Future Generation Computer Systems 166 (2025) 107678

Contents lists available at ScienceDirect

Future Generation Computer Systems

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

Scheduling energy-constrained parallel applications in heterogeneous systems

Hongzhi Xu [a,*], Binlian Zhang [a], Chen Pan [b], Keqin Li [c]

[a] College of Computer Science and Engineering, Jishou University, Zhangjiajie 427000, China
[b] Department of Electrical & Computer Engineering, The University of Texas at San Antonio (UTSA), San Antonio, TX 78249, USA
[c] Department of Computer Science, State University of New York, New Paltz, NY 12561, USA

ARTICLE INFO

Keywords:
Dynamic voltage and frequency scaling
Energy constraint
Makespan
Parallel application

ABSTRACT

With the rapid development of information technology, efficient energy utilization has become a major challenge in modern computing system design. This paper focuses on the energy-constrained parallel application scheduling problem in heterogeneous systems and proposes three algorithms to minimize the makespan of applications. The first one is the minimum makespan algorithm under energy constraints. In this algorithm, we construct an optimal cost table with energy constraints, which can be applied to determine the priority of tasks and the processors allocated in the application. The second one is the energy reclaiming algorithm, which is used to reclaim some energy from non-critical tasks while ensuring that the makespan of the application remains unchanged. The third one is the energy reallocation algorithm, which tends to allocate reclaimed energy to critical tasks to increase their execution frequency, thereby reducing the makespan of the entire application. Experiments were conducted on different parallel applications in various scenarios, and the results showed that the proposed algorithm can achieve smaller makespan compared to existing algorithms in most cases.

1. Introduction

With the widespread application of artificial intelligence, big data analysis, high-definition video processing, and other domains, the demands on computing systems have escalated. Traditional single-type processor systems often fall short in terms of efficiency and performance when handling these high-load, high-complexity tasks. Heterogeneous computing systems, by integrating diverse specialized computing units such as Central Processing Units (CPUs), Graphics Processing Units (GPUs), and Field-Programmable Gate Arrays (FPGAs), can more effectively handle complex and varied computational scenarios [1,2]. When executing parallel applications on mobile devices, Internet of Things (IoT) devices, and high-performance heterogeneous computing platforms, energy efficiency is crucial [3]. It is imperative to minimize system energy consumption while meeting specific user demands, such as service quality and response times. Reducing energy consumption not only lowers electricity costs during operation but also reduces carbon emissions.

Dynamic Voltage and Frequency Scaling (DVFS) stands as an effective technique for enhancing energy efficiency by dynamically adjusting the processor's voltage and execution frequency based on the actual requirements of applications. This approach can significantly reduce system energy consumption when the system operates under low workload conditions and can be widely applied in various fields [4–8]. However, the reduction in processor voltage and frequency through DVFS inevitably extends the execution time of applications, which might result in applications failing to meet their execution time constraints [9]. Consequently, in scenarios where DVFS is applied, it is vital to strike a balance between performance and energy conservation, ensuring that applications run efficiently without compromising the user experience.

In practical application scenarios, heterogeneous embedded systems inherently limited energy resources, such as those powered by batteries or reliant on energy harvesting mechanisms. When executing applications within these systems, meticulous management of energy consumption is crucial. In energy-constrained heterogeneous multiprocessor systems, task scheduling for parallel applications faces several challenges: firstly, how to appropriately assign tasks to suitable processors to balance computational performance and energy consumption, ensuring that the available energy within the system is not exhausted prematurely; secondly, how to dynamically adjust the voltage and operating frequency of processors based on system workload,

* Corresponding author.
E-mail addresses: xuhongzhi9@163.com (H. Xu), zhangbinlian@163.com (B. Zhang), chen.pan@utsa.edu (C. Pan), lik@newpaltz.edu (K. Li).

https://doi.org/10.1016/j.future.2024.107678
Received 30 July 2024; Received in revised form 14 November 2024; Accepted 14 December 2024
Available online 21 December 2024

thereby guaranteeing that applications meet their execution time constraints. Therefore, it is necessary to investigate the parallel application scheduling problem with energy constraints in heterogeneous systems.

In recent years, numerous researchers have proposed energy-efficient scheduling approaches that aim to reduce energy consumption while still satisfying the execution time constraints of applications [10–13]. However, these approaches do not consider energy as a constraint in task scheduling, rendering them unsuitable for scenarios where energy resources are strictly bounded. For energy-constrained systems, there are some algorithms that optimize the execution time (also known as makespan) of parallel applications [14–19]. However, most of them use the execution time required when the processor operates at its highest frequency to calculate the priority of tasks. When tasks have energy constraints, they may not be able to execute at the highest frequency, which often results in unreasonable task prioritization. Moreover, existing algorithms only evaluate the current task based on available energy when assigning tasks to processors, without considering the future situation of the current task, which may lead to suboptimal decisions in some cases. In addition, existing algorithms tend to allocate disproportionately high amounts of energy to non-critical subtasks in parallel applications, whose execution fails to contribute positively to reducing the overall application runtime. To address these shortcomings, we have designed a series of new algorithms. In most cases, our algorithms achieve shorter execution times for applications compared to the existing algorithms.

This paper primarily investigates the scheduling problem of parallel applications with energy constraints in heterogeneous systems, aiming to minimize the execution time of the applications. Our main contributions are outlined as follows:

(1) This paper constructs an energy-constrained Optimistic Cost Table (OCT), based on which the priority of tasks within the application is determined. Since the values in the OCT are calculated according to the energy budget of the application, the priority of tasks for the same application may vary depending on the available energy, which facilitates reducing the makespan of the application.

(2) This paper designs three algorithms. The first algorithm aims to minimize the makespan of the application under energy constraints. Since there are non-critical tasks in the results of this algorithm, the execution frequencies of these tasks can be further reduced. Therefore, the second algorithm is designed to reclaim energy from these non-critical tasks while ensuring the application's makespan remains unchanged. The third algorithm then employs this reclaimed energy to readjust the execution frequencies of tasks, thereby further reducing the makespan of applications.

(3) This paper conducts experiments using different applications across various scenarios. The experimental results illustrate that the makespan generated by the algorithm proposed in this paper is smaller than that generated by existing algorithms in most cases.

The rest of this paper is organized as follows. Section 2 reviews the related work. Section 3 introduces the models used in this paper and discusses the problems under investigation. Section 4 elaborates on the three algorithms we proposed and provides a case study. Finally, Sections 5 and 6 discuss our simulation results and conclusions.

2. Related work

Parallel applications are commonly modeled as Directed Acyclic Graphs (DAGs) due to the precedence constraints existing between their subtasks [10–13]. Scheduling parallel applications in heterogeneous systems to minimize execution time is known to be an NP-complete problem. The problem of scheduling parallel applications to optimize metrics such as execution time, energy consumption, reliability, and throughput has been widely investigated [12,20–24].

To reduce system energy consumption while meeting the execution time constraints of applications, Tang et al. [10] introduced the DEWTS algorithm, which utilizes DVFS and the method of processors merging to reduce system energy consumption while ensuring that applications meet their deadline requirements. In [11], Xie et al. employed the downward and upward methods to reduce system energy consumption. In [12], the authors separately developed non-DVFS and global DVFS-enabled scheduling algorithms to reduce system energy consumption. In [13], Huang et al. proposed the DVFS-weakly-dependent scheduling algorithm to minimize energy consumption. In [25], the authors introduced a deadline-constrained energy-aware scheduling approach for geographically distributed cloud data centers.

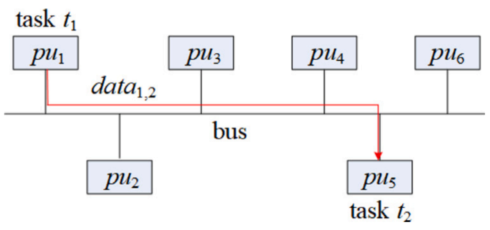


Fig. 1. Structure of heterogeneous systems.

In response to the varying user requirements and diverse execution environments of applications, researchers have developed scheduling algorithms subject to other constraints, such as minimizing energy consumption with reliability constraints [26–30], minimizing energy consumption with execution time and reliability constraints [31,32], maximizing reliability with execution time constraints [33,34], minimizing execution cost with budget constraints [35], and enhancing user quality of experience while minimizing energy consumption [36].

For scenarios with energy constraints, Xiao et al. [14] proposed the MSLECC algorithm to minimize the scheduling length of applications. Due to the imbalance of energy pre-allocation among tasks in the MSLECC algorithm, in [15], Song et al. designed the ESECC algorithm, which improves the pre-allocation energy method in the MSLECC algorithm. In [16], the authors introduced an enhanced scheduling algorithm to reduce the scheduling length of parallel applications. Furthermore, Quan et al. [17] also proposed an improved scheduling approach for energy consumption constrained parallel applications to reduce the scheduling length. In [18], Zhu et al. investigated a structure-aware task scheduling method, which takes into account the structure of tasks during scheduling. In [19], Chen et al. proposed a scheduling approach based on the energy difference coefficient to minimize the execution time of applications. In [37], Li et al. proposed a minimal schedule time with energy constraint method in the fog computing environment. In [38], the authors introduced a hierarchical computation offloading technique that maintains energy and delay constraints in the fog environment. In addition, there are also methods for network optimization and resource management, such as SASRM [39], WS-QSTM [40], and MIJTIP [41].

This paper primarily focuses on minimizing the execution time of energy-constrained parallel applications in heterogeneous systems. Table 1 lists the works that share the same objectives as this paper.

3. System model and problem description

3.1. Heterogeneous system model

The heterogeneous system utilized in this study consists of m processor units, denoted as $PU = \{pu_1, pu_2, \dots, pu_m\}$, with each processor unit equipped with DVFS capability. The platform architecture is shown in Fig. 1, where the processor units can collaborate in executing parallel applications, and the output from any given processor can be transmitted via a bus to other processors for further processing. Additionally, each task exclusively uses a processor during its execution, meaning that the execution of any task is considered non-preemptive [26].

Table 1
Works with the same research objectives.

Reference	Algorithm	Strengths	Weaknesses
[14]	MSLECC (Minimum Schedule Length with Energy Consumption Constraint)	The proposed algorithm can always satisfy the energy consumption constraint, and its correctness is verified using proof and experiments.	The large disparity in energy allocation to tasks affects the performance of the algorithm.
[15]	ESECC (Efficient Scheduling for Energy Consumption Constrained)	Proposed a method for averaging the allocation of available energy to tasks to meet energy constraints.	The inherent properties of the tasks were not considered during the energy pre-allocation.
[16]	EECC (Enhanced scheduling with Energy Consumption Constraint)	Proposed a new energy pre-allocation method and proved its correctness.	The inherent properties of the tasks were still not considered during the energy pre-allocation.
[17]	ISAECC (Improved Scheduling Approach for Energy Consumption Constrained)	Proposed a weight-based energy pre-allocation strategy, where the weights are calculated considering the maximum and minimum energy requirements of the tasks.	The energy consumption for executing critical and non-critical tasks was not further processed.
[18]	TSSA (Task Structure-aware Scheduling Algorithm)	During energy pre-allocation, the average execution time of tasks on different processors was considered. The scheduling algorithm takes into account the structure of the task graph.	Need to perform both forward and reverse direction scheduling on the task graph.
[19]	SEDC (Scheduling algorithm based on the Energy Difference Coefficient)	Proposed a new scheduling algorithm based on the energy difference coefficient, which prioritizes tasks and allocates energy constraints to them.	The given energy budget for the application cannot be fully utilized in some cases.
[37]	MSTEC (Minimal Schedule Time with Energy Constraint)	Taking into account the inherent properties of the tasks and introducing the concept of the energy consumption ratio during energy pre-allocation. The algorithm uses task execution time prediction to optimize scheduling.	When calculating the priority of tasks, energy constraints were not considered.

