedule length. According to the initial allocation scheme for the available energy, the overall strategy consists of three concrete algorithms: the α-Cyclic Iterative Algorithm, the β-Reverse Cyclic Algorithm, and the γ-Reverse Cyclic Algorithm.

As illustrated in Fig. 8, the three cyclic strategies are integrated within a unified feedback framework, rather than operating independently. In the offline phase, the scheduler first computes the minimum/maximum energy requirements of each task and derives the available energy budget according to the system model. It then selects one of the three initial allocation schemes (equal distribution, energy-based linear weighting, or time-based linear weighting), and iteratively refines the allocation by applying the α or β/γ feedback factors in multiple simulation rounds. In the online phase, only the best static schedule obtained offline is applied to the actual system.

5.1. α-Cyclic iterative algorithm

This subsection first introduces the α-Cyclic Iterative Algorithm. Following the principle of allocating additional energy preferentially to tasks that have not yet reached their maximum operating frequency, the algorithm adopts an energy-based linear weighting scheme for the initial allocation. After each scheduling round, the feedback factor α_n is updated according to the ratio between the actual energy consumption and the given energy constraint, thereby proportionally scaling the expected energy in the next round. In this manner, the algorithm iteratively moves toward maximizing the utilization of the available energy.

The pseudocode below presents the main workflow of the α-Cyclic Iterative Algorithm. The algorithm begins by sorting tasks according to their upward rank values and, under a given feedback factor α_n, selects for each task the processor–frequency combination that satisfies the energy constraint while yielding the earliest completion time. Across successive iterations, the feedback factor α_n is dynamically updated based on the energy utilization of the previous round, enabling the allocated energy to progressively approach the prescribed energy budget.

Algorithm 1 The α-Cyclic Iterative Algorithm

Require: $G = (N, M, C, W)$, $U = \{u_1, u_2, ..., u_{|U|}\}$, $E_{\text{given}}(G)$.
Ensure: $SL(G)$, $E(G)$.
Arrange the tasks in list $dlist$ in descending order based on $rank_u$.
for $n_i \in N$ **do**
  Compute $E_{\min}(n_i)$ and $E_{\max}(n_i)$ using Eq. (12) and Eq. (13).
  Compute $E_{\text{ave}}(n_i)$ using Eq. (19).
Compute $E_{\min}(G)$ and $E_{\max}(G)$ using Eq. (14) and Eq. (15).
Compute $E_{\text{available}}(G)$ and initialize $E^{(1)}_{\text{actual}}(G)$.
Initialize $\alpha_1 = \frac{E_{\text{given}}(G)}{E^{(1)}_{\text{actual}}(G)}$.
while $dlist$ is not empty **do**
  $n_i \leftarrow dlist$.out().
  $AFT(n_i) \leftarrow \infty$.
  Compute $E_{\text{given}}(n_i)$ using Eq. (24).
  for each $u_k \in U$ **do**
    for each $f_{k,h} \in [f_{k,\text{low}}, f_{k,\text{max}}]$ **do**
      Recalculate $E(n_i, u_k, f_{k,h})$ using Eq. (9).
      if $E(n_i, u_k, f_{k,h}) > E_{\text{given}}(n_i)$ **then**
        continue.
      Compute $EFT(n_i, u_k, f_{k,h})$.
      if $EFT(n_i, u_k, f_{k,h}) < AFT(n_i)$ **then**
        $pr(i) \leftarrow k$, $f_{pr(i),hz(i)} \leftarrow f_{k,h}$.
        $E(n_i, u_{pr(i)}, f_{pr(i),hz(i)}) \leftarrow E(n_i, u_k, f_{k,h})$.
        $AFT(n_i) \leftarrow EFT(n_i, u_k, f_{k,h})$.
Compute actual energy consumption $E(G)$.
Compute the schedule length $SL(G) \leftarrow AFT(n_{\text{exit}})$.
Reverse the DAG, resort tasks to obtain a new $dlist$, and repeat the procedure.
Compare both $SL(G)$ values and choose the smaller as the final output.
return $E(G), SL(G)$.

In practice, the α-Cyclic Iterative Algorithm operates as follows. First, tasks are sorted by their upward rank, and each task is assigned to the processor–frequency pair that yields the earliest finish time while satisfying its energy bound. Second, in each iteration, the feedback

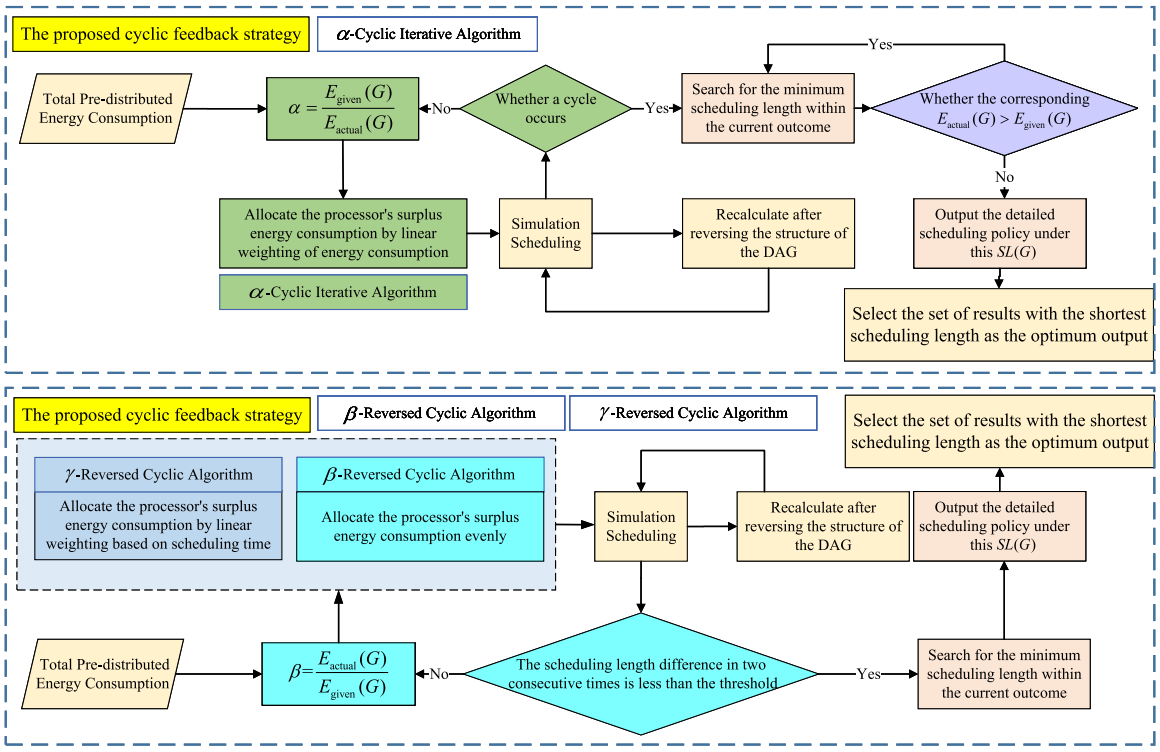


Fig. 8. Structural overview of the complete CFSECC framework. The proposed cyclic feedback strategy unifies the α-Cyclic, β-Reversed, and γ-Reversed energy-adjustment mechanisms within a common deterministic state-transition process. Although each component adopts a different energy redistribution policy, they share the same simulation scheduling core and cycle-detection mechanism. The final scheduling policy is selected from all cyclic outcomes based on the minimum scheduling length criterion.

factor $\alpha_n = \frac{E_{\text{given}}(G)}{E^{(n)}_{\text{actual}}(G)}$ scales the expected energy so that the actual energy consumption gradually approaches the given energy budget. Third, the algorithm is executed on both the original DAG and the reversed DAG, and the better schedule is selected to account for both forward and reverse critical paths.

The main time complexity of the α-Cyclic Iterative Algorithm lies in the nested traversal over tasks, processors, and discrete frequencies. Let $|N|$ be the number of tasks, $|U|$ the number of processors, and $|F|$ the number of frequency levels. For each iteration, selecting the best processor–frequency combination for all tasks requires $O(|N| \cdot |U| \cdot |F|)$ time. Considering the dynamic update of the task list and the fixed upper bound on iteration rounds, the overall complexity of the scheduling process is $O(|N|^2 \cdot |U| \cdot |F|)$. Since the algorithm is executed twice (on the original and reversed DAGs), the total time complexity remains $O(|N|^2 \cdot |U| \cdot |F|)$ up to a constant factor.

5.2. β-Reverse cyclic algorithm

This section introduces the β-Reverse Cyclic Algorithm. Unlike the α-cycle, which emphasizes increasing the energy assigned to tasks that have not reached full frequency, the β-reverse cycle starts from the perspective of reducing the energy allocated to tasks already operating at maximum frequency. It adopts an average energy distribution scheme for the initial allocation and then updates the feedback factor according to $\beta_n = E^{(n)}_{\text{actual}}(G)/E_{\text{given}}(G)$, which compresses the expected energy in the next iteration. In this way, the energy "saved" from full-frequency tasks is redistributed to the remaining tasks.

The pseudocode below summarizes the main procedure of the β-Reverse Cyclic Algorithm. The algorithm first sorts tasks according to their upward rank values and computes the minimum, maximum, and average energy requirements for each task. Based on an even distribution of the available energy, the initial $E_{\text{expected}}(n_i)$ values are obtained. During each scheduling iteration, the expected energy is adjusted using the feedback factor β_n, after which the algorithm selects, for every task, the processor–frequency pair that satisfies the energy constraint and yields the earliest completion time.

Algorithm 2 The β-Reverse Cyclic Algorithm

Require: $G = (N, M, C, W)$, $U = \{u_1, u_2, ..., u_{|U|}\}$, $E_{\text{given}}(G)$.
Ensure: $SL(G)$, $E(G)$.

Arrange the tasks in list $dlist$ in descending order based on $rank_u$.
for each $n_i \in N$ **do**
    Calculate $E_{\min}(n_i)$ and $E_{\max}(n_i)$ using Eq. (12) and Eq. (13).
    Calculate $E_{\text{ave}}(n_i)$ using Eq. (19).
Calculate $E_{\min}(G)$ and $E_{\max}(G)$ using Eq. (14) and Eq. (15).
Compute $E_{\text{available}}(G)$ and initialize β_1.
for each $n_i \in N$ **do**
    Calculate $E_{\text{expected}}(n_i)$ using Eq. (21).
while $dlist$ is not empty **do**
    $n_i \leftarrow dlist$.out().
    Calculate $E_{\text{given}}(n_i)$ using Eq. (24).
    $AFT(n_i) \leftarrow \infty$.
    for each $u_k \in U$ **do**
        for each frequency $f_{k,h}$ in ascending order **do**
            Recalculate $E(n_i, u_k, f_{k,h})$ using Eq. (9).
            if $E(n_i, u_k, f_{k,h}) \le E_{\text{given}}(n_i)$ **then**
                Compute $EFT(n_i, u_k, f_{k,h})$.
                if $EFT(n_i, u_k, f_{k,h}) < AFT(n_i)$ **then**
                    $pr(i) \leftarrow k$, $f_{pr(i),hz(i)} \leftarrow f_{k,h}$.
                    $E(n_i, u_{pr(i)}, f_{pr(i),hz(i)}) \leftarrow E(n_i, u_k, f_{k,h})$.
                    $AFT(n_i) \leftarrow EFT(n_i, u_k, f_{k,h})$.
Compute actual energy consumption $E(G)$.
Compute the schedule length $SL(G) \leftarrow AFT(n_{\text{exit}})$.
Reverse the DAG, resort tasks to obtain a new $dlist$, and repeat the procedure.
Compare both $SL(G)$ values and choose the smaller as the final output.
return $E(G), SL(G)$.

Compared with the α-cycle strategy, the β-Reverse Cyclic Algorithm has two distinctive features. First, the feedback factor $\beta_n = \frac{E^{(n)}_{\text{actual}}(G)}{E_{\text{given}}(G)}$ satisfies $\beta_n < 1$, so the expected energy is gradually scaled down across iterations, effectively cutting the energy assigned to tasks already operating at high or full frequencies and reallocating it to other tasks. Second, as the actual energy $E^{(n)}_{\text{actual}}(G)$ approaches the given budget from above and each task is bounded by $E_{\min}(n_i)$, the global energy constraint $E^{(n)}_{\text{actual}}(G) \le E_{\text{given}}(G)$ is naturally preserved.

