Received January 13, 2020, accepted January 18, 2020, date of publication January 29, 2020, date of current version February 7, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2970166

Energy-Aware Task Scheduling on Heterogeneous Computing Systems With Time Constraint

ZEXI DENG[1], ZIHAN YAN[1], HUIMIN HUANG[1], AND HONG SHEN[1,2]
[1]School of Data and Computer Science, Sun Yat-sen University, Guangzhou 510006, China
[2]School of Computer Science, The University of Adelaide, Adelaide, SA 5005, Australia

Corresponding authors: Zexi Deng (zexideng@qq.com) and Hong Shen (shenh3@mail.sysu.edu.cn)

This work was supported in part by the National Key Research and Development Program of China under project number 2017YFB0203201, and in part by the Australian Research Council Discovery under project number DP150104871.

ABSTRACT As a technique to help achieve high performance in parallel and distributed heterogeneous computing systems, task scheduling has attracted considerable interest. In this paper, we propose an effective Cuckoo Search algorithm based on Gaussian random walk and Adaptive discovery probability which combined with a cost-to-time ratio Modification strategy (GACSM), to address task scheduling on heterogeneous multiprocessor systems using Dynamic Voltage and Frequency Scaling (DVFS). First, to overcome the shortcomings of poor performance in exploitation of the cuckoo search algorithm, we use chaos variables to initialize populations to maintain the population diversity, a Gaussian random walk strategy to balance the exploration and exploitation capabilities of the algorithm, and an adaptive discovery probability strategy to improve population diversity. Then, we apply the improved Cuckoo Search (CS) algorithm to assign tasks to resources, and a widely used downward rank heuristic strategy to find the corresponding scheduling sequence. Finally, we apply a cost-to-time ratio improvement strategy to further improve the performance of the improved CS algorithm. Extensive experiments are conducted to evaluate the effectiveness and efficiency of our method. The results validate our approach and show its superiority in comparison with the state-of-the-art methods.

INDEX TERMS Task scheduling, DVFS, cuckoo search algorithm, heterogeneous multiprocessor system.

I. INTRODUCTION

Modern High Performance Computing (HPC) systems, such as Tianhe-2 [1] and Sunway TaihuLight [2], typically consist of heterogeneous computing components interconnected by a high speed network. Such systems are expected to be used for fast processing of computationally intensive applications with different computing needs. These applications often have certain time constraints. Because high energy consumption is a bottleneck for the deployment of HPC systems, a major research challenge for heterogeneous HPC systems is how to provide services to applications in such a way that minimizes energy consumption while satisfying the applications' time constraints.

Due to the importance of energy consumption, various techniques have been developed, such as DVFS, consolidation virtualization and duplication [3], [4]. Among them, DVFS has been shown to be a very promising technique, and has been widely used in energy-aware scheduling to make processors energy-efficient [3], [5]–[8]. DVFS reduces energy consumption by scaling down supply voltage/frequency of processors [9]. When a real-time application executes on a heterogeneous multiprocessor system with DVFS technique, it contains three phases, namely, task prioritizing, processor selection and power supplying phases [3], [10]. The task scheduling problem on heterogeneous multiprocessor systems has been proved to be NP-hard, as its time complexity grows exponentially on the choice of number for voltage settings [3], [11].

It is difficult to find an effective way to solve the above problems, because a processor has several voltage settings, and the same task has different processing times and energy consumption levels when executing on different processors. Traditional scheduling studies focus on heuristic-based algorithms, which are often based on greedy local optimal selection for some heuristic strategies [3], [12], [13]. However, due to the greedy nature, heuristic-based methods can not always produce consistent results for different

The associate editor coordinating the review of this manuscript and approving it for publication was Alberto Cano.

problem instances [3], [12]. Because of the high adaptability, many well-known meta-heuristic algorithms have been adopted, including Genetic Algorithms (GA) [12], [14]–[19], and [20], Simulated Annealing algorithms (SA) [21], [22], Quantum-inspired Hyper-heuristics Algorithms (QHA) [3], [16], Ant Colony Optimization (ACO) [23]–[26], etc. However, the search process of the meta-heuristic algorithm varies from problem to problem, and has the disadvantages of large randomness, low global search efficiency, and premature convergence in the late iteration.