3.2. Application model

In this study, we represent a parallel application as a DAG $A = (T, E)$ [2,26,27,42]. The components T and E are explained as follows:

T is the task set, denoted as $T = \{t_1, t_2, \ldots, t_n\}$, indicates that there are n tasks within application A. E is the set of directed edges between tasks, where each edge $e(i,j) \in E$ represents a dependency from task t_i to task t_j. If there is an edge from t_i to t_j, it means that the execution of task t_j must wait for task t_i to complete, i.e., t_i is a predecessor of t_j. For the convenience of representing the relationships between tasks, we define $pred(t_i)$ to represent the set of all direct predecessor tasks of task t_i, and $succ(t_i)$ to represent the set of all direct successor tasks of task t_i. Tasks without any direct predecessor and tasks without any direct successor are recorded as t_{entry} and t_{exit}, respectively.

Due to the heterogeneity of processor units in the system, the time required to execute the same task may differ across different processors. Define an $n \times m$ matrix W to represent the Worst-Case Execution Time (WCET) of each task on each processor unit. The notation $w_{i,k}$ signifies the WCET of task t_i when executed on processor unit pu_k. Since tasks are non-preemptive and the WCET is considered, we do not consider context switching overheads.

The communication between tasks mapped to different processors is performed through data transfer over the bus. If there is an edge $e(i,j)$, then task t_i (assigned to processor pu_u) needs to transmit data to task t_j (assigned to processor pu_v), and the time required for this data transmission can be calculated as [42,43]

$$dtt_{i,j} = L_u + \frac{data_{i,j}}{B_{u,v}}, \tag{1}$$

where L_u is the communication startup latency of processor pu_u, $data_{i,j}$ represents the size of the data to be transmitted, and $B_{u,v}$ represents the bandwidth of the link connecting processor pu_u and processor pu_v.

To ensure the robustness of the system, we adopt the Worst-Case Response Time (WCRT) for data transmission. The WCRT represents the maximum possible actual response time when transmitting data on a specific hardware platform [26]. Accordingly, the maximum of all possible actual response times $dtt_{i,j}$ is denoted by $c_{i,j}$, i.e., $c_{i,j}$ represents the WCRT for t_i to transmit data to t_j when t_i and t_j are not assigned to the same processor. When t_i and t_j are assigned to the same processor, the communication time is considered to be zero, because it is negligible compared to the inter-processor communication time [26]. In this study, WCRTs for all data transmissions are known and determined during the analysis phase using the WCRT analysis method [26].

3.3. Energy consumption model

This study adopts the widely used system-level power model [11,12,14,44]. The power dissipation of the processor when performing tasks at frequency f is given as

$$P(f) = P_{\text{st}} + \beta(P_{\text{in}} + P_{\text{de}}) = P_{\text{st}} + \beta(P_{\text{in}} + C_{\text{sw}} f^{\alpha}). \tag{2}$$

In (2), P_{st} represents static power dissipation, which is typically used to maintain the basic circuits and clock operation. This power persists continuously and can only be eliminated by shutting down the entire system. P_{in} denotes the dynamic power dissipation that is independent of the processor speed, and it is typically a constant. P_{de} represents the frequency-dependent dynamic power dissipation, which includes the power primarily consumed by the processor and any power that is dependent on the system's processing frequency. C_{sw} is the effective switching capacitance, and α is the dynamic power exponent, a system-related constant that is generally greater than 2 and less than 3. When the processor is in an operational state, $\beta = 1$; otherwise, $\beta = 0$. Considering that static energy consumption is unavoidably incurred, similar to research studies sharing the same objective as ours [14–19], we primarily concentrate on the dynamic energy consumption of processors.

Reducing the execution frequency leads to an extended task execution time, which in turn increases the frequency-independent energy consumption. If the execution frequency is lowered without limit, the overall system energy consumption may actually increase rather than decrease, because even though dynamic power consumption is reduced, the increase in frequency-independent energy consumption due to longer execution times might exceed the savings. Therefore, there exists a minimum energy-efficient frequency, which can be calculated as [11,12,14,44]

$$f_{\text{mee}} = \sqrt[\alpha]{\frac{P_{\text{in}}}{(\alpha-1)C_{\text{sw}}}}. \tag{3}$$

Since the execution frequency of the processor is discrete, we assume that the minimum available frequency of processor pu_k is $f_{k,\min}$. Therefore, $f_{k,\min}$ should be greater than or equal to f_{mee}. At this point, the available frequency range for processor pu_k is $[f_{k,\min}, f_{k,\max}]$.

Similar to [14–19,26], the switching overhead in the voltage/frequency levels is ignored as it is negligible (e.g., 10 μs–150 μs [26]) relative to task execution time. When task t_i is executed on processor pu_k at an operating frequency of $f_{k,l}$, the energy consumption can be calculated as

$$E(t_i, pu_k, f_{k,l}) = (P_{k,\text{in}} + C_{k,\text{sw}} {f_{k,l}}^{\alpha_k}) \times w_{i,k} \times \frac{f_{k,\max}}{f_{k,l}}. \tag{4}$$

In Eq. (4), we simplify the relationship between task execution time and processor frequency to a linear one, disregarding factors like thermal throttling, voltage-to-frequency conversion inefficiencies, memory access latency, and cache effects. This assumption is widely adopted in DVFS-based scheduling research to make the model more tractable [2,11–19]. Additionally, reducing the execution frequency of tasks will affect their reliability. However, our primary objective is to minimize application execution time within a constrained energy budget. Incorporating techniques like task re-execution or replication would increase energy consumption and extend execution time, which conflicts with our optimization goals. Furthermore, our assumption aligns with common practices in energy-efficient scheduling, where tasks are scheduled without strict reliability requirements. Hence, the reliability requirements of applications are still considered satisfied after the execution frequency is reduced.

When all tasks are assigned, the energy consumption of the application can be calculated as

$$E(A) = \sum_{i=1}^{n} E(t_i, pu_{ap(i)}, f_{ap(i),af(i)}). \tag{5}$$

In Eq. (5), $pu_{ap(i)}$ represents the processor assigned to task t_i, $f_{ap(i),af(i)}$ represents the execution frequency set for task t_i.

When all tasks are assigned to the processor that generates the minimum energy consumption for execution, i.e.

$$E_{\min}(t_i) = \min_{pu_k \in PU} E(t_i, pu_k, f_{k,\min}), \tag{6}$$

the minimum energy consumption required by the application can be calculated as

$$E_{\min}(A) = \sum_{i=1}^{n} E_{\min}(t_i). \tag{7}$$

Similarly, we can calculate the maximum energy demand $E_{\max}(t_i)$ for task t_i and the overall maximum energy demand $E_{\max}(A)$ for the application.

3.4. Task execution time

When tasks in a parallel application are assigned for execution on a heterogeneous multiprocessor system, the earliest start time (EST) and the earliest completion time (ECT) of a task can be respectively calculated as

$$\begin{cases} EST(t_{\text{entry}}, pu_k) = 0 \\ EST(t_i, pu_k) = \max\left(\begin{matrix} freetime[k], \\ \max\limits_{t_j \in pred(t_i)} \{ECT(t_j, pu_{ap(j)}) + c'_{j,i}\} \end{matrix} \right) \end{cases} \tag{8}$$

and

$$ECT(t_j, pu_{ap(j)}) = EST(t_j, pu_{ap(j)}) + w_{j,ap(j)} \times \frac{f_{ap(j),\max}}{f_{ap(j),af(j)}}. \tag{9}$$

In Eq. (8), $freetime[k]$ is the earliest free time of processor pu_k. When the task on processor pu_k completes, processor pu_k becomes available for allocation again. The term $c'_{j,i}$ represents the time required to transfer data from task t_j to task t_i. If t_j is also assigned to processor pu_k, then $c'_{j,i} = 0$; otherwise, $c'_{j,i} = c_{j,i}$.

When all tasks have been allocated to processors, the makespan of the application can be represented as

$$MS(A) = \max_{t_i \in T} ECT(t_i, pu_{ap(i)}), \tag{10}$$

which denotes the time from the start of the first task to the completion of all tasks on all processors.

3.5. Problem description

Consider a parallel application A executing on a heterogeneous distributed system with energy constraints, where the available energy is denoted as $E_{\text{avail}}(A)$. The problem investigated in this paper is how to optimally assign tasks in an application to appropriate processors and adjust their execution frequencies, thereby achieving the minimum makespan for the application while ensuring that the overall energy consumption stays below the available energy budget $E_{\text{avail}}(A)$. This problem can be formalized as minimizing the makespan

$$MS(A) = \max_{t_i \in T} ECT(t_i, pu_{ap(i)}), \tag{11}$$

subject to

$$E(A) \le E_{\text{avail}}(A). \tag{12}$$

4. The proposed algorithms

In this section, we design three algorithms to minimize the makespan of the application. The first algorithm is for minimizing the makespan with energy constraints, the second focuses on energy reclamation, and the third involves the reallocation of energy.

4.1. Minimizing the makespan with energy constraints

To design the algorithm for Minimizing the Makespan with Energy Constraints (MMEC), we first address two key issues: determining energy constraints for tasks and establishing priorities of tasks.

4.1.1. Determining energy constraints of tasks

If $E_{\text{avail}}(A) < E_{\min}(A)$, the application cannot be scheduled, a scenario which is not addressed in this paper. When $E_{\text{avail}}(A) \ge E_{\min}(A)$, the available energy of task t_i can be greater than or equal to $E_{\min}(t_i)$. Thus, our approach initiates with assigning each task its respective minimum demanded energy, followed by reallocating the surplus energy. The surplus energy of the application can be calculated by

$$E_{\text{sur}}(A) = E_{\text{avail}}(A) - E_{\min}(A). \tag{13}$$

In this study, we utilized the average energy consumption of tasks on each processor as a basis for allocating the surplus energy among tasks. Subsequently, the surplus energy obtained by task t_i is given by

$$E_{\text{sur}}(t_i) = E_{\text{sur}}(A) \times \frac{E_{\text{avg}}(t_i)}{E_{\text{total}}} \tag{14}$$

with

$$E_{\text{total}} = \sum_{i=1}^{n} E_{\text{avg}}(t_i) \tag{15}$$

and

$$E_{\text{avg}}(t_i) = \frac{\sum_{k=1}^{m} E(t_i, pu_k, f_{k,\max})}{m}. \tag{16}$$

Therefore, the energy pre-allocated to task t_i can be calculated by

$$E_{\text{pre}}(t_i) = E_{\text{sur}}(t_i) + E_{\text{min}}(t_i) \tag{17}$$

4.1.2. Establishing priorities of tasks

Heterogeneous Earliest Finish Time (HEFT) is a well-known scheduling algorithm in which the *upward rank* method is utilized to determine task priorities [43]. Following this, the *upward rank* approach has been extensively utilized [8,12,26]. In an effort to minimize the makespan of applications, the OCT method was introduced, and scheduling algorithms based on OCT are proven to outperform the HEFT algorithm in most scenarios [42]. Given the energy constraints of the applications considered in this paper, we construct a novel energy-constrained OCT. This OCT is utilized not only to establish priorities for tasks but also to predict the completion times of successor tasks during the scheduling process for currently allocated tasks. OCT oct is an $n \times m$ matrix, where $oct_{i,k}$ represents the maximum value among all shortest paths from the successor tasks of t_i to the exit task, given that task t_i is allocated to processor pu_k. The element $oct_{i,k}$ can be calculated as follows.

$$oct_{i,k} = \max_{t_j \in succ(t_i)} \left[\min_{pu_x \in PU} \left\{ oct_{j,x} + w'_{j,x} + c'_{i,j} \right\} \right] \tag{18}$$

In Eq. (18), $c'_{i,j}$ denotes the time required to transfer data from task t_i to task t_j. If task t_j is also assigned to processor pu_k, then $c'_{i,j} = 0$; otherwise, $c'_{i,j} = c_{i,j}$. $w'_{j,x}$ represents the minimum execution time of task t_j on processor pu_x under energy constraints, which can be calculated as

$$w'_{j,x} = \min_{\substack{f_{x,l} \in [f_{x,\min}, f_{x,\max}], \\ E(t_j, pu_x, f_{x,l}) < E_{\text{pre}}(t_j)}} \left\{ w_{j,x} \times \frac{f_{x,\max}}{f_{x,l}} \right\}. \tag{19}$$

Eq. (19) represents the minimum execution time of task t_j on processor pu_x under energy constraints. If task t_j fails to satisfy the energy constraints even when processor pu_x operates at its minimum frequency, we set $w'_{j,x}$=$w_{j,x} \times \frac{f_{x,\max}}{f_{x,\min}}$. For the exit task t_{exit}, the value of $w'_{\text{exit},x}$ is set to 0 for all processors $pu_x \in PU$.