The time complexity of the β-Reverse Cyclic Algorithm is similar to that of the α-Cyclic Iterative Algorithm. The dominant cost comes from the nested loops over tasks, processors, and frequency levels, yielding a per-iteration complexity of $O(|N| \cdot |U| \cdot |F|)$. Taking into account the dynamic update of the task list and the bounded number of feedback iterations, the overall scheduling complexity becomes $O(|N|^2 \cdot |U| \cdot |F|)$. The additional run on the reversed DAG only contributes a constant factor, so the total time complexity remains $O(|N|^2 \cdot |U| \cdot |F|)$.

5.3. γ-Reverse cyclic algorithm

This section introduces the γ-Reverse Cyclic Algorithm. Similar to the β-Reverse Cyclic approach, it follows the principle of reducing the energy allocated to tasks already operating at full frequency. The key distinction lies in its initial allocation scheme, which employs a time-based linear weighting derived from the average execution time of each task. This makes the method particularly suitable for scenarios where task execution times vary significantly.

The following pseudocode outlines the main procedure of the γ-Reverse Cyclic Algorithm. The algorithm begins by sorting tasks according to their upward ranking values and computing their minimum, maximum, and average energy requirements. It then performs an initial allocation of available energy using the time-based weighting factor $W_{\text{time}}(n_i)$. Subsequently, similar to the β-Reverse Cyclic method, a feedback factor β_n is introduced for multi-round iterations, progressively refining the schedule length while strictly adhering to the given energy constraints.

Algorithm 3 The γ-Reverse Cyclic Algorithm

Require: $G = (N, M, C, W)$, $U = \{u_1, u_2, \ldots, u_{|U|}\}$, $E_{\text{given}}(G)$
Ensure: $SL(G)$, $E(G)$

Arrange the tasks in list $dlist$ in descending order based on $rank_u$
for each $n_i \in N$ **do**
  Calculate $E_{\min}(n_i)$ and $E_{\max}(n_i)$ using Eq. (12) and Eq. (13)
  Calculate $E_{\text{ave}}(n_i)$ using Eq. (19)
Calculate $E_{\min}(G)$ and $E_{\max}(G)$ using Eq. (14) and Eq. (15)
Compute $E_{\text{available}}(G)$ using Eq. (17) and initialize β_1
for each $n_i \in N$ **do**
  Calculate $E_{\text{expected}}(n_i)$ using Eq. (23)
while $dlist$ is not empty **do**
  $n_i \leftarrow dlist$.out()
  Calculate $E_{\text{given}}(n_i)$ using Eq. (24)
  $AFT(n_i) \leftarrow \infty$
  for each $u_k \in U$ **do**
    for each frequency $f_{k,h}$ in ascending order **do**
      Recalculate $E(n_i, u_k, f_{k,h})$ using Eq. (9)
      if $E(n_i, u_k, f_{k,h}) \le E_{\text{given}}(n_i)$ **then**
        Compute $EFT(n_i, u_k, f_{k,h})$
        if $EFT(n_i, u_k, f_{k,h}) < AFT(n_i)$ **then**
          $pr(i) \leftarrow k$, $f_{pr(i),hz(i)} \leftarrow f_{k,h}$
          $E(n_i, u_{pr(i)}, f_{pr(i),hz(i)}) \leftarrow E(n_i, u_k, f_{k,h})$
          $AFT(n_i) \leftarrow EFT(n_i, u_k, f_{k,h})$
Compute the actual energy consumption $E(G)$
Calculate the schedule length $SL(G) \leftarrow AFT(n_{\text{exit}})$
Reverse the DAG, resort tasks to obtain a new $dlist$, and repeat the procedure
Compare both $SL(G)$ values and choose the smaller as the final output
return $E(G), SL(G)$

Table 4
Processor parameters.

u_k	$P_{k,\text{ind}}$	$C_{k,\text{ef}}$	m_k	$f_{k,\text{low}}$	$f_{k,\max}$
u_1	0.03	0.8	2.9	0.22	1.0
u_2	0.04	0.8	2.5	0.21	1.0
u_3	0.07	1.0	2.5	0.19	1.0

The γ-Reverse Cyclic Algorithm shares the same feedback mechanism with the β-Reverse Cyclic Algorithm. The key difference lies in the initial allocation phase: by exploiting the average execution time $T_{\text{ave}}(n_i)$, the algorithm assigns larger energy shares to tasks that are more time-consuming and thus more influential on the critical path. On top of this, the iterative scaling via β_n drives the energy allocation to converge while respecting the global constraint.

In all algorithmic procedures, the actual energy consumption of a task under a given processor–frequency pair is computed according to Eq. (9), ensuring consistency between the analytical model and the implementation.

In terms of time complexity, the γ-Reverse Cyclic Algorithm requires the same nested traversal over tasks, processors, and frequency levels. The per-iteration complexity is $O(|N| \cdot |U| \cdot |F|)$, and, accounting for dynamic list updates and the bounded number of feedback rounds, the total scheduling complexity is $O(|N|^2 \cdot |U| \cdot |F|)$. The additional run on the reversed DAG only increases the constant factor, so the overall time complexity remains $O(|N|^2 \cdot |U| \cdot |F|)$.

The three components in CFSECC play complementary roles in the cyclic feedback framework.

The α-Cyclic strategy performs forward energy reallocation based on the current scheduling imbalance, improving energy distribution toward critical-path tasks.

The β-Reverse strategy adjusts energy allocation from a reverse perspective, compensating for potential over-allocation in earlier rounds.

The γ-Reverse strategy further refines energy distribution under strict energy bounds, preventing oscillation and promoting convergence.

Together, these components form an integrated cyclic energy-adjustment mechanism. Removing any component would weaken the ability to handle specific imbalance patterns during iteration.

5.4. Example of the cyclic feedback scheduling strategy

We consider a small example with ten tasks. The task graph is shown in Fig. 1, the execution time matrix is given in Table 3, and the processor parameters are summarized in Table 4. The total energy budget is set to $E_{\text{given}}(G) = 70.995$. On this basis, we apply the α-Cyclic Iterative Algorithm, the β-Reverse Cyclic Algorithm, and the γ-Reverse Cyclic Algorithm to illustrate the effectiveness of the proposed cyclic feedback strategy.

Table 5 and Fig. 9 report the scheduling results before applying the α-Cyclic Iterative Algorithm, while Table 6 and Fig. 10 show the results after iteration. The schedule length is reduced from 106.0818 to 89.1111 (a 16.0% improvement), whereas the total energy consumption increases only slightly from 68.2629 to 70.1114, remaining strictly below $E_{\text{given}}(G)$.

The AST and AFT values have been recalculated strictly, and Table 5 and Fig. 10 have been updated accordingly to ensure consistency.

Table 7 and Fig. 11 present the scheduling results before applying the β-Reverse Cyclic Algorithm, while Table 8 and Fig. 12 show the results after feedback iteration. The schedule length is reduced from 101.0996 to 81.5085 (a 19.3% improvement), and the total energy increases from 68.0912 to 70.3882, which still satisfies the global energy constraint.

By comparing the three feedback strategies on this ten-task example, we observe that all of them can significantly reduce the schedule length while strictly satisfying the global energy constraint. Nonetheless, they

Table 5
The scheduling result before the α-Cyclic Iterative Algorithm.

n_i	$E_{given}(n_i)$	u_{n_i}	$f(n_i)$	$AST(n_i)$	$AFT(n_i)$	$E(n_i)$
n_1	6.9082	u_3	0.7700	0.0000	11.6883	6.8992
n_3	8.1440	u_1	0.9300	23.6883	35.5163	8.0214
n_4	7.7144	u_2	1.0000	20.6883	28.6883	6.7200
n_2	8.2997	u_1	0.8600	35.5163	50.6325	8.2622
n_5	5.7545	u_3	0.5900	11.6883	28.6375	5.7183
n_6	6.6816	u_3	0.7400	28.6375	40.7996	6.5805
n_9	9.9355	u_1	0.7900	51.6883	74.4731	9.8849
n_7	5.0888	u_1	0.9200	74.4731	82.0818	5.0078
n_8	5.2974	u_2	0.6500	69.6325	86.5556	5.2885
n_{10}	7.1710	u_2	1.0000	99.0818	106.0818	5.8800
$E(G) = 68.2629,\ SL(G) = 106.0818$						

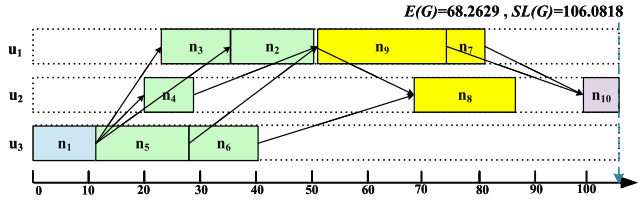


Fig. 9. The scheduling results before using the α-Cyclic Iterative Algorithm.

Table 6
The scheduling result after the α-Cyclic Iterative Algorithm.

n_i	$E_{given}(n_i)$	u_{n_i}	$f(n_i)$	$AST(n_i)$	$AFT(n_i)$	$E(n_i)$
n_1	7.3488	u_3	0.8100	0.0000	11.1111	7.3388
n_3	8.7607	u_1	0.9700	23.1111	34.4513	8.6454
n_4	8.2754	u_2	1.0000	20.1111	28.1111	6.7200
n_2	8.8965	u_3	0.5000	11.1111	47.1111	8.8840
n_5	6.1156	u_2	0.6300	28.1111	48.7460	6.0259
n_6	7.1174	u_1	0.7800	34.4513	51.1180	6.9865
n_9	10.6141	u_2	1.0000	63.1111	75.1111	10.0800
n_7	5.4719	u_1	0.9600	51.1180	58.4097	5.4008
n_8	5.7365	u_1	1.0000	66.1111	71.1111	4.1500
n_{10}	7.7037	u_2	1.0000	82.1111	89.1111	5.8800
$E(G) = 70.1114,\ SL(G) = 89.1111$						

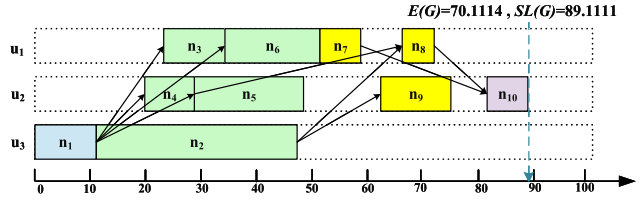


Fig. 10. The scheduling results after using the α-Cyclic Iterative Algorithm.

Table 7
The scheduling result before the β-Reverse Cyclic Algorithm.

n_i	$E_{given}(n_i)$	u_{n_i}	$f(n_i)$	$AST(n_i)$	$AFT(n_i)$	$E(n_i)$
n_1	7.5500	u_3	0.8200	0.0000	10.9756	7.4512
n_3	7.1170	u_1	0.8600	22.9756	35.7663	6.9911
n_4	7.2734	u_2	1.0000	19.9756	27.9756	6.7200
n_2	7.9261	u_1	0.8300	35.7663	51.4290	7.7692
n_5	7.3523	u_3	0.7400	10.9756	24.4891	7.3117
n_6	7.4134	u_3	0.8100	24.4891	35.6002	7.3388
n_9	8.2618	u_1	0.7000	51.4290	77.1432	8.0836
n_7	6.4872	u_3	0.6000	58.7663	77.0996	6.3957
n_8	6.0462	u_1	1.0000	77.1432	82.1432	4.1500
n_{10}	8.7838	u_2	1.0000	94.0996	101.0996	5.8800
$E(G) = 68.0912,\ SL(G) = 101.0996$						

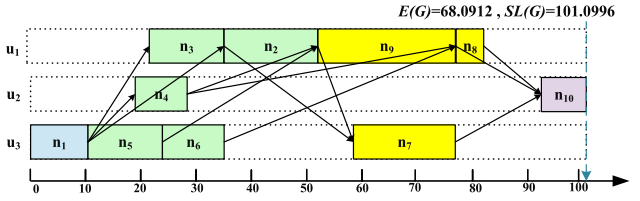


Fig. 11. The scheduling results before using the β-Reverse Cyclic Algorithm.