Although there have been many studies on task scheduling, energy-aware task scheduling using DVFS technique still faces many challenges. First of all, due to its greedy nature, existing heuristic algorithms are unable to obtain a consistently good scheduling scheme in complex situations [27]. Secondly, many existing random search algorithms have high time complexity and low search efficiency, and their search performance needs to be improved [3]. Since each scheduling technique has its pros and cons, and different techniques may complement each other, hybrid algorithms as an effective way to improve algorithm performance appeared. The Cuckoo Search (CS) algorithm, proposed by Yang and Deb in 2009, can solve the optimization problem by simulating the behavior of brood-parasitism and lévy flights [28], [29]. It has the characteristics of simple structure, fast search speed, and few parameters, and some studies have shown that the CS algorithm is more efficient than some swarm intelligence algorithms such as GA, the Artificial Bee Colony (ABC) algorithm and the Particle Swarm Optimization (PSO) algorithm, etc., [28]–[37]. CS has been widely used for solving optimization problems in engineering applications, thus using it to search task graph scheduling is expected to improve the scheduling quality and shorten the search speed. Therefore, in this paper, we propose an improved cuckoo search algorithm combined with a heuristic modification strategy. By combining these algorithms, we can maintain their complementary advantages and achieve better universality.

The standard CS algorithm can easily fall into local optimum when solving complex problems, and has the disadvantages of low solution accuracy [31]–[34]. In order to overcome this shortcoming, we propose a Cuckoo Search algorithm based on Gaussian random walk and Adaptive discovery probability (GACS). We use chaos variables to initialize populations to maintain population diversity, a Gaussian random walk strategy to balance the exploration and exploitation capabilities of the algorithm, and an adaptive discovery probability strategy to improve population diversity. In this paper, we apply the GACS algorithm to assign tasks to the processors and their voltage states, and then use a widely used downward rank heuristic to find the corresponding scheduling sequence, and a cost-to-time ratio heuristic strategy to further improve the performance of the GACS.

The four main contributions of this paper are listed below.

(1) We propose an improved cuckoo search algorithm deploying Gaussian random walk and adaptive discovery probability, which can effectively balance exploration and exploitation capabilities of the CS algorithm.

(2) We use an Adaptive Fitness Transformation (AFT) method to solve the performance-constrained energy optimization. As far as we know, this is the first time that AFT method is applied to task scheduling problem.

(3) We propose a improvement strategy on cost-to-time ratio to improve the performance of the GACS algorithm, which can further reduce energy consumption under performance constraint.

(4) The simulation results reveal that our algorithm has better performance compared with the state-of-the-art algorithms.

In this work, we propose the GACSM algorithm to study energy-aware task scheduling problem with DVFS. The goal of our task scheduling problem is to allocate tasks to available processors to meet the precedence constraints of these tasks, so as to minimize energy consumption under certain time constraints. The difference between GACSM and other algorithms is that our algorithm combines a heuristic modification algorithm with the improved CS algorithm, and we use single population strategy and AFT method. We propose the GACSM algorithm, which utilizes a chaotic search strategy, Gaussian random walk strategy and adaptive discovery probability strategy, and combines with the cost-to-time ratio modification strategy, to minimize energy consumption under a time constraint for task scheduling on heterogeneous computing systems with DVFS. The average complete computing time of our algorithm is shorter than two advanced algorithms with respect to different graph sizes under 1000 evaluations for 30 runs. We perform extensive experiments using real-world graphs and 18 randomly generated graphs. The results verify that our algorithm has good search accuracy and search efficiency, and is superior to the state-of-art algorithms.

The remainder of this paper is organized as follows. Section 2 reviews some existing related studies on task scheduling on heterogeneous systems. Section 3 describes the model of heterogeneous systems. Section 4 presents our GACSM algorithm. Section 5 reports our experiment results. Section 6 concludes the paper.

II. RELATED WORK

Static task scheduling of applications on multiprocessors has been widely studied [5], [38]. The proposed scheduling algorithms can be classified as heuristic-based and meta-heuristic. Heuristic-based scheduling algorithms typically find a scheduling scheme in polynomial time based on incomplete information [39]–[44]. Topcuoglu et al. in [38] proposed two classical algorithms: Heterogeneous Earliest Finish Time (HEFT) and Critical Path On a Processor (CPOP). Metaheuristic scheduling algorithms usually use the technique of random search [45]–[48]. Metaheuristic algorithm usually generates schedules of better quality than that of heuristic-based algorithm; however, due to the low search

TABLE 1. Summary of energy-aware task scheduling with DVFS.