After the OCT matrix is calculated, the OCT rank of the task can be calculated as

$$OCTRank(t_i) = \frac{\sum_{x=1}^{m} oct_{i,x}}{m}. \tag{20}$$

Tasks with a higher OCT rank value have a higher priority. Because Eq. (18) takes into account the data transfer time between tasks, the priority of any task will not be lower than that of its successor task. OCT reflects the cost of all descendant tasks from each task to the exit task. This information enables informed decisions when assigning tasks to processors. It is important to note that the OCT matrix presented in this paper is constructed based on the energy constraints of the application. Different energy constraints will generate different OCT matrices, which will affect the priority of tasks. Consequently, the OCT matrices we construct are different from those presented in [42], leading to possible variations in task priorities compared to those outlined in [42]. This aspect will be illustrated with examples in Section 4.4.

4.1.3. The MMEC algorithm

Before scheduling the application, we first calculate the pre-allocated energy for each task, and then the OCT table and the OCT rank (priority) for each task can be calculate based on the pre-allocated energy. During the scheduling process, we assign tasks to the processor and set the execution frequency while ensuring that the tasks meet their energy constraints. To assign tasks to the appropriate processors, we calculate the optimistic task completion time based on OCT table, which is defined as

$$ECT_{\text{OCT}}(t_i, pu_k) = ECT(t_i, pu_k) + oct_{i,k}. \tag{21}$$

At this point, we assign tasks to the processor with the minimum optimistic completion time for execution.

Based on Eq. (17), during the scheduling process, the available energy of task t_1 is given as

$$E_{\text{avail}}(t_1) = E_{\text{pre}}(t_1). \tag{22}$$

Once task t_1 is allocated, its actual energy consumption $E(t_1, pu_{ap(1)}, f_{ap(1),af(1)})$ should be less than or equal to $E_{\text{avail}}(t_1)$, at which point the residual energy $E_{\text{res}}(t_1)$ is $E_{\text{avail}}(t_1) - E(t_1, pu_{ap(1)}, f_{ap(1),af(1)})$. This residual energy can be utilized to execute task t_2. Consequently, the available energy for task t_2 is given by

$$E_{\text{avail}}(t_2) = E_{\text{pre}}(t_2) + E_{\text{res}}(t_1). \tag{23}$$

According to this method, the available energy for task t_i ($i \geq 2$) can be determined as

$$E_{\text{avail}}(t_i) = E_{\text{pre}}(t_i) + E_{\text{res}}(t_{i-1}). \tag{24}$$

Based on the above explanations, the MMEC algorithm is designed as shown in **Algorithm 1**, with its details explained as below.

(1) Lines 1–3 respectively calculate the pre-allocated energy for each task, the OCT table, and the OCT rank for each task.

(2) Line 4 sorts the tasks according to their priorities (OCT rank values).

(3) Lines 5–16 comprise a *for* loop which selects a processor and an execution frequency for each task. For each task t_i, the MMEC algorithm initially calculates its available energy (Line 6). Following this, it traverses over all combinations of processors and frequencies (Lines 7–8), computing the $ECT_{\text{OCT}}(t_i, pu_k)$ only for those combinations that satisfy the energy consumption constraints (Line 10). Finally, MMEC assigns task t_i to the processor that yields the minimal $ECT_{\text{OCT}}(t_i, pu_k)$ and configures its execution frequency accordingly (Line 15).

(4) Line 17 calculates the makespan of the application.

Algorithm 1 The MMEC Algorithm

```
Require: $PU = \{pu_1, pu_2, ..., pu_m\}$, application $A$, and $E_{\text{avail}}(A)$
Ensure: $MS(A)$
1: calculate the pre-allocated energy for each task
2: Calculate OCT table
3: Calculate the OCT Rank for each task
4: sort tasks by non-increasing order of $OCTRank$ value
5: for $i \leftarrow 1$ to $n$ do
6:   calculate $E_{\text{avail}}(t_i)$
7:   for each processor $pu_k \in PU$ do
8:     for each frequency $f_{k,l} \in [f_{k,\min}, f_{k,\max}]$ do
9:       if $E(t_i, pu_k, f_{k,l}) \leq E_{\text{avail}}(t_i)$ then
10:        calculate $ECT_{\text{OCT}}(t_i, pu_k)$ using Eq. (21)
11:        record $k$ and $l$ corresponding to the minimum $ECT_{\text{OCT}}(t_i, pu_k)$
12:      end if
13:    end for
14:  end for
15:  assign task $t_i$ to processor $pu_k$ and set the execution frequency to $f_{k,l}$
16: end for
17: calculate $MS(A)$ using Eq. (10)
```

4.1.4. Time complexity of MMEC

In Line 1, MMEC calculates the pre-allocated energy for each task with a time complexity of O($n \times m$). Before calculating the OCT table in Line 2, MMEC needs to calculate $w'_{j,x}$, with a time complexity of

$O(n \times m \times fl)$, where fl denotes the maximum frequency level of the processors. Following that, the time complexity required to calculate the OCT table by MMEC is $O(n^2 \times m)$. In Line 4, MMEC sorts the tasks, and this operation has a time complexity of $O(n \log n)$. Lines 5–16 assign tasks to processors and set their respective execution frequencies, with a time complexity of $O(n^2 \times m \times fl)$. From the analysis outlined, we can conclude that the time complexity of the MMEC algorithm is $O(n^2 \times m \times fl)$.

4.2. Reclaiming energy

4.2.1. Calculating the maximum available execution time

After the application is scheduled using the MMEC algorithm, we observe that there are certain tasks (non-critical tasks), which can be executed with further reduced frequencies without increasing the makespan of the application. This enables us to reclaim a portion of the energy from these tasks for reallocation. To reclaim the energy from these non-critical tasks, we first calculate their maximum available execution time. Following the assignment of tasks by the MMEC algorithm, the start time of any task t_i is denoted as $ST(t_i, pu_{ap(i)})$. With the start time of task t_i remaining unchanged, the latest completion time for task t_i, while not increasing the application's makespan, can be calculated as

$$\begin{cases} LCT(t_{\text{exit}}, pu_k) = MS_{\text{MMEC}}(A) \\ LCT(t_i, pu_k) = \min \begin{pmatrix} \min\limits_{t_j \in succ(t_i)} \left\{ ST(t_j, pu_{ap(j)}) - c'_{i,j} \right\}, \\ \min\limits_{j>i, ap(j)=k} \left\{ ST(t_j, pu_{ap(j)}) \right\} \end{pmatrix} \end{cases}, \tag{25}$$

where $MS_{\text{MMEC}}(A)$ represents the makespan obtained by the MMEC algorithm. Since processors are not reallocated for tasks, the maximum available execution time for task t_i can be calculated as

$$MAET(t_i, pu_{ap(i)}) = LCT(t_i, pu_{ap(i)}) - ST(t_i, pu_{ap(i)}). \tag{26}$$

Algorithm 2 the RE Algorithm

```
Require: the scheduling results of MMEC
Ensure: reclaimed energy
 1: reclaimed energy $re \leftarrow 0$
 2: for $i \leftarrow 1$ to $n$ do
 3:     $maet \leftarrow MAET(t_i, pu_{ap(i)})$
 4:     $k \leftarrow ap(i)$
 5:     for execution frequency $f_{k,l} \leftarrow [f_{k,\min}$ to $f_{k,\max}]$ do
 6:         if $w_{i,k} \times \frac{f_{k,\max}}{f_{k,l}} < maet$ then
 7:             calculate $E(t_i, pu_k, f_{k,l})$ using Eq. (4)
 8:             if $E(t_i, pu_k, f_{k,l}) < E_{\text{MMEC}}(t_i)$ then
 9:                 reset the execution frequency of task $t_i$ to $f_{k,l}$
10:                 $re \leftarrow re + E_{\text{MMEC}}(t_i) - E(t_i, pu_k, f_{k,l})$
11:             end if
12:             break the inner loop
13:         end if
14:     end for
15: end for
```

4.2.2. Reclaiming energy algorithm

Based on the maximum available execution time, the minimum execution frequency for the task can be determined. Consequently, an algorithm for Reclaiming Energy (RE) is outlined as **Algorithm** 2.

The RE algorithm begins by initializing the reclaimed energy re to zero in its first line. Lines 2–15 encompass a *for* loop designed to reclaim energy from each task. For every task t_i, the RE algorithm initially calculates its maximum available execution time (Line 3). It then traverses through processor execution frequencies from low to high. When a task satisfies the execution time requirement at a certain frequency, the energy consumption for that task is calculated (Line 7). If the task's energy consumption is less than the energy required in the MMEC algorithm, the task is assigned a new execution frequency with the excess energy reclaimed (Lines 8–11).

4.2.3. Time complexity of RE

For each task, the RE algorithm first calculates its maximum available execution time, with a complexity of $O(n)$. Then, RE traverses the processor execution frequencies, calculating the execution time and energy consumption for each task, with a time complexity of $O(fl)$. There are a total of n tasks to consider, so the time complexity of RE is $O(n \times (n + fl))$.

4.3. Resetting the execution frequency of tasks

4.3.1. Determining the energy reallocated to tasks

After the RE algorithm is completed, each task has obtained a predetermined amount of energy. During the energy reallocation process, we utilize the energy corresponding to each task from the RE algorithm as a reference to reassign energy to each task. Following this, the energy reallocated to task t_i can be recalculated as

$$E_{\text{realloc}}(t_i) = E_{\text{RE}}(t_i) + re \times \frac{E_{\text{RE}}(t_i)}{E_{\text{total}}}. \tag{27}$$

In Eq. (27), $E_{\text{total}} = \sum_{i=1}^{n} E_{\text{RE}}(t_i)$, where $E_{\text{RE}}(t_i)$ represents the energy consumption associated with the task in the RE algorithm.

Since the execution frequency of the processor is discrete, after the RE algorithm is completed, the execution time of some tasks remains less than their maximum available execution time. We refer to these tasks as non-critical tasks and the others as critical tasks. In the reallocating energy algorithm, the available energy for non-critical tasks is strictly limited to

$$E_{\text{avail}}(t_i) = E_{\text{realloc}}(t_i), \tag{28}$$

whereas for critical tasks, the available energy is

$$E_{\text{avail}}(t_i) = E_{\text{realloc}}(t_i) + E_{\text{res}}(t_{i-1}), \tag{29}$$

where $E_{\text{res}}(t_{i-1})$ represents the residual energy that was not exhausted by the tasks allocated before t_i.