Table 8
The scheduling result after the β-Reverse Cyclic Algorithm.

n_i	$E_{given}(n_i)$	u_{n_i}	$f(n_i)$	$AST(n_i)$	$AFT(n_i)$	$E(n_i)$
n_1	13.4400	u_3	1.0000	0.0000	9.0000	9.6300
n_3	14.0240	u_1	1.0000	21.0000	32.0000	9.1300
n_4	10.9058	u_2	1.0000	18.0000	26.0000	6.7200
n_2	10.4227	u_3	0.5900	9.0000	39.5085	10.2930
n_5	6.1895	u_2	0.6400	26.0000	46.3125	6.1373
n_6	6.2892	u_1	0.7300	32.0000	49.8082	6.2536
n_9	7.0870	u_2	0.7600	55.5085	71.2979	6.9921
n_7	5.2683	u_1	0.9400	49.8082	57.2550	5.2023
n_8	4.8849	u_1	1.0000	58.5085	63.5085	4.1500
n_{10}	6.4868	u_2	1.0000	74.5085	81.5085	5.8800
$E(G) = 70.3882,\ SL(G) = 81.5085$						

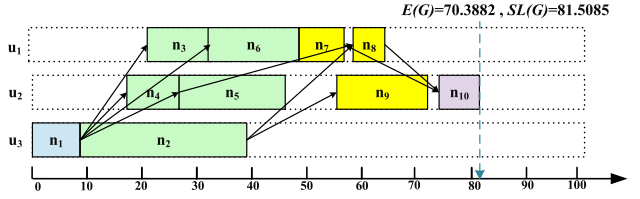


Fig. 12. The scheduling results after using the β-Reverse Cyclic Algorithm.

exhibit different advantages in different scenarios. The α-Cyclic Iterative Algorithm is generally effective when the energy budget is relatively loose and the task energy requirements do not differ dramatically. The β-Reverse Cyclic Algorithm achieves the shortest schedule length in this example and is particularly suitable for reallocating energy from fully utilized tasks to others. The γ-Reverse Cyclic Algorithm benefits applications where task execution times vary considerably, as it explicitly emphasizes time-critical tasks through the time-based weighting.

In this example, the β-Reverse Cyclic Algorithm yields the smallest final schedule length. Therefore, we adopt the scheduling result given in Table 8 as the final static schedule for this DAG, and use it as a reference configuration in the subsequent experimental evaluation.

For the γ-Reverse Cyclic Algorithm, Table 9 and Fig. 13 show the scheduling results before feedback, and Table 10 and Fig. 14 display the results after iteration. In this example, the schedule length decreases from 89.8458 to 84.0441 (a 6.46% improvement), and the total energy consumption is 70.3882 in both cases, satisfying the given energy budget.

5.5. Complexity analysis

Let $|N|$ denote the number of tasks in a DAG, $|U|$ denote the number of heterogeneous processors, and $|F|$ denote the number of available DVFS frequency levels. In each scheduling round, the dominant operation is to evaluate feasible task–processor–frequency combinations while updating precedence-aware ready times and earliest finish times. Therefore, the dominant cost of one scheduling round is bounded by

Table 9
The scheduling result before the γ-Reverse Cyclic Algorithm.

n_i	$E_{given}(n_i)$	u_{n_i}	$f(n_i)$	$AST(n_i)$	$AFT(n_i)$	$E(n_i)$
n_1	7.4233	u_3	0.81	0.0000	11.1111	7.3388
n_3	7.4828	u_1	0.89	23.1111	35.4707	7.4230
n_4	6.9539	u_2	1	20.1111	28.1111	6.7200
n_2	8.8737	u_3	0.49	11.1111	47.8458	8.7454
n_5	6.6902	u_2	0.68	28.1111	47.2288	6.5964
n_6	7.213	u_1	0.79	35.4707	51.9264	7.1391
n_9	9.5281	u_2	0.95	63.8458	76.4774	9.3943
n_7	5.556	u_1	0.97	51.9264	59.1429	5.5016
n_8	4.7419	u_1	1	66.8458	71.8458	4.1500
n_{10}	7.9863	u_2	1	82.8458	89.8458	5.8800
$E(G) = 70.3882,\ SL(G) = 89.8458$						

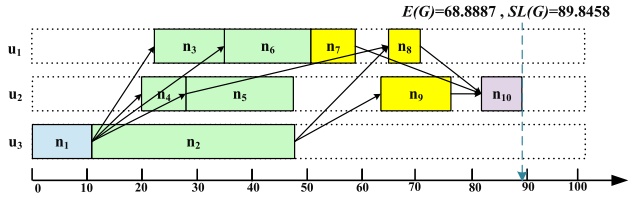


Fig. 13. The scheduling results before using the γ-Reverse Cyclic Algorithm.

Table 10
The scheduling result after the γ-Reverse Cyclic Algorithm.

n_i	$E_{given}(n_i)$	u_{n_i}	$f(n_i)$	$AST(n_i)$	$AFT(n_i)$	$E(n_i)$
n_1	7.7028	u_3	0.8400	0.0000	10.7143	7.6789
n_4	7.0012	u_2	1.0000	26.0441	34.0441	6.7200
n_5	6.7052	u_2	0.6800	45.9265	65.0441	6.5964
n_3	7.4222	u_1	0.8800	22.7143	35.2143	7.2774
n_6	7.1427	u_3	0.7900	34.6517	46.0441	7.1170
n_2	9.6864	u_1	0.9400	35.2143	49.0441	9.6614
n_7	6.8588	u_1	1.0000	53.0441	60.0441	5.8100
n_8	5.5880	u_1	1.0000	61.0441	66.0441	4.1500
n_9	10.9818	u_2	1.0000	65.0441	77.0441	10.0800
n_{10}	7.4697	u_2	1.0000	77.0441	84.0441	5.8800
$E(G) = 70.3882,\ SL(G) = 84.0441$						

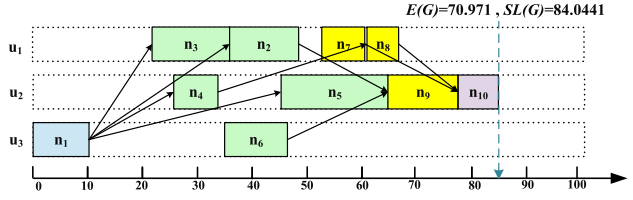


Fig. 14. The scheduling results after using the γ-Reverse Cyclic Algorithm.

$O(|N|^2|U\|F|)$, where the quadratic term mainly reflects precedence-aware task selection and ready-time update operations over the task graph.

For the three cyclic feedback variants, let I_α, I_β, and I_γ denote the numbers of cyclic iterations before their corresponding termination conditions are triggered. The computational costs of the α-, β-, and γ-variants can then be expressed as $O(I_\alpha|N|^2|U\|F|)$, $O(I_\beta|N|^2|U\|F|)$, and $O(I_\gamma|N|^2|U\|F|)$, respectively. Since CFSECC selects the best schedule from the three variants, its overall complexity is $O((I_\alpha + I_\beta + I_\gamma)|N|^2|U\|F|)$. If $I_{\max} = \max\{I_\alpha, I_\beta, I_\gamma\}$, the worst-case bound can also be written as $O(I_{\max}|N|^2|U\|F|)$ by ignoring constant factors. This analysis shows that the additional cost of CFSECC is mainly controlled by the number of cyclic iterations, which is further examined empirically in Section 6.4.

6. Experiments

We refer to the cyclic feedback scheduling strategy for energy-consumption constrained heterogeneous distributed real-time applications as CFSECC. To verify the effectiveness of the proposed strategy, FFT and GE applications are selected in the experiments, as these two applications are widely adopted in heterogeneous distributed embedded systems [36]. The parameters for applications and processors in this experiment follow [17–19,21]: $10\,\text{ms} \le \omega_{i,k} \le 100\,\text{ms}$, $10\,\text{ms} \le c_{i,j} \le 100\,\text{ms}$, $0.03 \le P_{k,\text{ind}} \le 0.07$, $0.8 \le C_{k,e,f} \le 1.2$, $2.5 \le m_k \le 3.0$, and $f_{k,\max} = 1\,\text{GHz}$. The processor frequency step is set to 0.01 GHz.

In the comparison experiments with existing approaches, MSLECC [17] (Minimizing Schedule Length of Energy Consumption Constrained Algorithm), ESECC [18] (Efficient Scheduling Algorithm for Energy Consumption Constrained), EECC [19] (Enhanced Energy-Constrained Scheduling), ISAECC [20] (Improved Scheduling Approach for Energy Consumption Constrained Algorithm), TSSA [21] (Task Structure-Aware Scheduling of Energy-Constrained Parallel Applications) and SEDC [22] (Scheduling based on Energy Difference Coefficient) share the same application model and optimization objective as CFSECC. Therefore, they are selected as baseline algorithms, and comparative experiments are conducted under different energy constraints and different task quantities. Specifically, the CFSECC framework consists of three cyclic variants: the α-Cyclic Iterative Algorithm, the β-Reverse Cyclic Algorithm, and the γ-Reverse Cyclic Algorithm. Among the schedules produced by these three variants, the one with the smallest makespan is selected as the final output. In the experimental figures, the labels "alpha-cycle", "beta-cycle", and "gamma-cycle" correspond respectively to the α-Cyclic Iterative Algorithm, β-Reversed Cyclic Algorithm, and γ-Reversed Cyclic Algorithm introduced in Section 5.

In each group of experiments, we present the results of all algorithms under different levels of energy availability. For scheduling performance, we provide box plots of the makespan and median curves of the makespan to enable comprehensive comparison. For energy consumption, we report the total energy usage of each algorithm under different energy constraints to demonstrate that the energy consumption always remains within the allowed budget.

To quantify the available energy budget, we define an energy availability ratio η, which represents the proportion of executable energy relative to the energy consumption obtained by scheduling the application using the HEFT algorithm [16]. The available energy for an application A is expressed as

$$E_{\text{avail}}(A) = \eta \cdot E_{\text{HEFT}}(A), \tag{37}$$

where $E_{\text{HEFT}}(A)$ denotes the energy required when executing application A using the HEFT scheduling strategy. In this work, the value of η ranges from 0.1 to 0.9 with an increment of 0.1, indicating different levels of energy constraints.

The experimental evaluation is conducted on three categories of DAGs: practical application DAGs (such as FFT and GE), randomly generated DAGs, and real-world industrial DAGs. Perturbations are introduced under different energy constraints and task scales to verify the robustness, transferability, and practical applicability of the proposed algorithms. For the first two categories of experiments, each algorithm is executed 50 times under each experimental condition. In the real-world industrial DAG experiments, each algorithm is executed 200 times in each energy interval.

6.1. Practical application DAGs

We first evaluate the proposed scheduling strategy on representative real-world applications. Two widely adopted benchmarks are considered: Gaussian Elimination (GE), characterized by a relatively limited degree of parallelism, and the Fast Fourier Transform (FFT), which typically exhibits high concurrency in its execution structure.

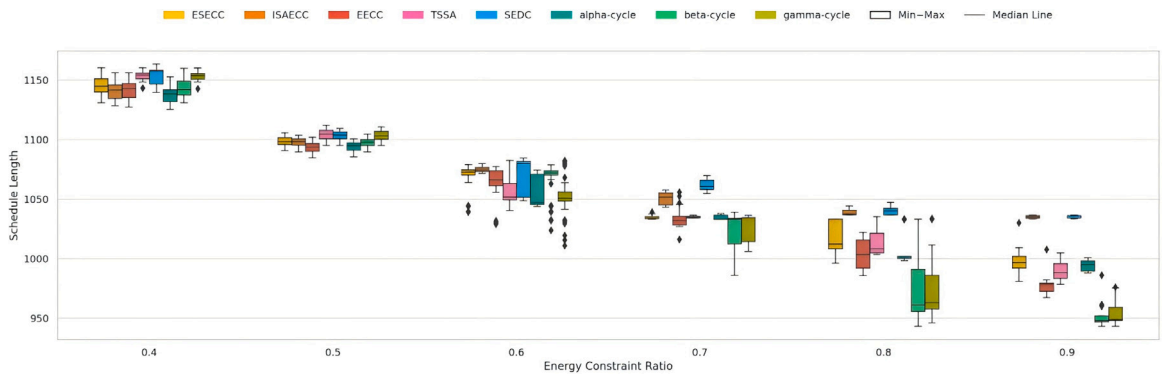


Fig. 15. Boxplots of makespan for the FFT application under different energy availability ratios (Experiment 1).