References	Optimization Goal	Architecture	Task	Speed Set	Method
Chen et al. [3]	Makespan or Energy Consumption	Heterogeneous	Dependant	Discrete	Metaheuristic
Lee and Zomaya [5]	Makespan & Energy Consumption	Heterogeneous	Dependant	Discrete	Heuristic
Xie et al. [7]	Energy Consumption	Heterogeneous	Dependant	Discrete	Heuristic
Li [49]	Makespan or Energy Consumption	Homogeneous	Independent	Discrete	Heuristic
Zhang et al. [8], [50]	Energy Consumption	Heterogeneous	Dependent	Continuous	Linear Programming
Nesmachnow et al. [51]	Makespan & Energy Consumption	Heterogeneous	Independent	Discrete	Heuristic
Li et al. [52]	Weighted probability	Heterogeneous	Independent	Discrete	Heuristic
Mezmaz [53]	Makespan & Energy Consumption	Heterogeneous	Dependant	Discrete	Metaheuristic
Mashayekhy [54]	Energy Consumption	Heterogeneous	Dependant	Discrete	Heuristic
Kang and Ranka [55]	Energy Consumption	Heterogeneous	Dependant	Continuous	Linear Programming
Hu et al. [27]	Energy Consumption	Heterogeneous	Dependant	Discrete	Heuristic
Huang et al. [56]	Energy Consumption	Heterogeneous	Dependant	Discrete	Heuristic
Tang et al. [57]	Energy Consumption	Heterogeneous	Dependant	Discrete	Heuristic
Li [58]	Makespan or Energy Consumption	Homogeneous	Independent	Discrete	Heuristic
Xiao et al. [59]	Reliability	Heterogeneous	Dependent	Discrete	Heuristic
Shekar and Izadi [60]	Makespan & Energy Consumption	Heterogeneous	Dependent	Discrete	Heuristic
Terzopoulos and Karatza [61]	Energy Consumption	Heterogeneous	Independent	Discrete	Heuristic
Zhang et al. [62]	Reliability & Energy Consumption	Heterogeneous	Dependent	Discrete	Heuristic

efficiency, its computation cost is much higher than that of heuristic-based algorithm [3], [15].

Many studies have been conducted for energy-aware task scheduling on processors with DVFS (see Table 1). Most of them either focused on homogeneous computing systems [49] and independent task scheduling [49], [51], [52], [58], [61], or have very high computational cost [3], [53], [54]. For the continuous DVFS situation, there are some studies that considered reducing energy consumption [8], [50], [55]. However, since many scheduling problems are discrete in reality, energy-aware task scheduling becomes quite complex in this case. For the discrete DVFS situation, the authors in [3], [5], [7], [27], [49], [51]–[54], [56]–[62] investigated the scheduling problems. However, some of them adopted the strategy of shutting down the processors in the system [56], [57], which is unreasonable in reality [7]. Lee and Zomaya in [5] proposed an Energy-Conscious Scheduling (ECS) algorithm, and the authors in [27] proposed an Energy Aware task scheduling in the context of Service Level Agreement (EASLA). These approaches are mainly based on heuristic methods which are not agile for different application situations [7], [27]. The work [3] proposed a quantum-inspired hyper-heuristics algorithm (QHA), but its computational cost is too high and its scheduling results may violate the precedence constraint of the tasks [63].

The authors in [26] proposed an Improved Multi-Population Co-evolution Ant Colony Optimization (ICMPACO) algorithm, which is based on the multi-population strategy, co-evolution mechanism, pheromone updating strategy and pheromone diffusion mechanism. The ICMPACO algorithm uses a positive feedback mechanism, which is different from our GACS algorithm. In the ICMPACO algorithm, each individual can only perceive local information and cannot directly use global information, while our GACS algorithm can share information through the current optimal individual. In this work, we address the performance-constrained energy optimization problem for task scheduling on heterogeneous computing systems with DVFS by combining the GACS algorithm and a cost-to-time ratio modification strategy.

III. THE MODELS

In this section, we discuss the mathematical models of heterogeneous multiprocessor systems with dynamically variable voltage. We assume that the heterogeneous multiprocessor system in this work has the following characteristics [12]: (1) non-preemptive; (2) fully interconnected network; (3) task duplication is prohibited; (4) communication links with different startup time and bandwidth; (5) each processor has an independent I/O unit that allows for communication and computation to be performed simultaneously [9], [12].

A. SYSTEM MODEL

We assume that the system consists of a set of heterogeneous processors $P = \{P_1, P_2, \cdots, P_M\}$ that are fully interconnected by a high-speed network, where M represents the number of heterogeneous processors and each processor $P_j \in P$ is DVFS-enabled with a finite numer of $h(k)$ different voltage supply levels [3]. Let $V_k = (V_{k1}, \cdots, V_{kh(k)})$ be the voltage supply vector of P_k, where V_{kr} is the voltage corresponding to the rth Voltage Supply Level (VSL) of processor P_k. Specially, we denote P_{kr} as the processor P_k under supply voltage V_{kr}. When processor P_k is idle, its supplied voltage $V_{kh(k)}$ is minimal [3].