The advantage of this energy allocation strategy is that it prevents accumulated residual energy from being allocated to non-critical tasks when the RE algorithm reclaims less energy. Conversely, when more energy is reclaimed, non-critical tasks, by obtaining this energy, can also increase their execution frequency, thereby reducing their execution time.

4.3.2. Resetting the execution frequency algorithm

According to the available energy of tasks, their execution frequencies can be re-evaluated. Therefore, an algorithm for Resetting the Execution Frequency (REF) of tasks is designed as depicted in **Algorithm** 3.

The REF algorithm attempts to reset the execution frequency of each task based on the newly available energy. For any given task t_i, the REF algorithm first acquires the processor it is assigned to and its current execution frequency, and then calculates its available energy based on whether it is a critical task. If task t_i can increase its execution frequency without violating the energy constraints, its execution frequency is reset accordingly.

4.3.3. Time complexity of REF

In the REF algorithm, since the processor assignment for each task remains unchanged, the time complexity of setting an execution frequency for a task is $O(fl)$. Given that there are a total of n tasks to be considered, so the overall time complexity of the REF algorithm is $O(n \times fl)$.

Algorithm 3 The REF Algorithm

```
Require: the scheduling results of RE
Ensure: $MS(A)$
 1: for $i \leftarrow 1$ to $n$ do
 2:     $k \leftarrow ap(i)$
 3:     $s \leftarrow af(i)$
 4:     $e \leftarrow E_{\text{avail}}(t_i)$
 5:     for execution frequency $f_{k,l} \leftarrow [f_{k,s}$ to $f_{k,\max}]$ do
 6:         calculate $E(t_i, pu_k, f_{k,l})$ using Eq. (4)
 7:         if $E(t_i, pu_k, f_{k,l}) < e$ then
 8:             reset the execution frequency of task $t_i$ to $f_{k,l}$
 9:         end if
10:     end for
11: end for
12: calculate $MS(A)$ using Eq. (10)
```

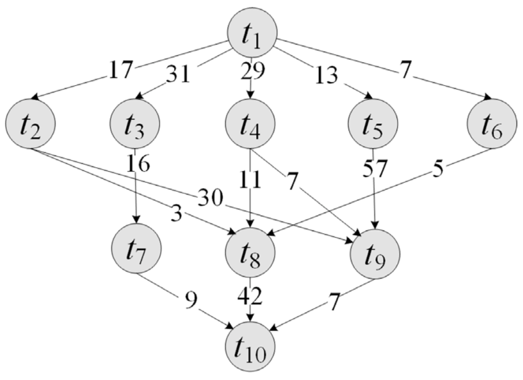


Fig. 2. A case study of an application.

Table 2
WCET of tasks on different processors.

Task	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
pu_1	22	22	32	7	29	26	14	29	15	13
pu_2	21	18	27	10	27	17	25	23	21	16
pu_3	36	18	43	4	35	24	30	36	8	33

Table 3
Power parameters of three processors.

pu_k	$P_{k,\text{in}}$	$C_{k,\text{sw}}$	α_k	$f_{k,\max}$
pu_1	0.03	0.8	2.9	1.0
pu_2	0.04	0.8	2.5	1.0
pu_3	0.07	1.0	2.5	1.0

Table 4
The OCT values and OCT ranks of tasks.

Task	pu_1	pu_2	pu_3	OCT rank
1	86.04	90.30	106.71	94.35
2	54.61	60.33	54.61	56.52
3	30.00	46.00	46.00	40.67
4	55.87	51.61	62.61	56.70
5	35.87	73.94	30.33	46.72
6	55.87	51.61	56.61	54.70
7	14.44	22.86	23.44	20.25
8	14.44	22.86	56.44	31.25
9	14.44	21.44	21.44	19.11
10	0	0	0	0

Table 5
The results of the MMEC algorithm.

Task	$E_{\text{avail}}(t_i)$	$E(t_i)$	pu_k	EF	ST	CT
t_1	13.22	12.34	1	0.8	0	27.50
t_4	4.44	3.93	1	0.8	27.50	36.25
t_2	11.04	9.88	1	0.7	36.25	67.68
t_6	13.30	12.37	2	0.9	34.50	53.39
t_5	16.64	16.27	1	0.8	67.68	103.93
t_3	17.99	16.81	2	0.8	58.50	92.25
t_8	16.60	14.32	2	0.8	92.25	121.00
t_7	12.86	11.62	1	1.0	108.25	122.25
t_9	8.75	8.42	1	0.8	122.25	141.00
t_{10}	10.09	9.96	2	0.8	148.00	168.00
$E(A) = 115.90,\ MS(A) = 168.00$						

Table 6
The results of the RE algorithm.

Task	$E_{\text{avail}}(t_i)$	$E(t_i)$	pu_k	EF	ST	CT
t_1	13.22	12.34	1	0.8	0	27.50
t_4	4.44	3.93	1	0.8	27.50	36.25
t_2	11.04	9.88	1	0.7	36.25	67.68
t_6	13.30	10.58	2	0.8	34.50	55.75
t_5	16.64	16.27	1	0.8	67.68	103.93
t_3	17.99	16.81	2	0.8	58.50	92.25
t_8	16.60	8.35	2	0.5	92.25	138.25
t_7	12.86	11.62	1	1.0	108.25	122.25
t_9	8.75	8.42	1	0.8	122.25	141.00
t_{10}	10.09	9.96	2	0.8	148.00	168.00
$E(A) = 108.15,\ MS(A) = 168.00$						

Table 7
The results of the REF algorithm.

Task	$E_{\text{avail}}(t_i)$	$E(t_i)$	pu_k	EF	ST	CT
t_1	13.23	12.34	1	0.8	0	27.50
t_4	5.09	4.82	1	0.9	27.50	35.28
t_2	10.87	9.88	1	0.7	35.28	66.71
t_6	11.34	10.58	2	0.8	34.50	55.75
t_5	17.44	16.27	1	0.8	66.71	102.96
t_3	20.92	19.64	2	0.9	58.50	88.50
t_8	8.94	8.35	2	0.5	88.50	134.50
t_7	14.33	11.62	1	1.0	104.50	118.50
t_9	11.73	10.32	1	0.9	118.50	135.17
t_{10}	12.08	11.64	2	0.9	142.17	159.94
$E(A) = 115.46,\ MS(A) = 159.94$						

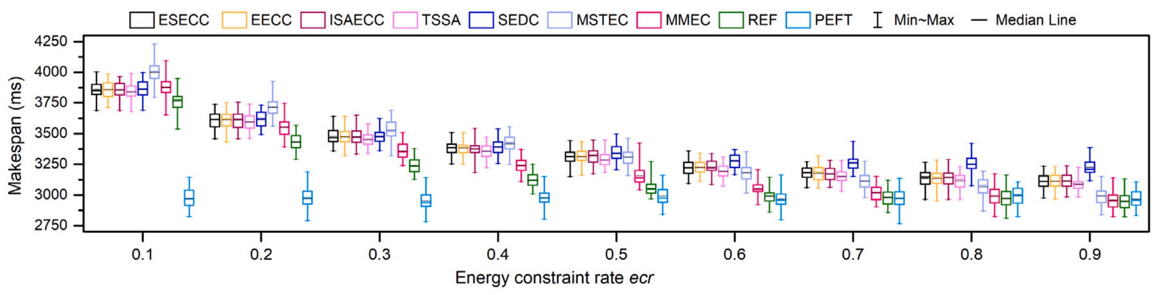
4.4. Case study

This section further illustrates the proposed algorithms through a practical example. The parallel application case is shown in Fig. 2, which is derived from [42]. This DAG (application) contains ten tasks, and their WCETs on three processors are shown in Table 2. The numbers labeled on the edges of the DAG represent the WCRTs. The power consumption parameters of the three processors are shown in Table 3 [17,18]. The execution frequency of the processors are normalized according to their respective maximum frequencies, and the frequency difference between adjacent levels is 0.1.

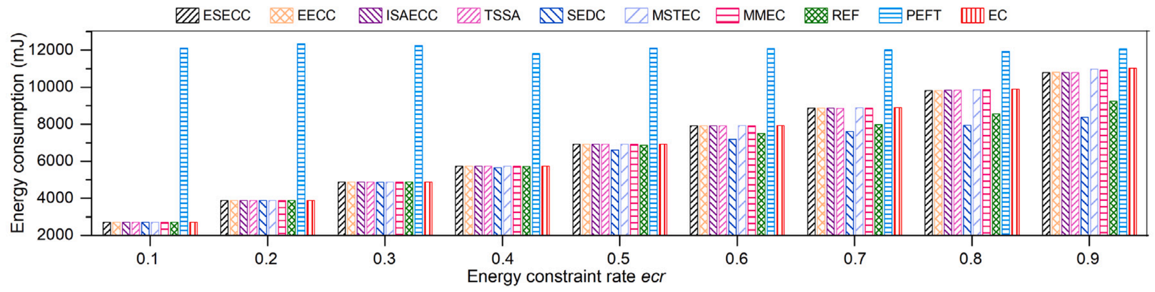
According to Eq. (7), the minimum energy requirement of the application can be calculated as 37.66, and similarly, the maximum energy requirement of the application can also be calculated as 298.89. If the available energy is given as $E_{\text{avail}}(A) = E_{\min}(A) + (E_{\max}(A) - E_{\min}(A)) \times 0.3 = 116.03$, the OCT table can be calculated as shown in Table 4. The scheduling results of the MMEC, RE, and REF algorithms are presented in Tables 5, 6, and 7, where EF, ST, and CT represent execution frequency, start time, and completion time, respectively.

By observing Table 5, it can be seen that the makespan generated by the MMEC algorithm is 168.00, and the energy consumption for executing the application is 115.90, which satisfies the specified energy constraint.

By implementing the RE algorithm, it is found that the non-critical tasks in the results of the MMEC algorithm are t_5, t_6, and t_8. Comparing



(a) Completion times



(b) Energy consumptions

Fig. 3. The results for different energy constraint rates when $fat = 0.6$. (**Experiment 1**).

Tables 5 and 6 reveals that the RE algorithm decreases the execution frequency of task t_6 from 0.9 to 0.8, and reduces the execution frequency of task t_8 from 0.8 to 0.5, while the execution frequency of task t_5 cannot be further decreased. In the end, the makespan generated by the RE algorithm remains at 168.00, while the energy consumption is reduced from 115.90 to 108.15.

By observing Table 7, it can be seen that after executing the REF algorithm, the execution frequencies of tasks t_3, t_9, and t_{10} are increased, resulting in a reduction of the application's makespan from 168.00 to 159.94.

In Section 4.1.2, we mentioned that different energy constraints will affect the priority of tasks. In this case, with an energy constraint of 116.03, the priority order of tasks is t_1, t_4, t_2, t_6, t_5, t_3, t_8, t_7, t_9, t_{10}. If the given available energy is $E_{\text{avail}}(A) = 160$, the priority order of tasks will change to t_1, t_2, t_4, t_6, t_3, t_5, t_8, t_7, t_9, t_{10}. When the available energy $E_{\text{avail}}(A)$ reaches 250, the priority order of tasks is t_1, t_4, t_6, t_2, t_3, t_5, t_8, t_7, t_9, t_{10}, which is the same as the task priority when there is no energy constraint. At this point, the makespan of the application is 122.