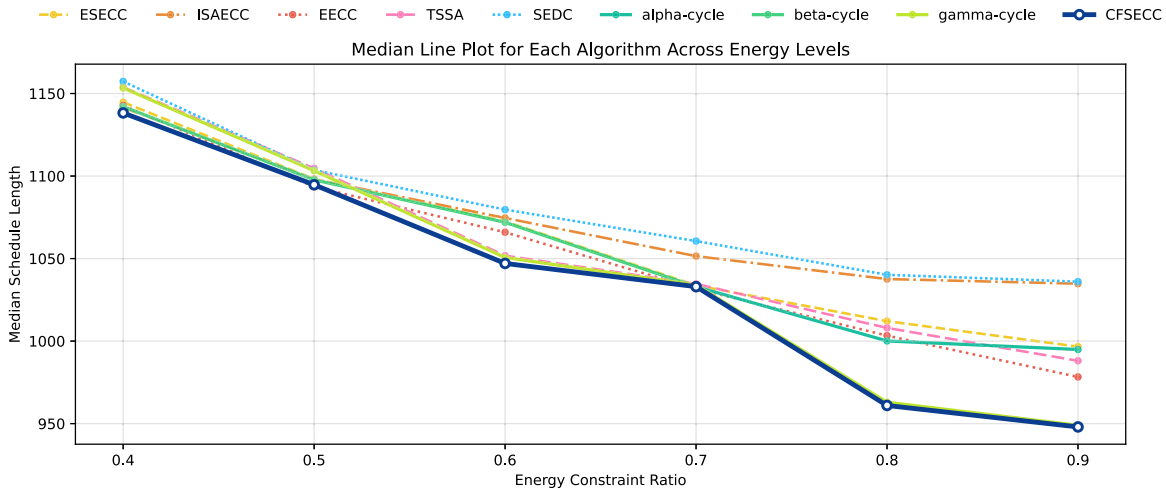


Fig. 16. Median makespan versus energy availability ratio for the FFT application (Experiment 1).

The detailed task structures of these applications have been introduced in Section 3.1, Task Structure.

Experiment 1: FFT — variation of energy availability ratio

In this experiment, the FFT benchmark with a scale parameter of $\rho = 64$ (totaling 511 tasks) is utilized to assess the robustness of the evaluated algorithms under varying levels of energy availability. The energy availability ratio η is adjusted from 0.4 to 0.9 with a step of 0.1, and the comparative makespan results are reported in Fig. 15.

From the box plots in Fig. 15, it can be observed that the makespan of all algorithms decreases as η increases, indicating that relaxing the energy constraint improves scheduling efficiency. CFSECC consistently achieves the shortest makespan, demonstrating superior stability under different η values.

Additionally, Fig. 16 shows that when $\eta = 0.4, 0.5$, and 0.7, the performance gap among the algorithms remains narrow, with CFSECC maintaining a slight advantage. When $\eta = 0.6$, the α-Cyclic Iterative Algorithm provides the most stable and optimal performance, significantly outperforming ESECC, ISAECC, and EECC. When $\eta = 0.8$ and 0.9, the β-Reverse Cyclic Algorithm and γ-Reverse Cyclic Algorithm deliver the most outstanding results, far surpassing traditional scheduling algorithms such as ESECC, ISAECC, EECC, TSSA, and SEDC.

Fig. 17 reveals that all algorithms satisfy the energy constraint. However, ISAECC and SEDC show relatively poor energy utilization efficiency.

Overall, under the best energy-constraint setting, the optimal makespan achieved by CFSECC corresponds to approximately 95.1% of ESECC, 96.9% of EECC, 91.6% of ISAECC, 95.9% of TSSA, and 91.5% of SEDC, demonstrating significant and consistent performance advantages across different energy levels.

Experiment 2: GE — variation of energy availability ratio

Similar to the design of Experiment 1, this experiment uses the GE benchmark with a scale parameter of $\rho = 32$ (totaling 527 tasks) to assess the robustness of the evaluated algorithms under varying levels of energy availability. The energy availability ratio η is adjusted from 0.4 to 0.9 with a step of 0.1, and the comparative makespan results are reported in Fig. 18.

From the median-performance curves in Fig. 19, noticeable performance differences emerge among the scheduling algorithms for the GE application, and these differences become more pronounced as the energy availability ratio η increases. ISAECC and SEDC show a slowing reduction in makespan when η exceeds 0.7, indicating a limited ability to further exploit additional energy resources. Among the proposed algorithms, the α-Cyclic Iterative Algorithm achieves a stable decrease in makespan with increasing η, while the β-Reverse Cyclic Algorithm and γ-Reverse Cyclic Algorithm deliver significantly better performance at $\eta = 0.6$, 0.7, and 0.8, clearly outperforming the traditional scheduling algorithms.

The energy consumption comparison in Fig. 20 shows that all algorithms operate within the allowed energy budget. Except for ISAECC and SEDC, most algorithms nearly fully utilize the available energy resources. Given that the first two experiments have already clarified the relationship between energy usage and available energy for each method, energy histograms will not be provided in subsequent experiments.

Overall, CFSECC also exhibits stable behavior in this experiment, and its average makespan shows clear advantages over the traditional algorithms. The optimal makespan achieved by CFSECC corresponds to approximately 96.6% of ESECC, 95.5% of EECC, 90.2% of ISAECC,

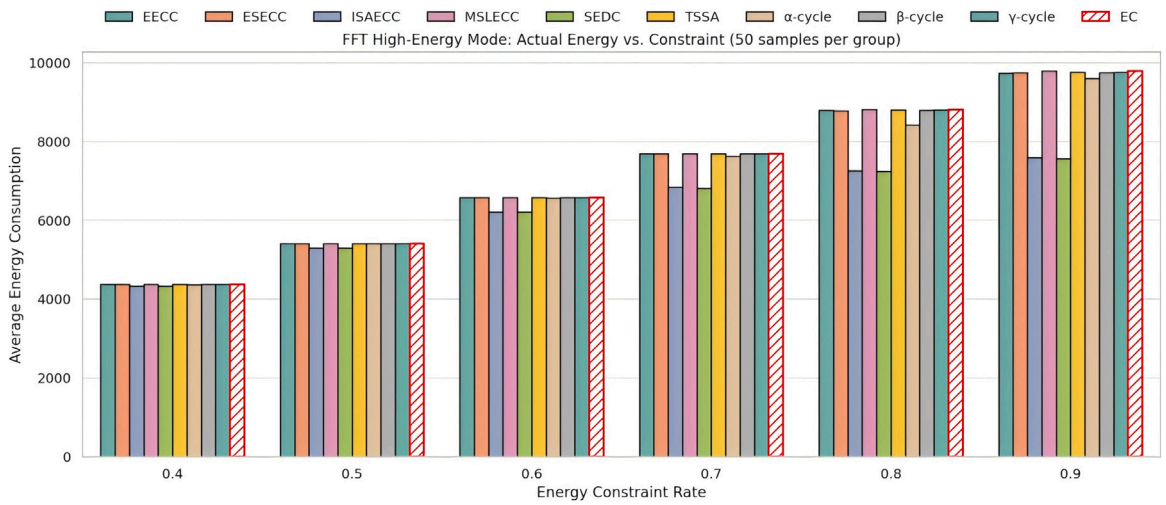


Fig. 17. Energy consumption comparison for the FFT application under different energy availability ratios (Experiment 1).

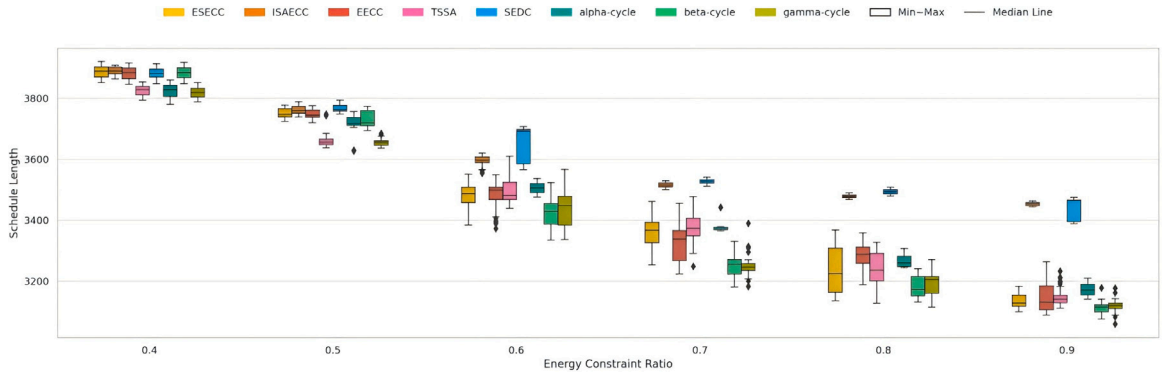


Fig. 18. Boxplots of makespan for the GE application under different energy availability ratios (Experiment 2).

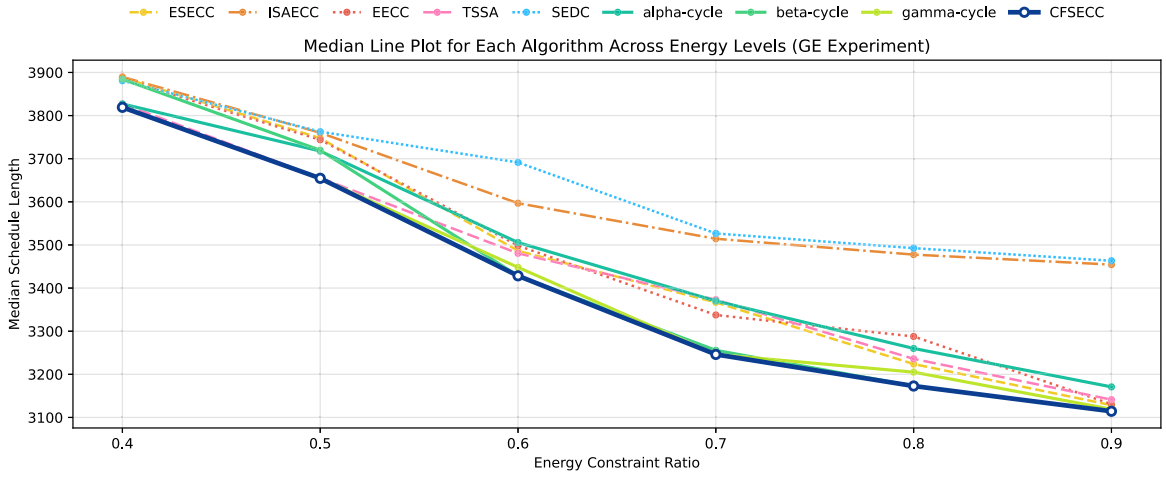


Fig. 19. Median makespan versus energy availability ratio for the GE application (Experiment 2).

97.1% of TSSA, and 89.9% of SEDC, indicating that CFSECC remains competitive with all baseline methods on the GE high-energy task graph and delivers particularly noticeable improvements over ISAECC and SEDC.

Experiment 3: FFT applications with different task quantities

This experiment evaluates the performance of the compared algorithms on FFT applications of varying sizes. The energy availability ratio is fixed at $\eta = 0.8$, while the FFT scale parameter ρ is set to powers of two: $8, 16, 32, 64, 128$, and 256, corresponding to task counts of $39, 95, 223, 511, 1151$, and 2559, respectively. The experimental results for all algorithms are presented in Fig. 21.