B. APPLICATION MODEL

Let a task graph $G = (T, E)$ be a Directed Acyclic Graph (DAG) composed of a set of tasks $T = \{T_1, \cdots, T_N\}$, where vertex set T represents tasks, edge set E represents execution precedences among tasks, and N is the number of tasks. We assume that each task can only be executed sequentially without preemption in the same processor. There is an entry task and an exit task in a DAG. The vertex weight, denoted as $D_w(T_i)$, which represents the computation amount of task T_i. Each edge $e_{ij} \in E$ represents a precedence constraint between T_i and T_j and implies that if $T_i \rightarrow T_j$, then T_i is the predecessor of T_j and T_j is the successor of T_i [15], i.e., the output of T_i has to be transmitted to T_j before T_j start its execution [5]. The edge weight is denoted as $C_w(T_i, T_j)$, which represents the communication amount

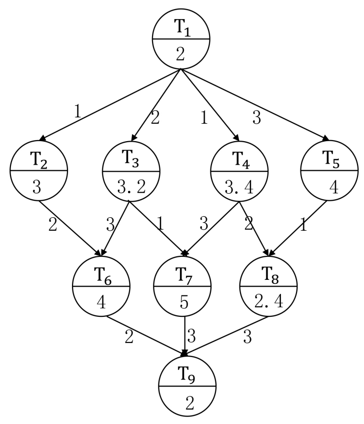


FIGURE 1. Example of DAG.

TABLE 2. Important notations in this work.

Notation	Description
N	Number of tasks.
V_{kr}	The rth voltage supply level of processor P_k.
P_{kr}	The processor P_k under supply voltage V_{kr}.
B_{ikr}	The processing time of task T_i on P_k with V_{kr}.
$C(T_i, T_j)$	The communication time from T_i to T_j.
$pr(T_i)$	The immediate predecessor task set of task T_i.
$Ms(G)$	The makespan of G.
$E(G)$	The total power consumption of a task graph G.
$Rk(T_i)$	The downward rank of a task T_i.
$DiE(T_i, P_{kr})$	The increased energy consumption when task T_i is moved from the currently assigned processor to P_{kr}.
$DiT(T_i, P_{kr})$	The increased execution time when task T_i is moved from the currently assigned processor to P_{kr}.
$Ra(T_i, P_{kr})$	The cost-to-time function when task T_i is moved from the currently assigned processor to P_{kr}.
Np	The population size.
GACS	Cuckoo Search algorithm based on Gaussian random walk and Adaptive discovery probability.
EASLA	Energy Aware task scheduling in the context of Service Level Agreement [27].
ICMPACO	The Improved Multi-Population Co-evolution Ant Colony Optimization algorithm [26].
GACSM	GACS algorithm combined with the cost-to-time ratio Modification strategy.
EASLAM	EASLA algorithm combined with the cost-to-time ratio Modification strategy.
ICMPACOM	ICMPACO algorithm combined with the cost-to-time ratio Modification strategy.
QHAM	QHA algorithm combined with the cost-to-time ratio Modification strategy.
EASLAM-HEFT	EASLAM simulation constrained by HEFT.
ICMPACOM-HEFT	ICMPACOM simulation constrained by HEFT.
QHAM-HEFT	QHAM simulation constrained by HEFT.
GACSM-HEFT	GACSM simulation constrained by HEFT.
EASLAM-ECS	EASLAM simulation constrained by ECS.
ICMPACOM-ECS	ICMPACOM simulation constrained by ECS.
QHAM-ECS	QHAM simulation constrained by ECS.
GACSM-ECS	GACSM simulation constrained by ECS.

between task T_i and T_j. An example of DAG is showed in Fig. 1, which shows a DAG of nine tasks that need to be assigned to the given number of available processors. The weight 3.2 of T_3 represents the computation amount of T_3 denoted as $D_w(T_3) = 3.2$, and the edge weight 2 between T_1 and T_3 indicates the communication amount denoted as $C_w(T_1, T_3) = 2$.

The precedence constraints of tasks are known a priori and remain unchange during scheduling and task execution. When scheduling the tasks of DAG to the processors, it is necessary to satisfy the precedence constraints among tasks and the availability of the processors.

Let B_{ikr} be the processing time of task T_i on P_k with V_{kr}, and $Sr(V_{kr})$ be the relative speed when T_i is executed on P_k with V_{kr}. Then B_{ikr} can be expressed as

$$B_{ikr} = \frac{D_w(T_i)}{Sr(V_{kr})} \tag{1}$$

The communication between tasks assigned to different processors is performed through message passing over the bus [7]. If the data that a task needs to read is available in the local memory, inter-processor communication will not occur [9]. When T_i and T_j are scheduled to the same processor, the communication time is zero as the intra-processor communication can be ignored [3], [7], [9].