5. Performance evaluation

5.1. Experimental parameters

The C++ program is used to implement a scheduling simulator. Before each experiment, we record the processor parameters, the DAG structure of the parallel application, and the WCET of tasks on different processors into respective disk files, ensuring that different algorithms can utilize the same experimental parameters later on. These parameters are primarily sourced from Refs. [17,18], and are listed as follows:

The WCET of task t_i on processor pu_k is 10 ms $\leq w_{i,k} \leq$ 100 ms, and the WCRT between task t_i and task t_j is 10 ms $\leq c_{i,j} \leq$ 100 ms. The frequency-independent active power, the switch capacitor, and the dynamic power exponent of processor pu_k are $0.03 \leq P_{k,\text{in}} \leq 0.07$, $0.8 \leq C_{k,\text{sw}} \leq 1.2$, and $2.5 \leq \alpha_k \leq 3.0$, respectively. The maximum frequency of the processor pu_k is $f_{k,\max}$ = 1.0 GHz and the frequency difference between adjacent levels is 0.1 GHz. These parameters can simulate performance characteristics of some high-performance processors, such as the ARM Cortex-A9 and the Intel Mobile Pentium III [12].

Table 8
Time complexity of algorithms.

Algorithm name	Time complexity
ESECC [15], EECC [16], ISAECC [17], TSSA [18], SEDC [19], MSTEC [37]	$O(n^2 \times m \times fl)$
PEFT [42]	$O(n^2 \times m)$
MMEC	$O(n^2 \times m \times fl)$
RE	$O(n \times (n + fl))$
REF	$O(n \times fl)$

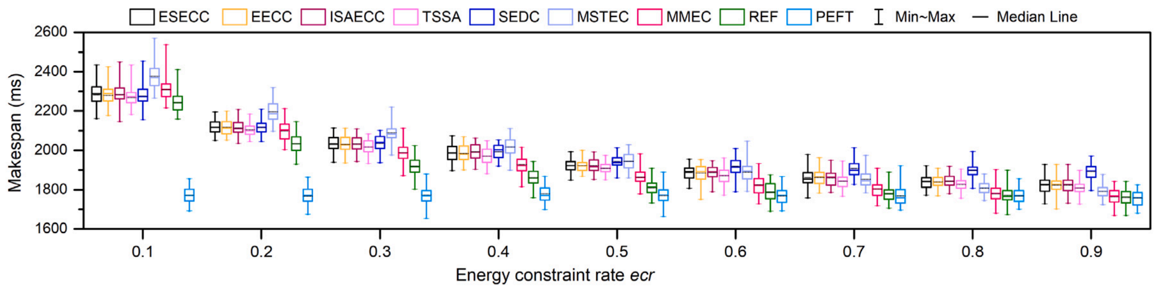
In recent years, there are MSLECC [14], ESECC [15], EECC [16], ISAECC [17], TSSA [18], SEDC [19], and MSTEC [37] algorithms that optimize application's scheduling length (makespan) under energy constraints. In addition, deep reinforcement learning methods have been applied to the field of scheduling, such as [21,24], but their objectives and constraints differ from those of this paper. Therefore, our proposed algorithms are mainly compared with the ESECC [15], EECC [16], ISAECC [17], TSSA [18], SEDC [19], and MSTEC [37] algorithms, as their scheduling results are superior to the MSLECC algorithm. In addition, to observe the scheduling performance of applications without energy constraints, we also include the results of the PEFT algorithm into the experiment. The time complexities of these algorithms are shown in Table 8, where the CPU usage rate of the MSTEC algorithm is determined by frequency level fl in the experiment, and the PEFT algorithm does not adjust the execution frequency of the processor.

To clarify the impact of our proposed the energy reclaiming algorithm and the execution frequency resetting algorithm on the makespan of applications, we presented the results of the MMEC algorithm and the REF algorithm in the experiments. Furthermore, to ensure that the available energy for application A exceeds $E_{\min}(A)$, we introduce the concept of energy constraint rate ecr, which is represented as

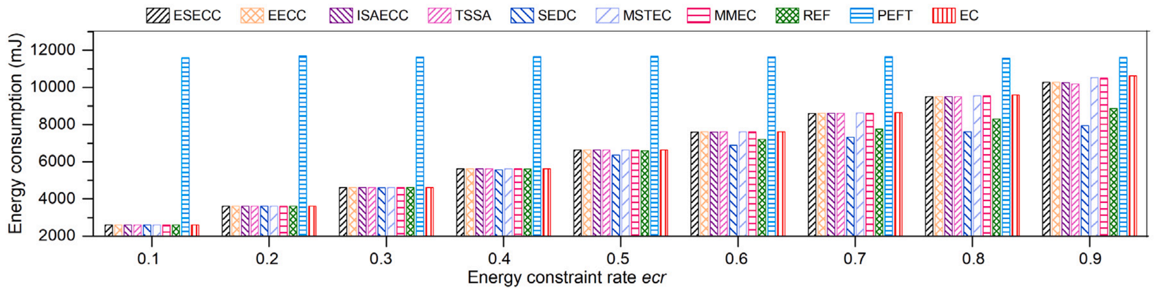
$$ecr = \frac{E_{\text{avail}}(A) - E_{\min}(A)}{E_{\text{PEFT}}(A) - E_{\min}(A)}, \tag{30}$$

where $E_{\text{PEFT}}(A)$ represents the energy required to execute the application when scheduling it using the PEFT algorithm. Therefore, the available energy of the application is given as

$$E_{\text{avail}}(A) = E_{\min}(A) + (E_{\text{PEFT}}(A) - E_{\min}(A)) \times ecr. \tag{31}$$



(a) Completion times



(b) Energy consumptions

Fig. 4. The results for different energy constraint rates when $fat = 1.0$. (**Experiment 1**).

In the following experiments, the number of processors in the platform is 32. To comprehensively evaluate our algorithm, we tested it using randomly generated DAGs of different shapes, real-world application DAGs with different levels of parallelism, as well as real-life industrial DAGs. In each experiment, each algorithm was executed 50 times, resulting in a total of 3400 executions for each algorithm across different scenarios.

5.2. Randomly generated DAG

We first test the performance of each algorithm using randomly generated DAGs. According to the method of generating DAG in [19,42], the shape parameter fat is defined to represent the overall structure of DAG. For a DAG consisting of n tasks, the number of tasks at each level is defined by a uniform distribution with a mean equal to $fat \times \sqrt{n}$. The parameter od is used to represent the out degree of a task, and $jump$ is used to represent an edge connecting a task from the level l to a task in the level $l + jump$. Similar to [19,42], in our experiments, the shape parameter fat takes values from the set $\{0.6, 0.8, 1.0\}$, and the parameter od takes values from the set $\{1, 2, 3, 4, 5\}$. The edges for $jump = 2$ and $jump = 3$ are approximately $n \times 10\%$ and $n \times 5\%$, respectively, while the $jump$ values of the other edges are set to 1.

Experiment 1: This experiment tests the impact of the energy constraint rate on scheduling results. The number of tasks in DAG is fixed at 500. When the energy constraint rate ecr increases from 0.1 to 0.9 with each increment of 0.1, the results of each algorithm with fat set at 0.6 and 1.0 are respectively shown in Figs. 3 and 4.

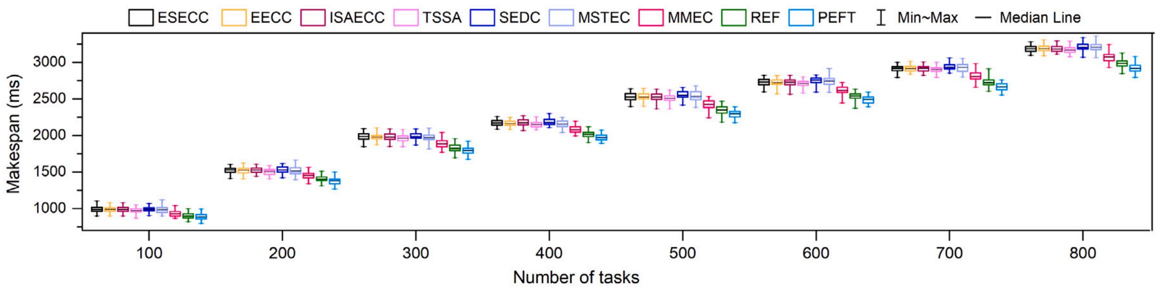
Figs. 3(a) and 4(a) show the box plots of the makespans generated by each algorithm, which display the minimum, the first quartile (the value at the 25th percentile), the median, the third quartile (the value at the 75th percentile), and the maximum. The figures also show the mean, which is indicated by a thin dotted line. From the figures, it can be observed that the PEFT algorithm generates the smallest makespan due to its not reducing the processor's execution frequency. Other algorithms show a trend towards decreasing makespan as the energy constraint rate increases. Among the other eight algorithms, the REF algorithm produces the smallest makespan, followed by the MMEC algorithm. When the energy constraint rate is less than 0.5, the MSTEC algorithm generates the largest makespan. However, when the energy constraint rate is greater than 0.5, the makespan produced by the MSTEC algorithm becomes smaller compared to the other five algorithms ESECC, EECC, ISAECC, TSSA, and SEDC. Additionally, when the energy constraint rate exceeds 0.5, the SEDC algorithm yields a larger makespan than the other algorithms. On average, the three algorithms ESECC, EECC, and ISAECC do not show significant differences in their performance, whereas the TSSA algorithm has a slight advantage over them. When the energy constraint rate reaches 0.8, the makespans generated by MMEC and REF are very close to those produced by PEFT. Specifically, the average makespan produced by the REF algorithm is approximately 94.5% of ESECC, 94.6% of EECC, 94.6% of ISAECC, 95.2% of TSSA, 93.5% of SEDC, 94.1% of MSTEC, and 97.6% of MMEC.

Figs. 3(b) and 4(b) show the average energy consumption of scheduling applications by each algorithm under different energy constraint rates. We have also added a legend indicating the energy constraints (EC), which is used to observe the relationship between the energy consumption produced by each algorithm and the energy constraint. By observing these two figures, it can be seen that all algorithms except PEFT are able to satisfy the energy constraints. When the energy constraint rate ecr is less than 0.5, all algorithms except PEFT have almost exhausted all available energy, and when ecr exceeds 0.5, the energy consumption generated by SEDC and REF algorithms is significantly lower than the energy constraint.

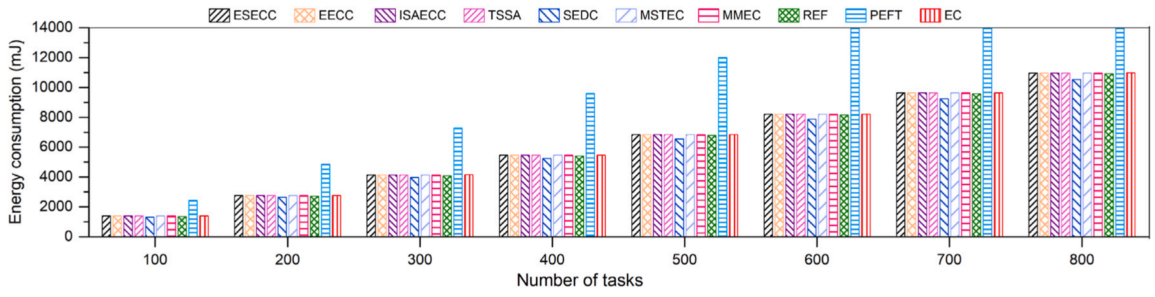
Experiment 2: This experiment tests the impact of the number of tasks on the scheduling results. The energy constraint rate ecr is fixed at 0.5, and the shape parameter fat is set to 0.8. As the number of tasks in the DAG increases from 100 to 800 with each increment of 100, the results of each algorithm are shown in Fig. 5.