From the median-performance curves in Fig. 22, the makespan of all algorithms increases with the task quantity, as expected. Nevertheless, the three cyclic algorithms proposed in this paper maintain clear advantages across all scales. For smaller FFT instances (39 and 95 tasks), the α-Cyclic Iterative Algorithm achieves the shortest makespan. As the number of tasks increases, the β-Reverse Cyclic Algorithm and γ-Reverse Cyclic Algorithm become more competitive. The γ-Reverse Cyclic Algorithm achieves the best performance at 223 tasks,

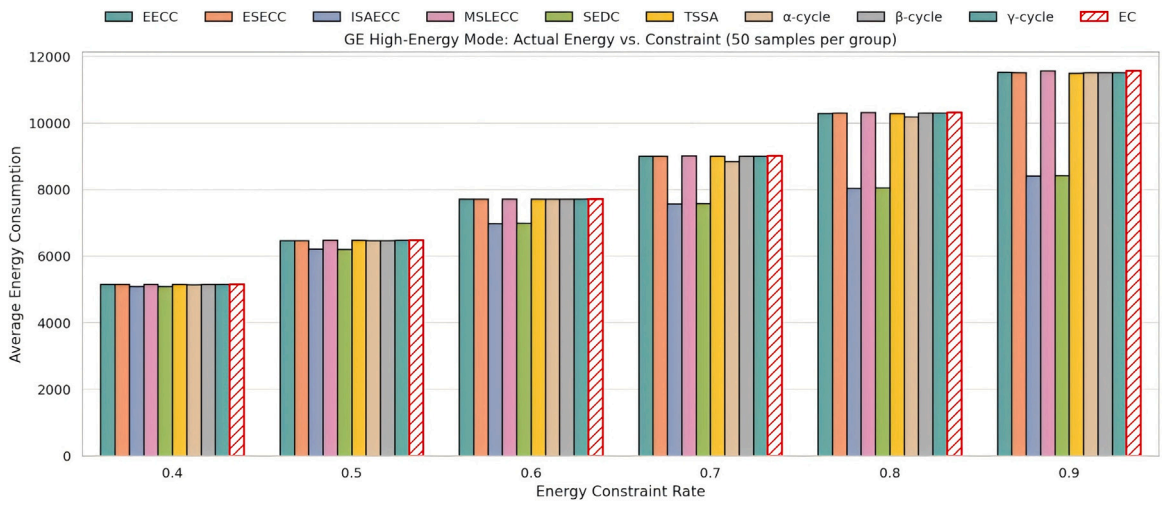


Fig. 20. Energy consumption comparison for the GE application under different energy availability ratios (Experiment 2).

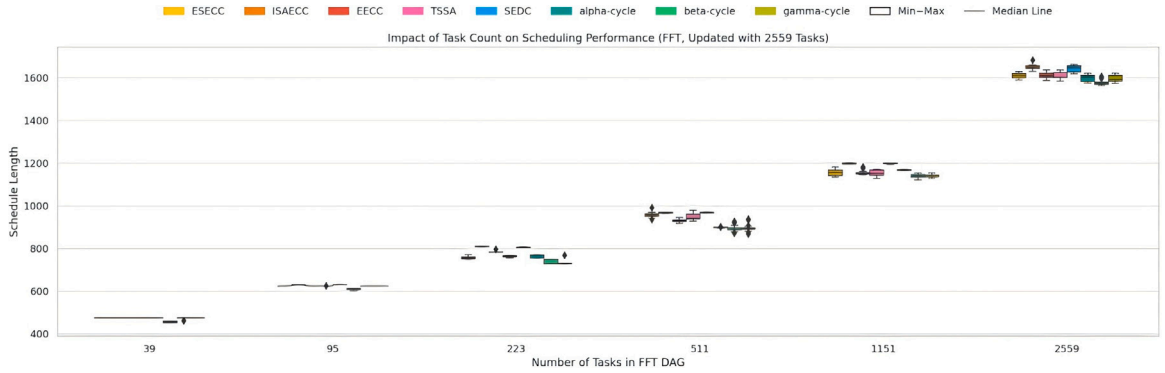


Fig. 21. Boxplots of makespan for FFT applications with different task quantities (Experiment 3).

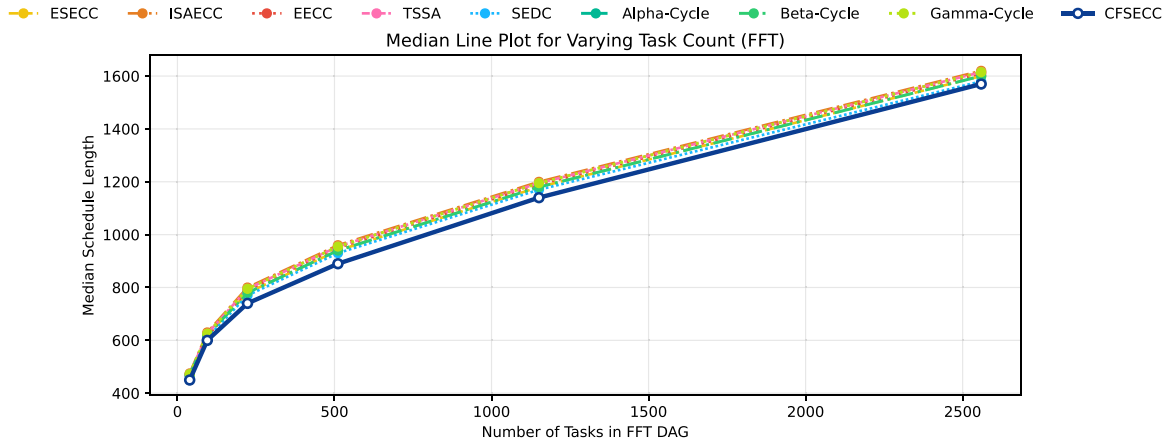


Fig. 22. Median makespan versus task quantity for FFT applications (Experiment 3).

while both reverse-cyclic variants exhibit similar performance at 511 and 1151 tasks. When the task count reaches 2559 and beyond, the β-Reverse Cyclic Algorithm consistently attains the shortest makespan.

In terms of overall average performance, CFSECC outperforms all baseline algorithms across the six FFT task scales. Its average makespan corresponds to approximately 96.5% of ESECC, 96.4% of EECC, 94.2% of ISAECC, 96.5% of TSSA, and 94.3% of SEDC. As the task size increases from a few dozen to several thousand, the advantage of CFSECC remains stable, demonstrating the effectiveness of its cyclic feedback-based energy allocation mechanism even on large-scale DAGs.

Experiment 4: GE applications with different task quantities

This experiment investigates the robustness and scheduling performance of the compared algorithms on GE applications of various scales. The energy availability ratio is fixed at $\eta = 0.7$. To ensure a more comprehensive evaluation under general integer-scale configurations, while satisfying the GE structural constraint that $(\rho^2 + \rho - 2)/2$ must be an integer, the scale parameter ρ is increased by approximately a factor of 2.5, taking values $14, 21, 31, 47$, and 71, corresponding to task quantities of $90, 230, 495, 1127$, and 2555, respectively. The results are shown in Fig. 23.

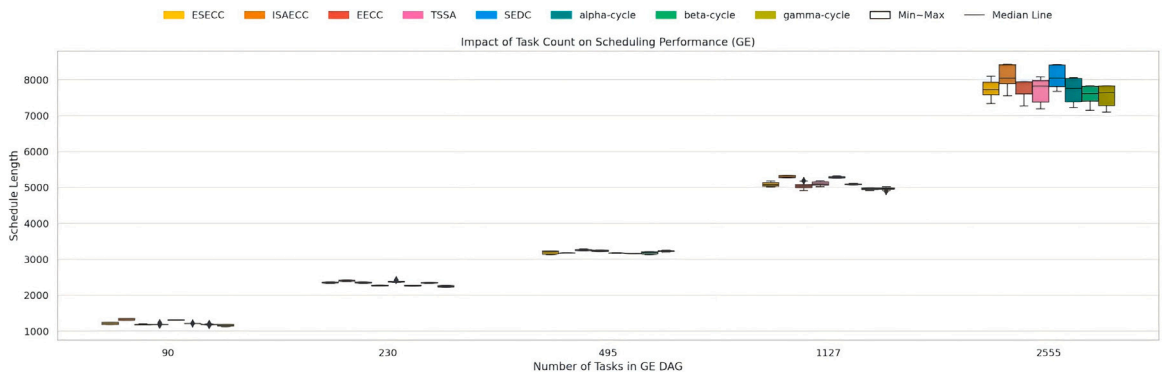


Fig. 23. Boxplots of makespan for GE applications with different task quantities (Experiment 4).

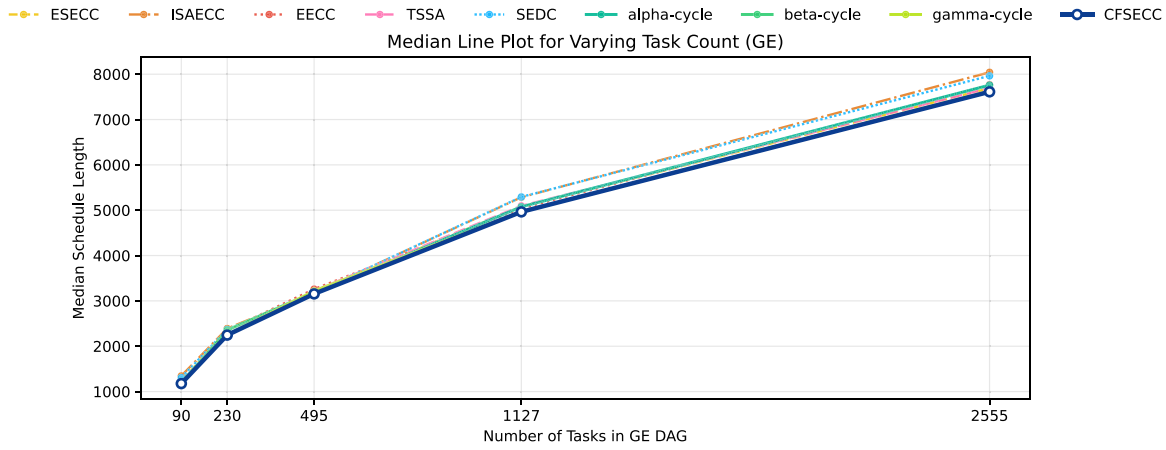


Fig. 24. Median makespan versus task quantity for GE applications (Experiment 4).

From the median-performance curves in Fig. 24, the makespan of all algorithms increases with the number of tasks. The proposed cyclic algorithms consistently maintain advantages across all scales, and their superiority becomes more pronounced as the application size grows. Specifically, the γ-Reverse Cyclic Algorithm achieves the shortest makespan for both small (90 tasks) and large (1127 and 2555 tasks) instances, whereas the α-Cyclic Iterative Algorithm and β-Reverse Cyclic Algorithm outperform the others at intermediate scales such as 495 tasks, illustrating the complementary strengths of the three cyclic variants.

On average, the makespan achieved by CFSECC corresponds to approximately 96.8% of ESECC, 97.1% of EECC, 93.3% of ISAECC, 97.9% of TSSA, and 93.9% of SEDC. These results indicate that the proposed cyclic feedback strategy maintains clear advantages in scheduling efficiency over all baseline algorithms, even for computation-intensive Gaussian Elimination applications with varying task scales.

6.2. Randomly generated DAGs

In the subsequent experiments, we employ randomly generated DAGs to evaluate the performance of the scheduling algorithms on general task graphs. The random DAGs are generated following the ideas in [22,37], where several structural parameters are used to control the overall shape, out-degree distribution, and inter-level dependencies of the graph. Specifically, the shape parameter fat characterizes the "fatness" of the DAG along the level dimension. For a DAG with n tasks, the number of tasks at level l follows a uniform distribution with a mean of $fat \times \sqrt{n}$. The parameter od controls the average out degree of each task, while the parameter $jump$ represents cross-level edges: an edge connecting a task at level l to a task at level $l + jump$ is assigned a $jump$ value equal to its level distance. Similar to [22,37], in our experiments the shape parameter fat is chosen from the set $\{0.6, 0.8, 1.0\}$ and od is selected from $\{1, 2, 3, 4, 5\}$. Approximately 10% of the edges are assigned $jump = 2$ and about 5% of the edges are assigned $jump = 3$, while all remaining edges use $jump = 1$. This random DAG generation method is adopted in Experiments 5 and 6 to investigate the impact of varying processor counts and task-graph complexity on the performance of the algorithms.

Experiment 5: Variation of processor quantity

To further evaluate the scalability and adaptability of the scheduling algorithms under heterogeneous platform resource expansion, this experiment fixes the number of tasks, the DAG structure, and the energy availability ratio, while varying the number of processors in the platform. This setting simulates execution environments ranging from resource-limited to resource-abundant systems.

Specifically, the DAG contains 1000 tasks, the shape parameter is set to $fat = 0.8$, and the energy availability ratio is fixed at $\eta = 0.5$. The number of processors in the platform is configured to $16, 32, 48, 64, 80, 96, 112$, and 128. These configurations allow us to examine how the scheduling performance of different algorithms evolves as the level of available parallelism increases.