If T_i and T_j are assigned to different processors, then the communication cost incurs [3], [7]. Suppose that T_i is assigned to processor P_k and T_j is assigned to processor P_l. Let $D(P_k, P_l)$ be data transfer rates between processor P_k and processor P_l, and $C_s(P_k)$ be the communication startup time of processor P_k [15]. The communication time $C(T_i, T_j)$, which represents the time spent in transferring data from T_i to T_j, is measured in seconds. Thus, $C(T_i, T_j)$ can be expressed as

$$C(T_i, T_j) = \begin{cases} 0, & \text{if } k = l, \\ C_s(P_k) + \frac{C_w(T_i,T_j)}{D(P_k,P_l)}, & \text{if } k \neq l. \end{cases} \tag{2}$$

Let $EFT(T_i)$ be the earliest finish time of task T_i on processor P_k given supply voltage V_{kr}. Then $EFT(T_i)$ is defined as

$$EFT(T_i) = EST(T_i) + B_{ikr}, \tag{3}$$

addedwhere $EST(T_i)$ represents the earliest start time of task T_i. $EST(T_i)$ can be expressed as

$$EST(T_i) = \begin{cases} 0, & \text{if } T_i = T_{en} \\ \max\{eavt(P_k), \\ \quad \max\limits_{T_j \in pr(T_i)} \{EFT(T_j) + C(T_i, T_j)\}\}, & \text{otherwise} \end{cases} \tag{4}$$

where $eavt(P_k)$ represents the earliest available time when P_k is ready for task schedule, $pr(T_i)$ is the immediate predecessor task set of task T_i, and T_{en} represents the entry task.

Let $Ms(G)$ be the makespan (scheduling length) of G. Then $Ms(G)$ is defined as [3]

$$Ms(G) = \max_{T_i \in T} EFT(T_i). \tag{5}$$

C. ENERGY CONSUMPTION MODEL

The energy consumption consumed by processors, power supply modules, memory and fans varies under different workload in high-performance computing systems [64], [65]. Some studies show that processors are the main consumers of system energy [64], [65]. In this work, we focus on the energy consumption of the processors. We assume processors are based on the Complementary Metal Oxide Semiconductor (CMOS) technology [3], [5], [66]. The power consumption is dominated by dynamic power dissipation P_d, which is defined as

$$P_d = CV^2F, \tag{6}$$

where C is the effective switched capacitance, V is the supply voltage, and F is the processor clock frequency. Since $F \propto V$, so $P_d = \lambda V^3$, where λ represents a parameter that differs with each type processor [3].

The dynamic energy consumption of all the tasks executed can be expressed as [3]

$$E_d = \sum_{k=1}^{M} \sum_{T_i \in U_k} \lambda_k V_{kr}^3 B_{ikr}, \tag{7}$$

where U_k is the task set on processor P_k. Obviously, U_k is a subset of T.

The total idle energy consumption of all the idle nodes is [3]

$$E_i = \sum_{k=1}^{M} \left(\left(Ms(G) - \sum_{T_i \in U_k} B_{ikr} \right) \lambda_k V_{kh(k)}^3 \right). \tag{8}$$

where $V_{kh(k)}$ is the minimum supply voltage on P_k.

Let $E(G)$ be the total power consumption of a task graph G, then it can be calculated as [3], [5]

$$E(G) = E_d + E_i. \tag{9}$$

D. PROBLEM MODEL

The problem of performance-constrained energy optimization we study in this paper is defined as:

$$Minimize : E(G), \tag{10}$$

subject to:

$$\begin{aligned} &EST(T_i) \geq EFT(T_j), T_j \in pr(T_i), \\ &Ms(G) \leq S. \end{aligned} \tag{11}$$

where $E(G)$ represents the total energy consumption of task graph G, and S represents the time constraint of task graph G.

IV. ALGORITHM FRAMEWORK

In this section, we will present the framework of our improved cuckoo search (GACSM) algorithm deploying Gaussian random walk and adaptive discovery probability and combined with a modification strategy.

In order to take the advantages of GACS-based and heuristic-based algorithms and avoid their disadvantages, we use an approach by combining GACS algorithm and heuristics. In this paper, we apply the GACS algorithm to assign task to the processor and its voltage state. In our algorithm, after obtaining the task-to-resource mapping scheme, we use a widely used downward rank heuristic to calculate the task priority according to the mapping results, and then we can evaluate the fitness value $f(x_i)$ and constraint violation degree $v(x_i)$.

Firstly, we call the chaos method to create an initial population P^0 (line 2). Secondly, if the random number $r_2 \leq rank(i)/Np$, where $rank(i)$ represents the order in which the individual x_i^g is in the population according to the fitness value from small to large and Np represents the population size, then we use the Gaussian random walk strategy to balance the exploration and exploitation capabilities of the algorithm by Eq.(17) (line 6-8). Then, we select Np better individuals in population P, P' (line 9). Thirdly, in line 10-16, we perform the CS operator. Among the CS operator, we abandon a fraction P_a of worst nests by using Eq.(19) and build new ones via Eq.(20) (line 15).