From Fig. 5(a), it can be observed that as the number of tasks increases, the makespan generated by all algorithms also increases. Except for the PEFT algorithm, the REF algorithm produces the smallest makespan, followed closely by the MMEC algorithm. Within the other six algorithms, TSSA generates a slightly shorter makespan compared to the remaining five algorithms. Specifically, the average makespan produced by the REF algorithm is approximately 93.0% of ESECC, 93.0% of EECC, 92.9% of ISAECC, 93.7% of TSSA, 92.4% of SEDC, 92.9% of MSTEC, and 97.0% of MMEC.

As shown in Fig. 5(b), the PEFT algorithm fails to meet the energy constraint, while all other algorithms satisfy the energy constraint and almost exhaust all available energy.

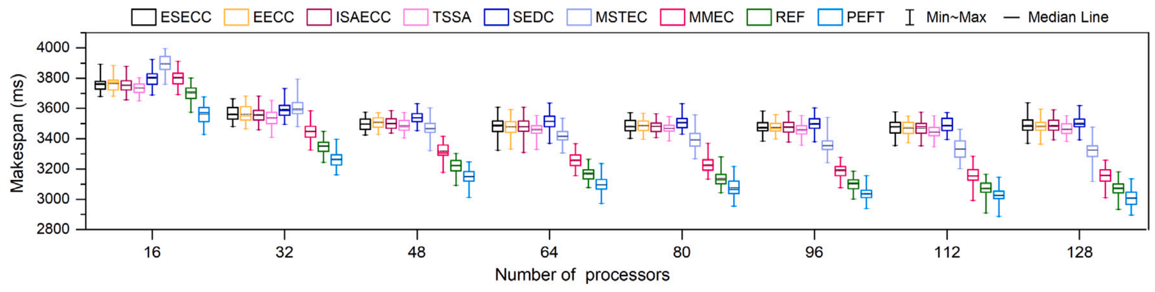


(a) Completion times

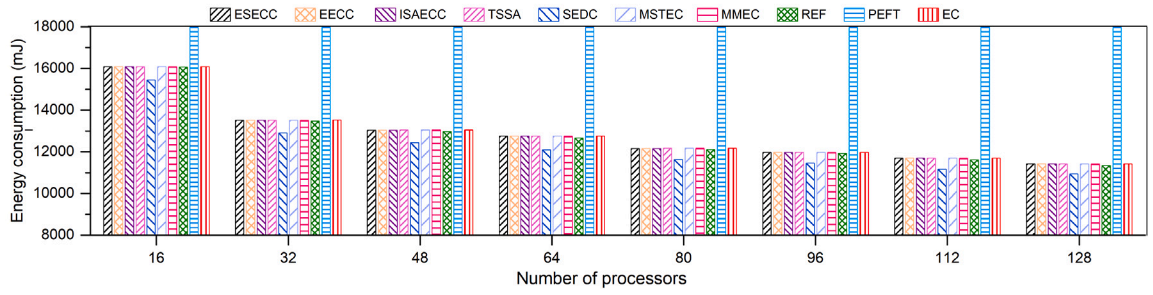


(b) Energy consumptions

Fig. 5. The results for different numbers of tasks. (**Experiment 2**).



(a) Completion times



(b) Energy consumptions

Fig. 6. The results for different numbers of processors. (**Experiment 3**).

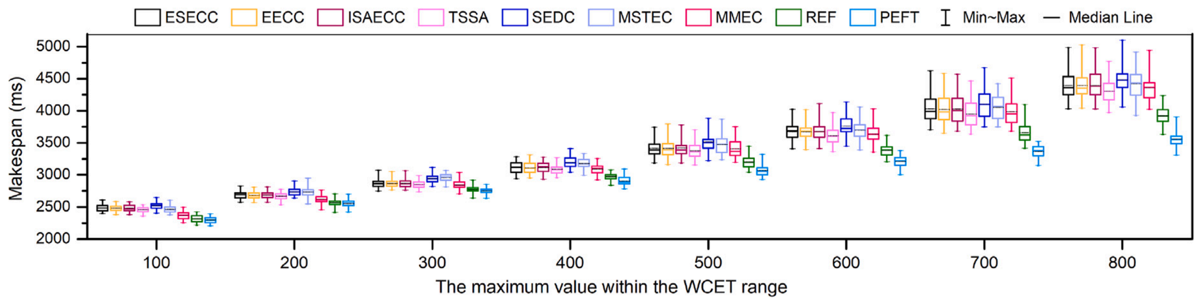


Fig. 7. Results for different WCET value ranges (**Experiment 4**).

Experiment 3: This experiment tests the impact of the number of processors on scheduling results. The number of tasks in DAG is set to 1000, the shape parameter fat is set to 0.8, and the energy constraint rate ecr is set to 0.5. The results of different algorithms are illustrated in Fig. 6 when the number of processors in the platform is 16, 32, 48, 64, 80, 96, 112, and 128, respectively.

As shown in Fig. 6(a), when the number of processors is less than 80, the makespan generated by all algorithms decreases as the number of processors increases. When the number of processors exceeds 80, there is no significant change in the makespans generated by all algorithms. Overall, the PEFT algorithm produces the smallest makespan, followed by REF and MMEC. When the number of processors is large (such as over 48), the MSTEC algorithm produces a smaller makespan than the other five algorithms. The makespans generated by the ESECC, EECC, and ISAECC algorithms show no significant difference, but they are all slightly smaller than the makespan produced by the SEDC algorithm and slightly larger than the makespan generated by TSSA. Specifically, the average makespan produced by the REF algorithm is approximately 91.5% of ESECC, 91.5% of EECC, 91.5% of ISAECC, 92.1% of TSSA, 90.8% of SEDC, and 92.9% of MSTEC.

As shown in Fig. 6(b), all algorithms except PEFT are able to meet the energy consumption constraint requirements. With the exception of the SEDC algorithm, the other seven algorithms have almost exhausted all the available energy. Because we have already understood the approximate relationship between the energy consumption and available energy generated by each algorithm through the previous three experiments, we will not draw energy consumption histograms for each algorithm in the following experiments.

Experiment 4: This experiment tests the impact of different task sizes on the performance of various algorithms. The number of tasks is set to 500, with a shape parameter fat of 0.8 and an energy constraint rate of 0.6. The WCET range is set to [10 ms, sw ms]. When sw increases from 100 to 800 with each increment of 100, the makespans generated by each algorithm are shown in Fig. 7.

From Fig. 7, it can be seen that the makespans generated by all algorithms increase as sw increases. Among all the algorithms, the makespans generated by ESECC, EECC, and ISAECC have little difference and are all smaller than those generated by SEDC and MSTEC. The makespans generated by both TSSA and REF are competitive, with REF producing a smaller makespan compared to TSSA. On average, the makespan produced by the REF algorithm is 93.0% of ESECC, 93.1% of EECC, 93.0% of ISAECC, 94.2% of TSSA, 91.0% of SEDC, 92.0% of MSTEC and 94.3% of MMEC.

5.3. Real-world application DAG

In addition to randomly constructed DAGs, we also tested the algorithm's performance using real-world applications, such as Gaussian Elimination (GE) and Fourier Transform (FT), which exhibit characteristics of low parallelism and high parallelism, respectively [11,26,45]. In our experiments, a parameter v is introduced to describe the scale size of the application. In the GE application, the total number of tasks is $n = (v^2 + v - 2)/2$; whereas in the FT application, it is $n = (2 \times v - 1) + v \times \log_2 v$, where $v = 2^u$ with u is a nonnegative integer.

Experiment 5: The FT application with scale size $v = 64$ (511 tasks) is used to test the performance of different algorithms under varying energy consumption rates. The results of different algorithms are shown in Fig. 8 when the energy constraint rate ecr increases from 0.1 to 0.9 with each increment of 0.1.

As shown in Fig. 8, the makespan produced by PEFT is the smallest and remains almost constant, whereas the makespan generated by other algorithms decreases as the energy constraint increases. When the energy constraint rate ecr is 0.1, the makespan produced by MSTEC, MMEC, and REF is greater than that of the other five algorithms. After ecr reaches 0.2, REF generates the smallest makespan (except for PEFT), followed by MMEC. When ecr is less than 0.5, the makespan generated by the MSTEC algorithm is greater than that generated by the ESECC, EECC, ISAECC, and TSSA algorithms. However, when ecr exceeds 0.5, the makespan generated by MSTEC becomes less than the makespans produced by these four algorithms. On average, the makespan produced by the REF algorithm is approximately 95.1% of ESECC, 95.5% of EECC, 95.1% of ISAECC, 95.6% of TSSA, 93.6% of SEDC, 94.9% of MSTEC, and 98.3% of MMEC.

Experiment 6: GE applications with different numbers of tasks are employed for evaluating different algorithms. The energy constraint rate ecr is fixed at 0.5. When the scale size v increases from 12 to 40 with each increment of 4 (corresponding to task numbers of 77, 135, 209, 299, 405, 527, 665, and 819 respectively), the results of each algorithm are shown in Fig. 9.

From Fig. 9, it can be seen that the makespans generated by all algorithms increase with the increase of the number of tasks. Among these, the makespan generated by the ESECC, EECC, ISAECC, and SEDC algorithms does not show a significant difference and is relatively larger. The other five algorithms, ranked in ascending order of makespan, are PEFT, REF, MMEC, MSTEC, and TSSA. In this experiment, our proposed algorithms have significant competitiveness. On average, the makespan produced by the REF algorithm is approximately 75.8% of ESECC, 75.9% of EECC, 75.9% of ISAECC, 77.3% of TSSA, 75.8% of SEDC, 88.8% of MSTEC, and 94.7% of MMEC.

The analysis of the reasons for the aforementioned experimental results is as follows:

(1) Although all algorithms assign energy to tasks in different ways, they generally aim for a relatively balanced allocation of energy. This means that no task is assigned significantly more energy than others while some receive disproportionately less. The ESECC, EECC, and ISAECC algorithms all utilize the *upward rank* method to determine the priority of tasks, implying that any given task has the same priority across these three algorithms. Consequently, while the makespan generated by these three algorithms for a single application might vary. On average, the makespans generated by these three algorithms are unlikely to have significant differences.

(2) The TSSA algorithm also employs the upward rank method to determine the priority of tasks. However, TSSA considers scheduling after reversing the task graph. As a result, TSSA can produce a shorter makespan than ESECC, EECC, and ISAECC under certain scenarios.

(3) SEDC adopts a method based on the time difference coefficient to calculate the priority of tasks, which arranges tasks more reasonably. Therefore, the SEDC algorithm performs better in some scenarios (such as when $ecr = 0.1$ in **Experiment 5**). However, in the energy allocation scheme of SEDC, the residual energy of task t_i is reclassified into the available energy of the subsequent $n - i$ tasks, resulting in lower-priority tasks (tasks assigned later) receiving more energy. Consequently, when the energy constraint rate is high (such as exceeding 0.5), some low-priority tasks consume significantly less energy than their energy constraints, leading to the algorithm generating much less energy than its energy constraints. Due to some energy not being fully utilized by high-priority tasks, it has a negative impact on optimizing the makespan.