The goal of this experiment is to determine whether the algorithms can effectively exploit additional multicore resources and whether the makespan improves as more processors become available, thereby revealing their scalability and resource-utilization efficiency. The comparative results are presented in Fig. 25.

From Fig. 26, it can be observed that the makespan of all algorithms decreases as the number of processors increases. To evaluate the robustness of the algorithms under limited energy conditions, the energy availability ratio is set to a relatively restricted value of $\eta = 0.5$, leaving little surplus energy available. Even under this constrained setting, all

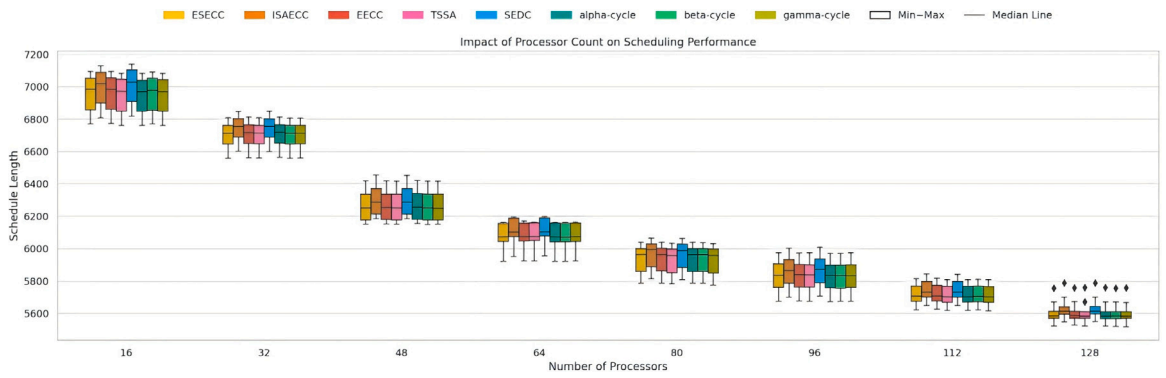


Fig. 25. Boxplots of makespan for different processor counts on random DAGs (Experiment 5).

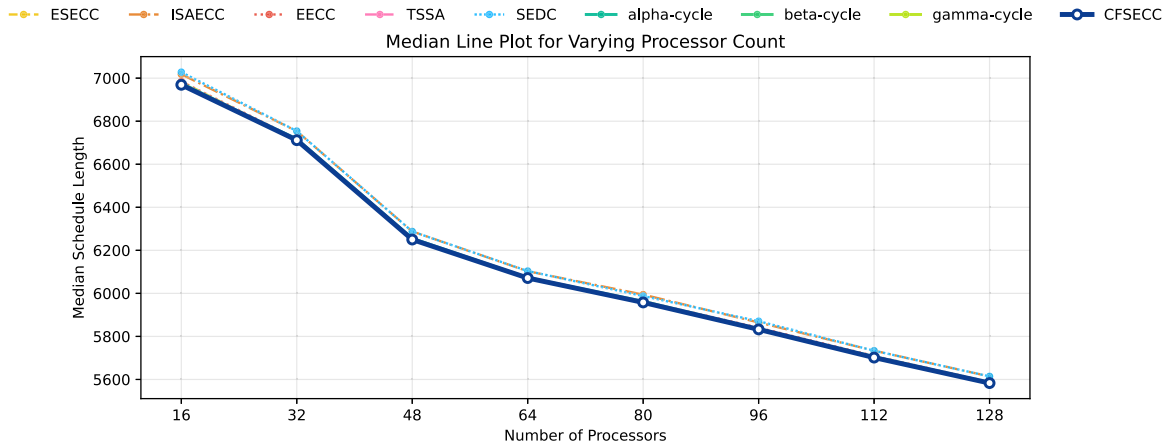


Fig. 26. Median makespan versus processor count for random DAGs (Experiment 5).

algorithms exhibit a steady reduction in makespan as additional parallel resources are introduced. The proposed CFSECC framework consistently maintains shorter schedules, demonstrating strong scalability with respect to processor expansion.

Specifically, the average makespan achieved by CFSECC corresponds to approximately 91.5% of ESECC, 91.5% of EECC, 91.5% of ISAECC, 92.1% of TSSA, and 90.8% of SEDC. These results indicate that, as the number of processors continues to increase, the proposed cyclic feedback strategy is able to maintain scheduling performance comparable to — or even slightly better than — all baseline algorithms, without any degradation attributable to processor-scale expansion.

Experiment 6: Variation of task-graph complexity

To further assess the adaptability of the scheduling algorithms to the structural complexity of task graphs, this experiment fixes the number of tasks, the number of processors, and the energy availability ratio, while varying the shape parameter fat to generate random DAGs with different width–depth characteristics. A smaller value of fat produces deeper DAGs with lower parallelism, whereas a larger value yields wider DAGs with higher degrees of parallelism.

In this experiment, the number of tasks is set to 500, the number of processors is fixed at 32, and the energy availability ratio is set to $\eta = 0.5$. The shape parameter is varied over the set $fat \in \{0.2, 0.4, 0.6, 0.8, 1.0, 1.2\}$. The remaining structural parameters are configured as $od = 3$ and $jump = 3$. These settings produce a variety of DAGs ranging from deep, low-parallelism task graphs to wide, highly parallel structures, enabling a comprehensive evaluation of how algorithm performance responds to increasing task-graph complexity. The results are presented in Fig. 27.

From the median-performance curves in Fig. 28, it can be observed that the structural complexity of the DAG, controlled by the fat parameter, has a pronounced impact on scheduling performance. When fat increases from 0.2 to 0.4, the makespan of all algorithms drops sharply, indicating that the transition from a deep, serial-like structure to a wider DAG significantly enhances available parallelism and improves scheduling efficiency. As fat increases further from 0.4 to 1.2, the makespan gradually rises, suggesting that excessively wide DAGs lead to reduced dependency constraints but higher contention among ready tasks, diminishing the marginal benefits of increased parallelism.

Across the entire range of fat values, the three cyclic variants proposed in CFSECC consistently demonstrate superior performance. The α-Cyclic Iterative Algorithm achieves the shortest makespan near fat = 0.4, where the DAG attains an optimal balance between width and depth. As the DAG becomes wider, the β-Reverse Cyclic Algorithm and γ-Reverse Cyclic Algorithm increasingly outperform the traditional algorithms, maintaining lower makespans at fat = 0.8, 1.0, and 1.2. Overall, CFSECC exhibits robust adaptability to varying DAG structural complexity and consistently maintains stable performance advantages. On average, its makespan corresponds to approximately 99.8% of ESECC, 99.7% of EECC, 99.3% of ISAECC, 99.9% of TSSA, and 99.2% of SEDC, demonstrating strong resilience to changes in graph density and topology.

6.3. Real-world application DAGs

Experiment 7: Real-world industrial DAG

To evaluate the performance of the scheduling algorithms in a practical application scenario, this experiment adopts a real industrial task graph commonly used for benchmarking in embedded control systems [21]. The DAG consists of six subsystems (Fig. 29): the engine controller (tasks C_1–C_7), the automatic gearbox (C_8–C_{11}), the anti-lock braking system (C_{12}–C_{17}), the wheel-angle sensor (C_{18}–C_{19}), the suspension controller (C_{20}–C_{24}), and the bodywork control unit (C_{25}–C_{31}). The task

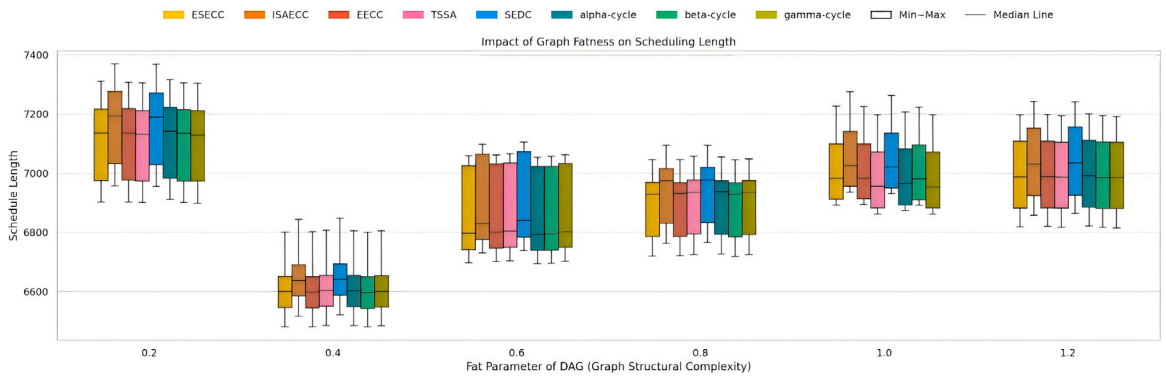


Fig. 27. Boxplots of makespan for different DAG structural complexities (fat parameter) on random DAGs (Experiment 6).

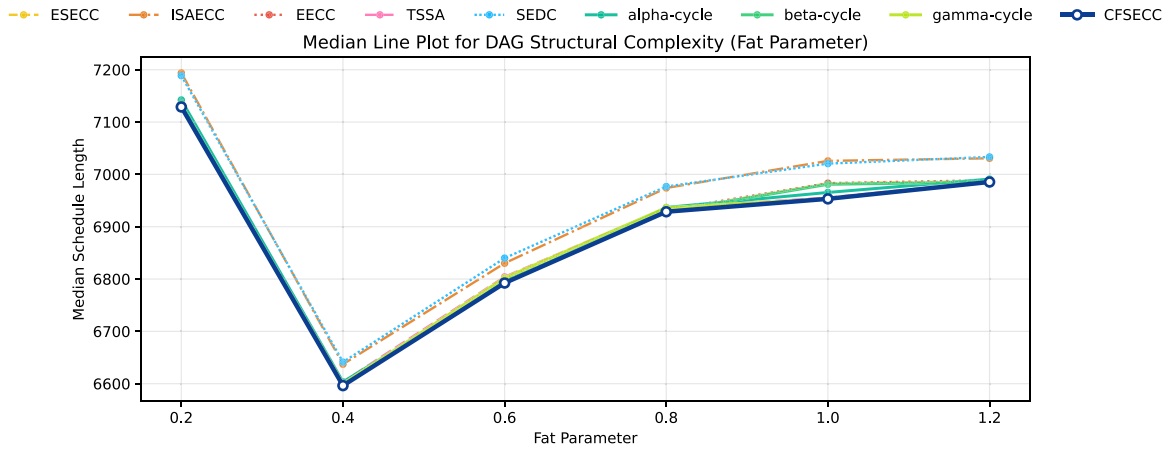


Fig. 28. Median makespan versus fat parameter (DAG structural complexity) on random DAGs (Experiment 6).

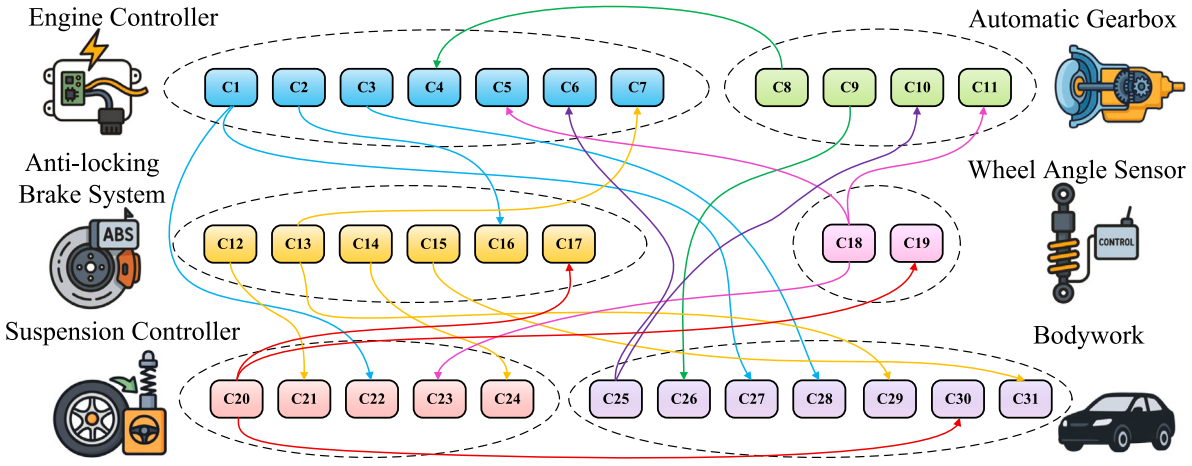


Fig. 29. Task graph of a real-world automotive control application.

parameters follow the configuration in [21]: $100\,\mu s \leq w_{i,k} \leq 400\,\mu s$ and $100\,\mu s \leq c_{i,j} \leq 400\,\mu s$. The number of processors is set to 16.