The loop iterates until the stopping criterion is reached. After performing GACS, we use the cost-to-time ratio strategy to improve its performance (line 17). The outline of GACSM is depicted in Algorithm 1.

Algorithm 1 GACSM

```
Require: Parameters for GACSM and task scheduling.
Ensure: A task schedule.
 1: $g = 0$;
 2: Call Algorithm 2 to create an initial population $P^0$;
 3: repeat
 4:     $g = g + 1$;
 5:     $P^g = P^{g-1}$;
 6:     if the random number $r_2 \leq rank(i)/Np$ then
 7:         Generate new individuals by using Eq.(17) and
            obtain new population $P'^{g-1}$;
 8:     end if
 9:     $P^g$=ChooseBestIndividual($P^{g-1}, P'^{g-1}$)
    (Algorithm 3);
10:     Get cuckoo with eggs randomly by lévy flights;
11:     Choose nest $j$ randomly among $P^g$;
12:     if $x_k$ is better than $x_j$ then
13:         replace $x_j$ by the new individual $x_k$;
14:     end if
15:     Abandon a fraction $P_a$ of worst nests by using Eq.(19)
        and build new ones via Eq.(20);
16:     Obtain new population $P^g_{new}$;
17:     $P^g = P^g_{new}$;
18: until the stopping criterion is reached;
19: Call modification strategy to further improve the popula-
    tion $P^g$(Algorithm 4);
20: return the best solution of schedule.
```

A. CUCKOO SEARCH

The CS algorithm is an emerging biological heuristic algorithm proposed by Yang and Deb in 2009 which simulates the brood parasitism behavior of cuckoos. Because of its simplicity and easy implementation, CS has been successfully applied to solving practical problems such as engineering optimization, and widely accepted in the field of intelligent algorithms [32]–[37]. Its main idea is below:

When generating a new solution x_i^{g+1}, a lévy flight is performed as follows

$$x_i^{g+1} = x_i^g + \alpha \oplus \text{Lévy}(\beta), \tag{12}$$

where α represents the step size, x_i^{g+1} the next generation solution, x_i^g the current generation solution, and product $\oplus$ the entry-wise multiplications. Lévy(β) represents the lévy random number. For the convenience of calculation, the literature [29] uses the Eq.(13) to calculate the lévy random number

$$\text{Lévy}(\beta) \sim \frac{\mu}{||\nu||^{1/\beta}}, \tag{13}$$

where μ and ν are the random numbers of normal distributions satisfying the following conditions:

$$\theta_\mu^2 = [\frac{\Gamma(1+\beta)\cdot sin(\pi\beta/2)}{\Gamma((1+\beta)/2)\cdot\beta\cdot 2^{(\beta-1)/2}}]^{1/\beta},\ \theta_\nu^2 = 1$$

CS discards some inferior solutions by a discover probability P_a, and then regenerates the same number of new solutions by using preference random walks:

$$x_{i,d}^{g+1} = x_{i,d}^g + r_1(x_{j,d}^g - x_{k,d}^g) \tag{14}$$

where $x_{j,d}^g$, $x_{k,d}^g$ represents two randomly selected solution, and r_1 is a uniformly distributed random number in the interval (0,1).

B. CUCKOO SEARCH ALGORITHM BASED ON GAUSSIAN RANDOM WALK AND ADAPTIVE DISCOVER PROBABILITY (GACS)

1) CHAOS METHOD

We use the chaos method to initialize the population. The nature of chaos is random, unpredictable, and regular. Searching by chaos method can make the algorithm jump out of local optimum, maintain population diversity, and improve global search ability [67]. In this paper, an ergodic chaos mapping is introduced to transform the initial variables into chaos variables. The sinusoidal iteration formula is adopted as follows

$$cf_{k+1}^j = sin(cf_k^j\pi), \tag{15}$$

where cf_k^j is a randomly generated number of interval (0,1), $j = 1, \cdots, N; k = 0, 1, \cdots, MaxCh$, $MaxCh$ is the maximum numbers of chaotic iterations, and N is the number of tasks.

The sinusoidal iteration formula Eq.(15) is introduced into the process of population initialization, and the population variable transformation formula is as follows

$$x_i^j = x_{\min}^j + cf_k^j(x_{\max}^j - x_{\min}^j), \tag{16}$$

where $x_{\min}^j$ and $x_{\max}^j$ are the lower and upper limits of the jth dimension variable, respectively.