(4) MSTEC also calculates the priority of tasks based on their OCT values. However, MSTEC does not consider energy constraints when calculating OCT. When the energy constraint rate is low (e.g., less than 0.6), processors must execute tasks at lower frequencies, leading to execution times that differ significantly from those at the highest frequency. In this case, OCT does not accurately reflect the actual conditions of the tasks, resulting in poorer performance of the algorithm. However, when the energy constraint rate is high, processors can operate at higher frequencies, and OCT more closely reflects the actual conditions of the tasks, leading to better relative performance of MSTEC when the energy constraint rate is high (see **Experiment 1** and **Experiment 5**).

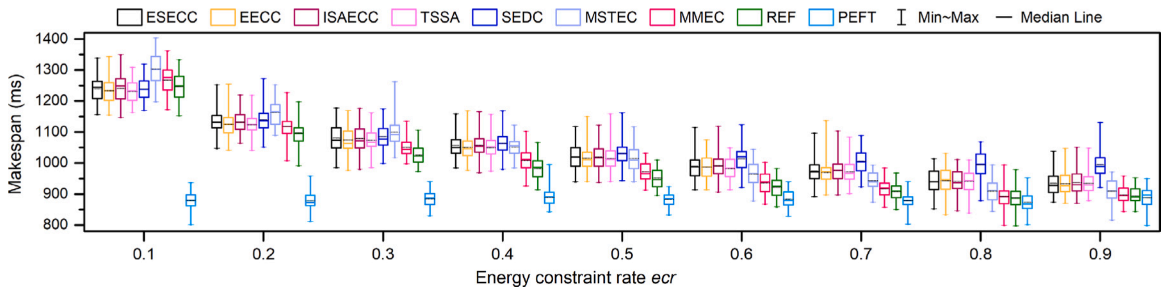


Fig. 8. Results for different energy constraint rates (**Experiment 5**).

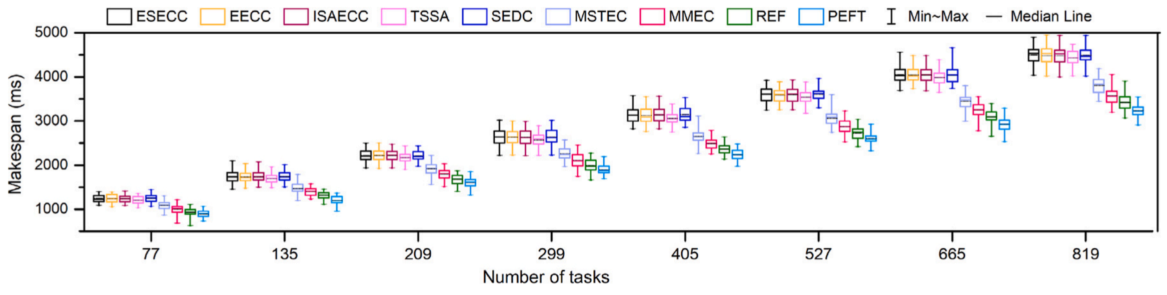


Fig. 9. Results for different numbers of tasks (**Experiment 6**).

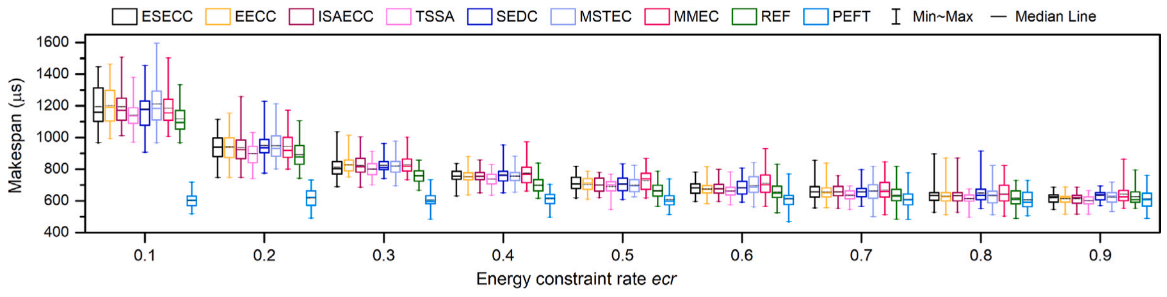


Fig. 10. The results for different energy constraint rates (**Experiment 7**).

(5) MMEC takes energy constraints into account when calculating OCT, resulting in OCT values for tasks that more accurately reflect their actual conditions. When MMEC assigns a task to a processor, it not only considers the earliest completion time of the current task but also attempts to achieve shorter completion times for subsequent tasks through OCT. Therefore, MMEC generates a relatively short makespan. Building upon MMEC, the REF algorithm further optimizes the makespan, resulting in the smallest makespan in most cases.

(6) In all the algorithms that consider energy constraints (excluding SEDC), the remaining energy after scheduling task t_i is mainly added to the energy budget of task t_{i+1}, rather than being reassigned to all subsequent tasks (t_{i+1} to t_n). This approach ensures that tasks scheduled later do not receive a larger share of available energy. As a result, these algorithms will essentially exhaust all available energy. Due to the REF algorithm reallocating the energy collected by the RE algorithm, it cannot assign additional energy when the execution frequency of critical tasks in the MMEC algorithm has already reached its maximum. Consequently, when the energy constraint ratio is high, the energy consumption produced by the REF algorithm will be significantly lower than the energy constraint (refer to **Experiment 1**).

5.4. Real-life industrial DAG

Experiment 7: This section employs a specific real-life industrial DAG, previously used for testing algorithm performance in [18], to evaluate different algorithms. As depicted in Fig. 11, the DAG consists of six components: namely, the engine controller ($t_1 - t_7$), the automatic gearbox ($t_8 - t_{11}$), the anti-locking brake system ($t_{12} - t_{17}$), the wheel angle sensor ($t_{18} - t_{19}$), the suspension controller ($t_{20} - t_{24}$), and the bodywork ($t_{25} - t_{31}$). In the experiment, the parameters of the DAG were taken from [18] and are listed as follows: $100\ \mu s \le w_{i,k} \le 400\ \mu s$, $100\ \mu s \le c_{i,j} \le 400\ \mu s$. Similar to [18], the number of processors is set to 16. When the energy constraint rate *ecr* increases from 0.1 to 0.9, the scheduling results of each algorithm are shown in Fig. 10.

From Fig. 10, it can be observed that as the energy constraint rate increases, the makespans generated by all algorithms (except for PEFT) tend to decrease. Overall, there is not much difference in the performance of the ESECC, EECC, and IASECC algorithms. The makespans generated by them are slightly larger than those generated by TSSA and REF but slightly smaller than those generated by MSTEC and MMEC. When the energy constraint rate is 0.1, the average makespan generated by SEDC is slightly smaller than that produced by the ESECC, EECC, and IASECC algorithms. However, once the energy constraint rate reaches 0.2, the average makespan generated by SEDC becomes slightly larger than that produced by the ESECC, EECC, and IASECC algorithms. The

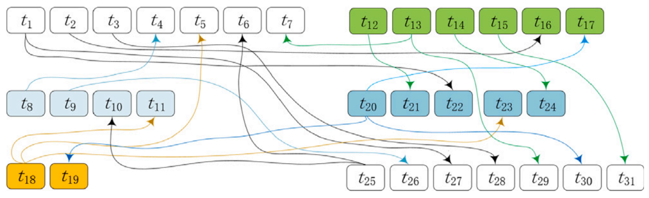


Fig. 11. Real-life industrial DAG.

makespan generated by MMEC is greater than that produced by ESECC, EECC, IASECC, TSSA, and SEDC in most cases. Thanks to the collection and reallocation of energy, the REF algorithm generates relatively small makespans in most cases. When the energy constraint rate is less than or equal to 0.8, the average makespan generated by REF is smaller than that produced by other algorithms (except for the PEFT algorithm). Overall, the makespans generated by TSSA and REF are relatively small. The average makespan produced by REF is 94.6% of ESECC, 94.6% of EECC, 94.7% of ISAECC, 97.6% of TSSA, 93.9% of SEDC, 93.9% of MSTEC, and 92.9% of MMEC.

Based on the results from the aforementioned seven experiments, the following summary can be derived (without considering PEFT):

(1) As the energy constraint rate increases, the makespan generated by all algorithms tends to decrease.

(2) The relative performance of different algorithms in terms of makespan varies across different scenarios, which indicates that none of the eight algorithms compared in this paper outperforms the others in every scenario.

(3) The REF algorithm can generate the minimum makespan in most scenarios and consumes significantly less energy than the energy constraint. This indicates that in most scenarios, the REF algorithm not only produces the smallest makespan, but also has relatively lower energy consumption compared to other algorithms.

6. Conclusions

In this paper, we have investigated the scheduling problem for parallel applications on heterogeneous systems with energy constraints, introducing three algorithms designed to minimize the makespan. To the best of our knowledge, the method of creating task priorities based on available energy is being proposed for the first time. This approach provides a fresh perspective for task scheduling strategies in energy-constrained environments, offering new insights into how energy availability can influence scheduling decisions. The experimental results demonstrate that the algorithms proposed in this paper produce a smaller makespan than existing algorithms in most scenarios, alongside relatively lower energy consumption. This highlights the effectiveness of our proposed algorithms in energy-constrained situations. In future work, we will attempt to apply artificial intelligence methods to the scheduling of parallel applications under energy constraints, and consider the overheads associated with context switching during task execution and processor voltage-frequency scaling.

CRediT authorship contribution statement

Hongzhi Xu: Writing – original draft, Validation, Software, Methodology, Investigation, Conceptualization. **Binlian Zhang:** Validation, Software, Methodology, Investigation. **Chen Pan:** Writing – review & editing, Methodology, Investigation. **Keqin Li:** Writing – review & editing, Visualization, Methodology.

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

The authors express gratitude to the anonymous reviewers for their constructive comments, which have helped improve the manuscript. The research was partially funded by the National Natural Science Foundation of China (Grant Nos. 62062036)

Data availability

Data will be made available on request.

References

[1] R. Kavanagh, K. Djemame, J. Ejarque, R.M. Badia, D. Garcia-Perez, Energy-aware self-adaptation for application execution on heterogeneous parallel architectures, IEEE Trans. Sustain. Comput. 5 (1) (2020) 81–94, http://dx.doi.org/10.1109/TSUSC.2019.2912000.

[2] J. Peng, K. Li, J. Chen, K. Li, Reliability/performance-aware scheduling for parallel applications with energy constraints on heterogeneous computing systems, IEEE Trans. Sustain. Comput. 7 (3) (2022) 681–695, http://dx.doi.org/10.1109/TSUSC.2022.3146138.

[3] H.-E. Zahaf, A.E.H. Benyamina, R. Olejnik, G. Lipari, Energy-efficient scheduling for moldable real-time tasks on heterogeneous computing platforms, J. Syst. Archit. 74 (2017) 46–60, http://dx.doi.org/10.1016/j.sysarc.2017.01.002.

[4] K. Li, Power and performance management for parallel computations in clouds and data centers, J. Comput. System Sci. 82 (2) (2016) 174–190.

[5] A. Suyyagh, Z. Zilic, Energy and task-aware partitioning on single-ISA clustered heterogeneous processors, IEEE Trans. Parallel Distrib. Syst. 31 (2) (2020) 306–317, http://dx.doi.org/10.1109/TPDS.2019.2937029.

[6] H.A. Hassan, S.A. Salem, E.M. Saad, A smart energy and reliability aware scheduling algorithm for workflow execution in DVFS-enabled cloud environment, Future Gener. Comput. Syst. 112 (2020) 431–448, http://dx.doi.org/10.1016/j.future.2020.05.040.