In this experiment, the energy availability ratio η is varied from 0.4 to 0.9, and the scheduling results of all algorithms are illustrated in Fig. 30.

From Fig. 31, it can be observed that the makespans of all algorithms decrease as the energy availability ratio increases, which aligns with the expected improvement in scheduling flexibility under higher energy budgets. The traditional algorithms ESECC, EECC, and ISAECC exhibit similar overall behaviors, while TSSA and SEDC achieve slightly better results in certain regions but lack consistency across all settings. In contrast, the proposed CFSECC framework consistently yields shorter makespans across all levels of η, demonstrating strong robustness and effective energy utilization.

Notably, as more energy becomes available, the performance advantages of our proposed algorithm become increasingly pronounced, with the improvement margin expanding steadily. When the energy availability ratio reaches 0.9, this advantage becomes most significant: CFSECC reduces the makespan by approximately 12.13%, 18.00%, 12.01%, 6.80%, and 19.31% compared to ESECC, ISAECC, EECC, TSSA, and SEDC, respectively, further confirming its strong optimization capability on real-world industrial applications.

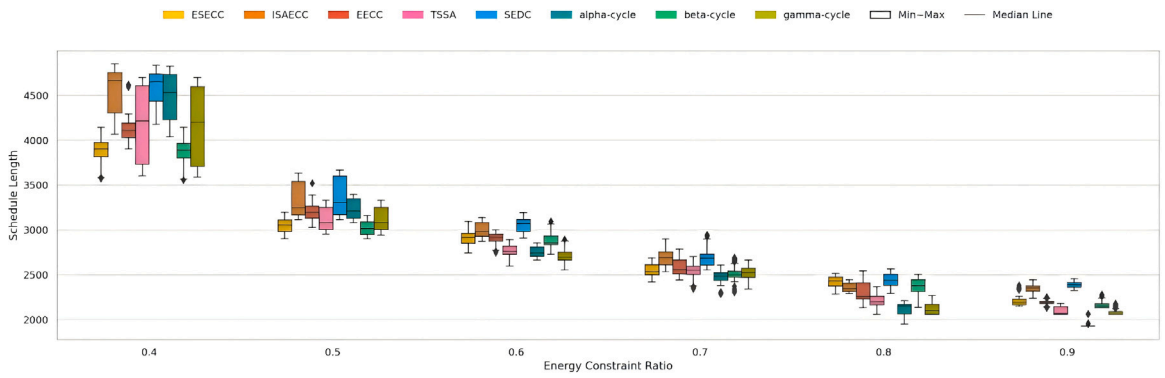


Fig. 30. Boxplots of makespan for the real-world industrial DAG under different energy availability ratios (Experiment 7).

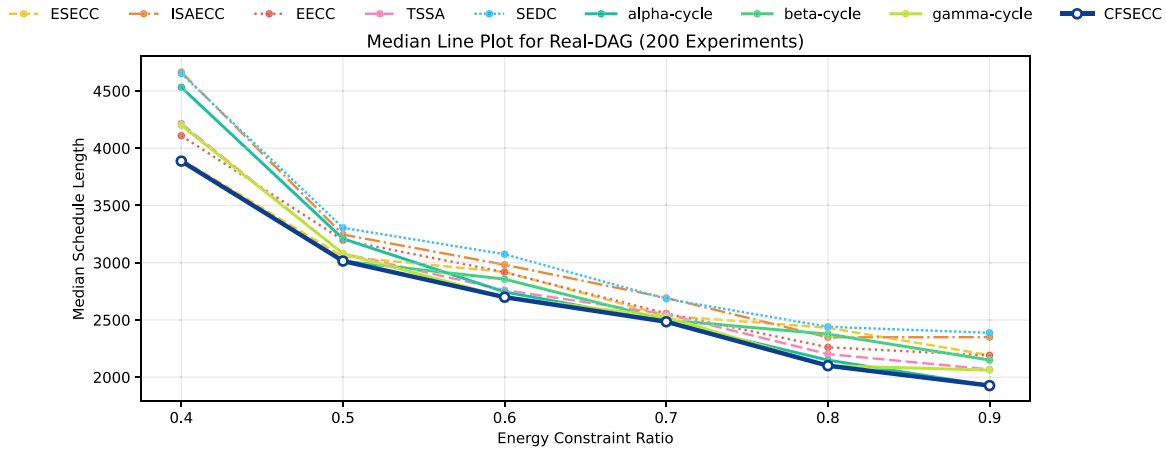


Fig. 31. Median makespan versus energy availability ratio for the real-world industrial DAG (Experiment 7).

Table 11
Winning-frequency distribution of the three cyclic variants across the seven experiment groups. Values are reported as counts with percentages in parentheses.

Experiment	Cases	α wins (%)	β wins (%)	γ wins (%)	Ties (%)
Exp. 1: FFT, energy ratio	300	0 (0.0)	77 (25.7)	220 (73.3)	3 (1.0)
Exp. 2: GE, energy ratio	300	6 (2.0)	172 (57.3)	119 (39.7)	3 (1.0)
Exp. 3: FFT, task number	300	118 (39.3)	104 (34.7)	59 (19.7)	19 (6.3)
Exp. 4: GE, task number	250	46 (18.4)	98 (39.2)	55 (22.0)	51 (20.4)
Exp. 5: Random DAG, processors	400	7 (1.8)	165 (41.2)	228 (57.0)	0 (0.0)
Exp. 6: Random DAG, graph complexity	300	40 (13.3)	15 (5.0)	245 (81.7)	0 (0.0)
Exp. 7: Real-world industrial DAG	1200	311 (25.9)	126 (10.5)	763 (63.6)	0 (0.0)

Overall, the average makespan produced by CFSECC corresponds to 91.5% of ESECC, 91.5% of EECC, 91.5% of ISAECC, 92.1% of TSSA, and 90.8% of SEDC, demonstrating continuous superiority on the real-world industrial DAG.

The analysis of the mechanisms underlying the aforementioned experimental results, together with the corresponding insight summary, is presented as follows.

6.4. Empirical ablation and convergence analysis

To examine the contribution of each cyclic feedback variant, we compare the makespans produced by the α-, β-, and γ-cycle variants in each test case. The variant with the shortest makespan is counted as the winner, and ties are counted separately. Table 11 reports the winning-frequency distribution across the seven experiment groups.

The results show that the three variants have complementary advantages. The γ-variant wins most often in the FFT-energy, processor-number, DAG-complexity, and real-world DAG experiments; the β-variant is stronger in GE-related settings; and the α-variant remains competitive in the FFT-size experiment. Thus, no single variant consistently dominates all conditions, supporting the design of selecting the best schedule from the three cyclic feedback variants.

We further evaluate convergence on the real-world industrial DAG, where the cyclic iteration count is recorded before each variant reaches its corresponding termination condition. This setting provides a practical view of the iterative overhead under six energy-ratio constraints.

Table 12 shows that all three variants terminate within bounded iteration counts. The average iteration counts of α, β, and γ are 14.3, 7.3, and 9.2, respectively, and their worst-case counts are 31, 34, and 48. The β-variant converges fastest on average, while the γ-variant maintains moderate overhead despite reverse-graph exploration. These

Table 12
Convergence statistics on the real-world industrial DAG. Values are reported as average/worst-case iteration counts.

Experiment setting	Cases	α avg./max.	β avg./max.	γ avg./max.
Real-world DAG, $\eta = 0.4$	200	12.1/18	5.5/33	12.2/48
Real-world DAG, $\eta = 0.5$	200	10.3/19	5.8/27	10.4/39
Real-world DAG, $\eta = 0.6$	200	10.2/15	5.0/19	10.5/46
Real-world DAG, $\eta = 0.7$	200	12.5/19	5.0/26	8.3/22
Real-world DAG, $\eta = 0.8$	200	16.4/24	8.6/34	7.3/25
Real-world DAG, $\eta = 0.9$	200	24.3/31	14.1/29	6.6/17
Overall	1200	14.3/31	7.3/34	9.2/48

results indicate that CFSECC improves scheduling performance without excessive iterative overhead in practical scenarios.

6.5. Insight summary

(1) Across all experiments, traditional algorithms such as ESECC, EECC, and ISAECC exhibit highly similar scheduling behaviors. Although their energy-assignment policies differ slightly, they all rely on the same upward-rank priority mechanism, resulting in comparable task-order decisions. Consequently, while individual runs may show minor variation, their average makespans remain close. This indicates that priority-based scheduling without structural adaptation leads to limited performance divergence under varying constraints.

(2) The TSSA algorithm also adopts the upward-rank metric, but performs scheduling on a reversed task graph. This structural inversion sometimes improves path utilization and reduces idle time, allowing TSSA to surpass ESECC, EECC, and ISAECC in certain configurations. However, its advantage is not universal, as the benefit strongly depends on graph shape and task-level depth distribution.

(3) SEDC determines task priority using a time-difference coefficient, resulting in more fine-grained ordering compared with the aforementioned methods. Consequently, SEDC can achieve shorter makespans in scenarios with small task scales or tight energy budgets (e.g., when the task number is 39 in Experiment). However, its residual-energy redistribution mechanism allocates surplus energy primarily to later, lower-priority tasks. When the available energy ratio is high, this leads to insufficient energy utilization by critical tasks, ultimately restricting the algorithm's ability to further optimize makespan.

(4) Across all experimental conditions — including varying energy availability, processor scalability (Experiment 5), and structural complexity of DAGs (Experiment 6) — the proposed CFSECC framework achieves stable and consistently superior performance. The three cyclic variants exhibit complementary strengths: the α-Cyclic Iterative Algorithm is particularly effective under moderate energy budgets due to its balanced exploration–exploitation scheduling pattern, whereas the β-Reverse Cyclic Algorithm and γ-Reverse Cyclic Algorithm dominate in high-energy or high-parallelism settings, where their reverse traversal strategy allows more efficient utilization of execution slack. Compared with traditional methods that fail to fully exploit additional resources, CFSECC demonstrates that cyclic task–energy coordination is essential for robustness and efficiency under diverse system constraints.

6.6. Model limitations and future extensions

The energy model adopted in this work follows the widely used DVFS-based formulation and focuses on processor-frequency-dependent dynamic energy and processor-independent power. This abstraction enables the scheduling algorithm to explicitly control task execution frequencies under a given energy constraint. However, it does not explicitly model inter-core memory contention, cache interference, or shared-bus interference. On real heterogeneous platforms, such shared-resource effects may increase the actual WCET and energy consumption, especially for memory-intensive workloads. Therefore, the reported results should be interpreted as scheduling-level optimization under a standard DVFS-based abstraction. Incorporating contention-aware WCET estimation and shared-resource energy modeling will be considered in future work.

7. Conclusion

To address the energy-constrained scheduling problem in heterogeneous distributed systems, this paper proposes a cyclic feedback scheduling optimization strategy. To fully leverage the potential of multi-round optimization and effectively balance energy allocation and scheduling time, various iterative methods are designed. Multi-round validation experiments demonstrate that, compared to current state-of-the-art research, the proposed method improves energy utilization and shortens scheduling time. Future work will incorporate new application scenarios, such as intelligent connected vehicles, to develop more complex iterative strategies. Methodologically, this will involve integrating intelligent optimization and machine learning techniques to achieve more efficient and refined scheduling improvements. Future work may explore the integration of learning-based strategies with the proposed cyclic feedback mechanism for adaptive scheduling scenarios.

CRediT authorship contribution statement

Yuxi Chen: Writing – original draft, Methodology, Formal analysis, Data curation. **Wufei Wu:** Writing – review & editing, Investigation, Funding acquisition. **Wei Li:** Writing – review & editing. **Jiaxin Zeng:** Software, Formal analysis. **Keqin Li:** Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China under Grant No. 62462045, in part by the Jiangxi province science and technology development program 20244BBC30006, and the China Postdoctoral Science Foundation under Grant No. 2020TQ0134.