2) GAUSSIAN RANDOM WALK STRATEGY

Since Gaussian random walk strategy has strong local exploitation ability [68], [69], we use this strategy to generate a new random population, which can balance the global exploration and local exploitation ability of the algorithm. We use individual fitness values to determine individual performance. Let $rank(i)$ be the order in which the individual x_i^g is in the population according to the fitness value from small to large, Np be the population size, and r_2 be a random number of interval [0, 1]. If the random number $r_2 \leq rank(i)/Np$, then x_i^{g+1} is operated as follows

Algorithm 2 Chaos Method

```
for $i = 1$ to $Np$ do
    for $j = 1$ to $N$ do
        Randomly generate $cf_0^j$ in the interval (0, 1);
        for $k = 1$ to $MaxCh$ do
            $cf_k^j = sin(cf_{k-1}^j\pi)$
        end for
        $x_i^j = x_{\min}^j + cf_k^j(x_{\max}^j - x_{\min}^j)$;
    end for
end for
return the generated $Np$ individuals as the initial population.
```

$$x_i^{g+1} = Gaussian(x_b^g, \xi) + r_3\cdot(x_b^g - x_i^g) \tag{17}$$

where x_i^g is the ith candidate solution in the population, and x_b^g is the best solution. r_3 is a random number of interval [0, 1], and ζ is defined as

$$\xi = \frac{1}{g^{3/5}}\cdot(x_b^g - x_i^g) \tag{18}$$

Using the best individual to guide the poor individual can help the poor individual to move toward the best individual, which can speed up the convergence of the algorithm. This strategy mainly operates on poor individuals with a high probability, which increases the efficiency of algorithm evolution. In addition, the Gaussian distribution is controlled by the adaptively adjusted variance ξ. In the early stage of the algorithm, the value of the variance ξ is large, which helps maintain the global exploration ability of the algorithm; the value of the variance ξ decreases with the increase of the number of iterations g, which helps to improve the local exploitation ability of the algorithm.

3) ADAPTIVE DISCOVER PROBABILITY

The CS algorithm discards some worse nests at a probability P_a, and continue searching from the rest. It determines a suitable probability P_a. If P_a is too small, it is difficult to generate new individuals. If P_a is too large, the algorithm will become a pure random search algorithm. Therefore, the convergence of the CS algorithm is affected by choosing an appropriate P_a. In the standard CS algorithm, the value of P_a is usually equal to a constant number. Intuitively, a fixed P_a is likely to reduce the convergence performance of the algorithm. To overcome this problem, We use the following dynamic adaptive mechanism to adjust the discover probability P_a:

$$P_a = (P_{max} - P_{min})\left(\frac{f_i - f_{min}}{f_{max} - f_{min}}\right)^2 + P_{min}, \tag{19}$$

TABLE 3. Voltage-relative speed pairs.

level	pair1			pair2			pair3			pair4		
	Index	Voltage	Relative speed(%)	Index	Voltage	Relative speed(%)	Index	Voltage	Relative speed(%)	Index	Voltage	Relative speed(%)
0	1	1.75	100	4	1.5	100	10	2.2	100	14	1.5	100
1	2	1.4	80	5	1.4	90	11	1.9	85	15	1.2	80
2	3	1.2	60	6	1.3	80	12	1.6	65		0.9	50
3		0.9	40	7	1.2	70	13	1.3	50			
4				8	1.1	60		1.0	35			
5				9	1.0	50						
6					0.9	40						

Algorithm 3 ChooseBestIndividual(P, P')

```
Calculate fitness value $f(x_i)$ and constraint violation degree $v(x_i)$ of each individual in $P$;
Calculate the proportion $\phi_1$ of feasible solutions in $P$;
Calculate the transformed fitness value $f_{fit}(x_i)$ by AFT method according to $\phi_1$;
Calculate fitness value $f(x_i')$ and constraint violation degree $v(x_i')$ of each individual in $P'$;
Calculate the proportion $\phi_2$ of feasible solutions in $P'$;
Calculate the transformed fitness value $f_{fit}(x_i')$ by AFT method according to $\phi_2$;
if $f_{fit}(P') < f_{fit}(P)$ then
    return $P'$;
else
    return $P$;
end if
```

where f_i represents the fitness of the the current solution x_i, f_{min} is the minimum fitness of all solutions, and f_{max} is the maximum fitness of all solutions. P_{max}, P_{min} are two parameters in the interval (0,1).

It can be seen from Eq.(19) that the closer the solution is to the optimal solution, the smaller the P_a is, which makes the solution more likely to be retained to the next generation. When the difference between the fitness of the current solution and the optimal solution is large, the P_a is large, which makes the solution to be discarded easily.