[7] Z. Wu, L. Han, J. Liu, Y. Robert, F. Vivien, Energy-aware mapping and scheduling strategies for real-time workflows under reliability constraints, J. Parallel Distrib. Comput. 176 (2023) 1–16, http://dx.doi.org/10.1016/j.jpdc.2023.02.004.

[8] H. Xu, B. Zhang, C. Pan, K. Li, Energy-efficient triple modular redundancy scheduling on heterogeneous multi-core real-time systems, J. Parallel Distrib. Comput. 191 (2024) 104915, http://dx.doi.org/10.1016/j.jpdc.2024.104915.

[9] P. Zhu, D. Luo, X. Chen, Fault-tolerant and power-aware scheduling in embedded real-time systems, in: 2020 International Conference on Computer, Information and Telecommunication Systems, CITS, 2020, pp. 1–5, http://dx.doi.org/10.1109/CITS49457.2020.9232471.

[10] Z. Tang, L. Qi, Z. Cheng, K. Li, S.U. Khan, K. Li, An energy-efficient task scheduling algorithm in DVFS-enabled cloud environment, J. Grid Comput. 14 (2016) 55–74, http://dx.doi.org/10.1007/s10723-015-9334-y.

[11] 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, http://dx.doi.org/10.1109/TII.2017.2676183.

[12] 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, http://dx.doi.org/10.1109/TPDS.2017.2730876.

[13] J. Huang, R. Li, J. An, H. Zeng, W. Chang, A DVFS-weakly dependent energy-efficient scheduling approach for deadline-constrained parallel applications on heterogeneous systems, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 40 (12) (2021) 2481–2494, http://dx.doi.org/10.1109/TCAD.2021.3049688.

[14] 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, 2016, pp. 1471–1476, http://dx.doi.org/10.1109/TrustCom.2016.0230.

[15] 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 Processing with Applications and 2017 IEEE International Conference on Ubiquitous Computing and Communications (ISPA/IUCC), 2017, pp. 32–39, http://dx.doi.org/10.1109/ISPA/IUCC.2017.00015.

[16] 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, http://dx.doi.org/10.1142/S0218126619501901.

[17] Z. Quan, Z.-J. Wang, T. Ye, S. Guo, Task scheduling for energy consumption constrained parallel applications on heterogeneous computing systems, IEEE Trans. Parallel Distrib. Syst. 31 (5) (2020) 1165–1182, http://dx.doi.org/10.1109/TPDS.2019.2959533.

[18] 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, http://dx.doi.org/10.1016/j.sysarc.2022.102741.

[19] 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, http://dx.doi.org/10.1016/j.sysarc.2023.102938.

[20] M.A. Beg, M. Alam, M. Shahid, A workflow allocation strategy under precedence constraints for IaaS cloud environment, in: Innovations in Computational Intelligence and Computer Vision: Proceedings of ICICV 2020, Springer Singapore, Springer, 2021, pp. 10–17.

[21] A. Jayanetti, S. Halgamuge, R. Buyya, Deep reinforcement learning for energy and time optimized scheduling of precedence-constrained tasks in edge-cloud computing environments, Future Gener. Comput. Syst. 137 (2022) 14–30, http://dx.doi.org/10.1016/j.future.2022.06.012.

[22] M. Alam, M. Shahid, S. Mustajab, F. Ahmad, Security driven dynamic level scheduling under precedence constrained tasks in IaaS cloud, Int. J. Inf. Technol. 16 (2) (2024) 721–729, http://dx.doi.org/10.1007/s41870-023-01523-0.

[23] M.I. Khaleel, A dynamic weight-assignment load balancing approach for workflow scheduling in edge-cloud computing using ameliorated moth flame and rock hyrax optimization algorithms, Future Gener. Comput. Syst. 155 (2024) 465–485, http://dx.doi.org/10.1016/j.future.2024.02.025.

[24] G.P. Koslovski, K. Pereira, P.R. Albuquerque, DAG-based workflows scheduling using actor-critic deep reinforcement learning, Future Gener. Comput. Syst. 150 (2024) 354–363, http://dx.doi.org/10.1016/j.future.2023.09.018.

[25] M. Hussain, L.-F. Wei, A. Rehman, F. Abbas, A. Hussain, M. Ali, Deadline-constrained energy-aware workflow scheduling in geographically distributed cloud data centers, Future Gener. Comput. Syst. 132 (2022) 211–222, http://dx.doi.org/10.1016/j.future.2022.02.018.

[26] 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) (2018) 167–181, http://dx.doi.org/10.1109/TSUSC.2017.2711362.

[27] H. Xu, R. Li, C. Pan, K. Li, Minimizing energy consumption with reliability goal on heterogeneous embedded systems, J. Parallel Distrib. Comput. 127 (2019) 44–57, http://dx.doi.org/10.1016/j.jpdc.2019.01.006.

[28] L. Ye, Y. Xia, S. Tao, C. Yan, R. Gao, Y. Zhan, Reliability-aware and energy-efficient workflow scheduling in IaaS clouds, IEEE Trans. Autom. Sci. Eng. 20 (3) (2023) 2156–2169, http://dx.doi.org/10.1109/TASE.2022.3195958.

[29] Y. Han, J. Liu, W. Hu, Y. Gan, High-reliability and energy-saving DAG scheduling in heterogeneous multi-core systems based on task replication, in: 2021 IEEE International Conference on Systems, Man, and Cybernetics, SMC, IEEE, 2021, pp. 2012–2017.

[30] S.K. Biswas, P.K. Muhuri, U.K. Roy, Binary search-based fast scheduling algorithms for reliability-aware energy-efficient task graph scheduling with fault tolerance, IEEE Trans. Sustain. Comput. 9 (3) (2024) 433–451, http://dx.doi.org/10.1109/TSUSC.2023.3295939.

[31] B. Hu, Z. Cao, M. Zhou, Energy-minimized scheduling of real-time parallel workflows on heterogeneous distributed computing systems, IEEE Trans. Serv. Comput. 15 (5) (2022) 2766–2779, http://dx.doi.org/10.1109/TSC.2021.3054754.

[32] H. Xu, B. Zhang, C. Pan, K. Li, Energy-efficient scheduling for parallel applications with reliability and time constraints on heterogeneous distributed systems, J. Syst. Archit. 152 (2024) 103173, http://dx.doi.org/10.1016/j.sysarc.2024.103173.

[33] J. Liu, Z. Zhu, C. Deng, A novel and adaptive transient fault-tolerant algorithm considering timing constraint on heterogeneous systems, IEEE Access 8 (2020) 103047–103061, http://dx.doi.org/10.1109/ACCESS.2020.2999092.

[34] D. Mao, W. Hu, Y. Gan, J. Liu, H. Gu, A fault-tolerant scheduling algorithm based on local maximum reliability replication strategy in real-time heterogeneous systems, in: 2022 IEEE International Conference on Systems, Man, and Cybernetics, SMC, IEEE, 2022, pp. 3192–3197, http://dx.doi.org/10.1109/SMC53654.2022.9945550.

[35] K.K. Chakravarthi, L. Shyamala, TOPSIS inspired budget and deadline aware multi-workflow scheduling for cloud computing, J. Syst. Archit. 114 (2021) 101916, http://dx.doi.org/10.1016/j.sysarc.2020.101916.

[36] S.C. Ghoshal, M.M. Hossain, B.C. Das, P. Roy, M.A. Razzaque, S. Azad, M.M. Hassan, C. Savaglio, G. Fortino, VESBELT: An energy-efficient and low-latency aware task offloading in maritime Internet-of-Things networks using ensemble neural networks, Future Gener. Comput. Syst. 161 (2024) 572–585, http://dx.doi.org/10.1016/j.future.2024.07.034.

[37] H. Li, X. Zhang, H. Li, X. Duan, C. Xu, SLA-based task offloading for energy consumption constrained workflows in fog computing, Future Gener. Comput. Syst. 156 (2024) 64–76, http://dx.doi.org/10.1016/j.future.2024.03.013.

[38] A. Hazra, P.K. Donta, T. Amgoth, S. Dustdar, Cooperative transmission scheduling and computation offloading with collaboration of fog and cloud for industrial IoT applications, IEEE Internet Things J. 10 (5) (2023) 3944–3953, http://dx.doi.org/10.1109/JIOT.2022.3150070.

[39] Q. Wang, W. Li, A. Mohajer, Load-aware continuous-time optimization for multi-agent systems: toward dynamic resource allocation and real-time adaptability, Comput. Netw. 250 (2024) 110526, http://dx.doi.org/10.1016/j.comnet.2024.110526.

[40] YangTing, SunJiabao, MohajerAmin, Queue stability and dynamic throughput maximization in multi-agent heterogeneous wireless networks, Wirel. Netw. (2024) 3229–3255, http://dx.doi.org/10.1007/s11276-024-03730-4.

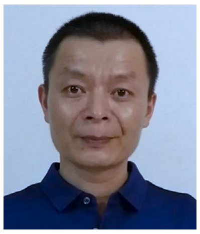
[41] L. Gu, A. Mohajer, Joint throughput maximization, interference cancellation, and power efficiency for multi-IRS-empowered UAV communications, Signal Image Video Process. 18 (5) (2024) 4029–4043.

[42] 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, http://dx.doi.org/10.1109/TPDS.2013.57.

[43] 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, http://dx.doi.org/10.1109/71.993206.

[44] B. Zhao, H. Aydin, D. Zhu, Shared recovery for energy efficiency and reliability enhancements in real-time applications with precedence constraints, ACM Trans. Des. Autom. Electron. Syst. 18 (2) (2013) 23, http://dx.doi.org/10.1145/2442087.2442094.

[45] 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) (2017) 1629–1640, http://dx.doi.org/10.1109/TII.2016.2641473.



Hongzhi Xu received the Ph.D. degree in computer science and engineering from Hunan University, Changsha, China, in 2018. He is a professor at Jishou University, Zhangjiajie, China. His research interests include heterogeneous computing systems and energy-efficient computing.

Binlian Zhang received the M.S. degree in computer science and engineering from Hunan Normal University, Changsha, China, in 2007. She is an associate professor at Jishou University, Zhangjiajie, China. Her research interests include heterogeneous computing systems and energy-efficient computing.

Chen Pan received his M.S. degree in Electrical Engineering from Oklahoma State University in 2017 and his Ph.D. degree in Electrical and Computer Engineering from the University of Pittsburgh in 2019. Dr. Pan is currently a tenure-track Assistant Professor with the Department of ECE at UTSA and the director of the Resilient, Intelligent, and Sustainable Embedded Computing and Networking (RISE). Dr. Pan's research interests include Sustainable Air-ground IoT Systems, Tiny Machine Learning, Intelligent Sparse Sensing, Transient Computing and Communication, and Emerging Non-volatile Memory in the broad area of IoT. Dr. Pan has published widely in top conference and journal venues, including DAC, DATE, ASP-DAC, ISQED, EMSOFT, CODES+ISSS, RTCSA, LCTES, TVLSI, TMSCS, TECS, TCAD, TODAES, and TCPS. Dr. Pan's research projects are funded by various esteemed organizations including NSF, USDA, NOAA, and TGLO.

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 1000 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 nearly 75 patents announced or authorized by the Chinese National Intellectual Property Administration. He is among the world's top five most influential scientists in parallel and distributed computing in terms of single-year and career-long impacts 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'n Science and Engineering, Who's Who in America, Who's Who'n the World, and Who's Who'n 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 is a Member of the SUNY Distinguished Academy. He is an AAAS Fellow, an IEEE Fellow, an AAIA Fellow, and an ACIS Founding Fellow. He is an Academician Member and Fellow of the International Artificial Intelligence Industry Alliance. He is a Member of Academia Europaea (Academician of the Academy of Europe).