Data availability

Data will be made available on request.

References

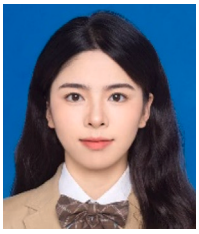
[1] A. Mascitti, T. Cucinotta, M. Marinoni, L. Abeni, Dynamic partitioned scheduling of real-time tasks on ARM big.LITTLE architectures, J. Syst. Softw. 173 (2021).
[2] L. Abeni, G. Lipari, T. Cucinotta, On the scheduling of mixed-criticality real-time tasks on ARM big.LITTLE architectures, Real-Time Syst. 58 (3) (2022) 345–371.
[3] T. Yin, Z. Gao, Z. Xiao, Z. Ma, M. Zheng, C. Zhang, KextFuzz: A practical fuzzer for macos kernel extensions on apple silicon, IEEE Trans. Dependable Secur. Comput. 21 (4) (2024) 3453–3468.
[4] F. Farshchi, Q. Huang, H. Yun, Integrating NVIDIA deep learning accelerator (NVDLA) with RISC-V SoC on FireSim, 2019, CoRR abs/1903.06495.
[5] G. Williams, P. Kanapathipillai, Qualcomm oryon CPU in snapdragon X elite: Micro-architecture and design, IEEE Micro 45 (3) (2025) 8–14.
[6] G. Hattenberger, F. Bonneval, M. Ladeira, E. Grolleau, Y. Ouhammou, Design of micro-drone autopilot architecture with static scheduling optimization, Unmanned Syst. 12 (3) (2023).
[7] G. Fan, Z. Jiang, Hybrid scheduling method for automatic guided vehicles in intelligent warehouses considering power management, Int. J. Adv. Manuf. Technol. 130 (7) (2024) 3685–3695.

[8] P. Song, Q. Yang, G. Wen, Z. Zhang, State feedback periodic event-triggered control for distributed systems of aircraft engines, J. Aerosp. Power 38 (08) (2023) 2015–2023, http://dx.doi.org/10.13224/j.cnki.jasp.20210640.
[9] D. Kim, S. Lee, H. Jung, A time-series model for varying worker ability in heterogeneous distributed computing systems, Appl. Sci. 13 (8) (2023).
[10] D. Sirisha, S.S. Prasad, CPTF–a new heuristic based branch and bound algorithm for workflow scheduling in heterogeneous distributed computing systems, CCF Trans. High Perform. Comput. 6 (5) (2024) 472–487.
[11] J. Jiang, H. Cai, L. Xie, Z. Sun, L. Pan, Z. Yang, R. Lu, A time and reliability optimization algorithm for workflow scheduling in heterogeneous distributed computing system, J. Circuits Syst. Comput. 31 (14) (2022).
[12] H. Fujiwara, H. Mori, et al., A 5-nm 254-TOPS/W 221-TOPS/mm2 computing-in-memory macro supporting DVFS and simultaneous MAC operations, in: 2022 IEEE International Solid-State Circuits Conference, ISSCC, 2022.
[13] E. Le Sueur, G. Heiser, Dynamic voltage and frequency scaling: The laws of diminishing returns, in: Proceedings of the 2010 International Conference on Power Aware Computing and Systems, 2010, pp. 1–8.
[14] J. Zidar, T. Matić, I. Aleksić, Ž. Hocenski, Dynamic voltage and frequency scaling for reducing energy in ultra-low-power embedded systems, Electronics 13 (5) (2024).
[15] J. Li, C. Jia, L. Cheng, Q. Zhao, Active suspension state feedback H_∞ control for electric vehicles on impulse pavement, J. Hunan Univ. (Natural Sciences) 49 (8) (2022) 12–20.
[16] H. Topcuoglu, S. Hariri, M.-Y. Wu, Performance-effective and low-complexity task scheduling for heterogeneous computing, IEEE Trans. Parallel Distrib. Syst. 13 (3) (2002) 260–274.
[17] X. Xiao, G. Xie, R. Li, K. Li, Minimizing schedule length of energy consumption constrained parallel applications on heterogeneous distributed systems, in: 2016 IEEE Trustcom/BigDataSE/ISPA, IEEE, 2016, pp. 1471–1476.
[18] J. Song, G. Xie, R. Li, X. Chen, An efficient scheduling algorithm for energy consumption constrained parallel applications on heterogeneous distributed systems, in: 2017 IEEE International Symposium on Parallel and Distributed rocessing with Applications, ISPA, IEEE, 2017, pp. 32–39.
[19] J. Li, G. Xie, K. Li, Z. Tang, Enhanced parallel application scheduling algorithm with energy consumption constraint in heterogeneous distributed systems, J. Circuits Syst. Comput. 28 (11) (2019) 1950190.
[20] T. Ye, Z.-J. Wang, Z. Quan, S. Guo, K. Li, K. Li, ISAECC: An improved scheduling approach for energy consumption constrained parallel applications on heterogeneous distributed systems, in: 2018 IEEE 24th International Conference on Parallel and Distributed Systems, ICPADS, IEEE, 2018, pp. 267–274.
[21] W. Zhu, W. Wu, X. Yang, G. Zeng, TSSA: Task structure-aware scheduling of energy-constrained parallel applications on heterogeneous distributed embedded platforms, J. Syst. Archit. 132 (2022) 102741.
[22] J. Chen, P. Han, Y. Zhang, T. You, P. Zheng, Scheduling energy consumption-constrained workflows in heterogeneous multi-processor embedded systems, J. Syst. Archit. 142 (2023) 102938.
[23] P. Khaledian, S.B. Arami, A. Khonsari, An energy-efficient and deadline-aware workflow scheduling mechanism for fog–cloud computing environments, J. Netw. Comput. Appl. 212 (2023) 103556, http://dx.doi.org/10.1016/j.jnca.2023.103556.
[24] M. Ijaz, G. Abbas, M. Fatima, et al., An energy and makespan efficient task scheduling algorithm in fog–cloud environments, Comput. Electr. Eng. 93 (2021) 107265, http://dx.doi.org/10.1016/j.compeleceng.2021.107265.
[25] H. Li, K. Li, K. Cai, et al., An energy-aware scheduling method for multiple scientific workflows in heterogeneous hybrid clouds, Concurr. Comput.: Pr. Exp. 34 (20) (2022) e7044, http://dx.doi.org/10.1002/cpe.7044.
[26] A. Jayanetti, S. Perera, W.M.T.D. Ranasinghe, et al., Energy and time-aware task scheduling in edge–cloud environments using deep reinforcement learning, J. Syst. Archit. 127 (2022) 102481, http://dx.doi.org/10.1016/j.sysarc.2022.102481.
[27] X. Hou, J. Zhou, L. Li, P. Cong, Z. Wu, M. Chen, Latency and reliability-aware dynamic task offloading and scheduling for energy-harvesting systems in mobile edge computing, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. (2025).
[28] J. Ke, J. Zhou, D. Meng, Y. Zeng, Y. Shi, X. Qu, S. Guo, JCSRC: Joint client selection and resource configuration for energy-efficient multi-task federated learning, IEEE Trans. Comput. (2025).
[29] Y. Zeng, J. Zhou, B. Ye, Z. Qu, S. Guo, T. Gong, P. Li, ExpertDRL: Request dispatching and instance configuration for serverless edge inference with foundation models, IEEE Trans. Mob. Comput. (2025).
[30] R. Jia, K. Zhao, X. Wei, et al., Joint trajectory planning, service function deploying, and DAG task scheduling in UAV-empowered edge computing, Drones 7 (7) (2023).
[31] W. Deng, L. Zhu, Y. Shen, et al., A multi-objective optimized DAG task scheduling strategy for fog computing, Wirel. Netw. (2024).
[32] W. Yang, Y. Mao, X. Chen, et al., Trajectory planning and DAG task scheduling in multi-UAV assisted edge computing, Ad Hoc Networks 166 (2025) 103668.
[33] G. Xie, J. Jiang, Y. Liu, R. Li, K. Li, Minimizing energy consumption of real-time parallel applications using downward and upward approaches on heterogeneous systems, IEEE Trans. Ind. Inform. 13 (3) (2017) 1068–1078.
[34] G. Xie, Y. Chen, X. Xiao, C. Xu, R. Li, K. Li, Energy-efficient fault-tolerant scheduling of reliable parallel applications on heterogeneous distributed embedded systems, IEEE Trans. Sustain. Comput. 3 (3) (2017) 167–181.
[35] G. Xie, Y. Chen, Y. Liu, Y. Wei, R. Li, K. Li, Resource consumption cost minimization of reliable parallel applications on heterogeneous embedded systems, IEEE Trans. Ind. Inform. 13 (4) (2016) 1629–1640.
[36] G. Xie, G. Zeng, X. Xiao, R. Li, K. Li, Energy-efficient scheduling algorithms for real-time parallel applications on heterogeneous distributed embedded systems, IEEE Trans. Parallel Distrib. Syst. 28 (12) (2017) 3426–3442.
[37] H. Arabnejad, J.G. Barbosa, List scheduling algorithm for heterogeneous systems by an optimistic cost table, IEEE Trans. Parallel Distrib. Syst. 25 (3) (2014) 682–694.

Yuxi Chen: currently he is working towards a bachelor's degree at Nanchang University. His research interests are primarily centered on system-on-chip and embedded distributed systems, specifically aimed at enhancing energy efficiency and optimizing scheduling duration for task management.

Wufei Wu: h.D. degree in College of Computer Science and Electronic Engineering from Hunan University, Changsha, China, in 2018. He was a Visiting Scholar at Nagoya University, Nagoya, Japan, from 2016 to 2017. He is currently an associate professor at the School of Information Engineering, Nanchang University. He has worked on scheduling optimization for distributed systems, real-time embedded systems, and cybersecurity enhancements for in-vehicle networks. He is a senior member of IEEE and CCF.

Wei Li: received the Ph.D. degree from Huazhong University of Science and Technology (HUST), China, in 2020, and the B.S. degree from Wuhan University of Technology (WHUT), in 2015. She was a Research Assistant at the Chinese University of Hong Kong, Shenzhen, from 2019 to 2020. Currently, she is a lecturer at the School of Information Engineering, Nanchang University. Her current research interests include distributed resource allocation, causal inference, and applications of artificial intelligence.



Jiaxin Zeng: she received her B.A. degree from the School of Foreign Languages of Jiangxi Agricultural University, China, in 2024. She is pursuing a Master's degree at the School of Information Engineering, Nanchang University. Her research interest is in high-performance heterogeneous computing and edging computing.

Keqin Li: received a B.S. degree in computer science from Tsinghua University in 1985 and a Ph.D. degree in computer science from the University of Houston in 1990. He is a SUNY Distinguished Professor with the State University of New York and a National Distinguished Professor with Hunan University (China). He has authored or co-authored more than 1060 journal articles, book chapters, and refereed conference papers. He received several best paper awards from international conferences including PDPTA-1996, NAECON-1997, IPDPS-2000, ISPA-2016, NPC-2019, ISPA-2019, and CPSCom-2022. He holds over 75 patents announced or authorized by the Chinese National Intellectual Property Administration. Since 2020, he has been among the world's top few most influential scientists in parallel and distributed computing regarding single-year impact (ranked #2) and career-long impact (ranked #4) based on a composite indicator of the Scopus citation database. He was a 2017 recipient of the Albert Nelson Marquis Lifetime Achievement

Award for being listed in Marquis Who's Who in Science and Engineering, Who's Who in America, Who's Who in the World, and Who's Who in American Education for over twenty consecutive years. He received the Distinguished Alumnus Award from the Computer Science Department at the University of Houston in 2018. He received the IEEE TCCLD Research Impact Award from the IEEE CS Technical Committee on Cloud Computing in 2022 and the IEEE TCSVC Research Innovation Award from the IEEE CS Technical Community on Services Computing in 2023. He won the IEEE Region 1 Technological Innovation Award (Academic) in 2023. He was a recipient of the 2022–2023 International Science and Technology Cooperation Award of Hunan Province, China. He is a Member of the SUNY Distinguished Academy. He is an AAAS Fellow, an IEEE Fellow, an AILA Fellow, an ACIS Fellow, and an AIAA Fellow. He is a Member of Academia Europaea and an Academician of the Academy of Europe.