Let r_4, r_5 be a random number in the interval [0,1]. If $r_4 \leq P_a$, then the individual $x_i^{\prime g+1}$ is operated as follows:

$$x_i^{\prime g+1} = \begin{cases} x_i^{g+1} + (r_6 x_j^{g+1} - r_7 x_k^{g+1}), & r_5 \leq 0.5 \\ x_i^{g+1} + r_8(x_b^{g+1} - x_j^{g+1}), & \text{otherwise.} \end{cases} \tag{20}$$

where x_b^{g+1} is the best individual, x_j^{g+1}, x_k^{g+1} are two randomly selected different individual, r_6, r_7 are two uniformly distributed random number in the interval [0,1], and x_8 are a random number of interval [0, 1]. It can be seen that through individual screening strategy, individuals with poor fitness are more likely to be discarded, and new individuals are generated according to Eq.(19). At the same time, Eq.(20) uses two different types of mutation operators, namely random search and mutation operator of optimal individuals, in order to enhance the exploratory ability of the algorithm while improving its development ability.

T1	T2	T3	T4	T5	T6	T7	T8	T9
4. 1	2. 8	6. 2	10. 9	14. 3	1. 7	10. 1	15. 2	3. 9

FIGURE 2. Nest's position.

C. ENCODING OF SOLUTIONS

We first show the priority queues for DAG applications, and then present the encoding mechanism of task scheduling.

1) TASK PRIORITY CALCULATION

We use a widely used downward rank heuristic to calculate task priority by strategy [15]. Its definition is below:

Let $Rk(T_i)$ be the downward rank of a task T_i on P_k given supply voltage V_{kr}, then $Rk(T_i)$ can be defined by Eq.(21):

$$Rk(T_i) = \max_{T_j \in pr(T_i)} (B_{ikr} + C(T_i, T_j) + Rk(T_j)), \tag{21}$$

where $pr(T_i)$ denotes the set of immediate predecessors of task T_i.

2) NEST REPRESENTATION

For mapping tasks to resources, we first divide the voltage supply levels of a processor into non-idle and idle voltage supply levels. Then we encode the non-idle voltages supply levels of all processors in turn. Each processor has several non-idle voltages supply levels, and each non-idle voltage supply level corresponds to a unique index number (see Table 3). Finally, we can determine the corresponding processor based on the index of the non-idle voltage supply level, and use Eq.(8) to calculate the total energy consumption of all the idle nodes. The encoding of solutions is chosen randomly from 1 to N_{tv}, where N_{tv} is the total number of the non-idle voltage supply levels.

Cuckoo search works on the problem with continuous space, but the problem of graph scheduling is a problem of discrete space, so we need to discretize the space. In our algorithm, the dimension of individual $x_i = (x_{i1}, \cdots, x_{iN})$ is N, which is consistent with the tasks number of DAG. If there are N_{tv} non-idle voltage states, each task can be assigned to voltage states in the range of $1, \cdots, N_{tv}$. We set $x_{ij}(j = 1, \cdots, N)$ in the range $(0.5, N_{tv} + 0.5)$, and then rounded the x_{ij} value to the nearest whole number. For example, the value of 10.9 in the fourth dimension in Fig. 2 indicates that task T_4 is assigned to a processor of pair3 with the non-idle voltage

T1	T2	T3	T4	T5	T6	T7	T8	T9
l4	l3	l6	l11	l14	l2	l10	l15	l4

FIGURE 3. Task to resource mapping.

index of 11 (see Fig. 3), i.e., task T_4 is executed on a processor with a voltage of 1.9 and a relative speed of 0.85.

D. CONSTRAINT HANDLING STRATEGY

We apply constraint optimization for the GACS search process. Constrained optimization problems are usually expressed as follows:

$$\begin{aligned} & min\, f(x) \\ & s.t.\; g_j(x) \le 0, \quad j = 1, \cdots, q \\ & \qquad h_k(x) = 0, \quad k = q+1, \cdots, p. \end{aligned} \tag{22}$$

where $x \in \omega \subseteq Sp$ represents the decision vector, ω represents the feasible area, and Sp represents the search space.

Usually, this constraint is transformed into the following inequality constraint:

$$|h_k(x)| - \eta \le 0, k = q+1, \cdots, p, \tag{23}$$

where η represents tolerance factor, and it usually greater than 0. Then the degree of constraint violation of an individual on jth constraint can be evaluate as

$$G_j(x) = \begin{cases} \max\{0, g_j(x)\}, & j = 1, \cdots, q \\ \max\{0, |h_k(x)| - \eta\}, & k = q+1, \cdots, p \end{cases} \tag{24}$$

Then, the total degree of standardization constraint violation $v(x)$ of individual x can be calculate as

$$v(x) = \sum_{j=0}^{p} G_j(x). \tag{25}$$

In order to deal with the constrained optimization problem, the authors proposed an Adaptive Fitness Transformation (AFT) method to divide the population into three states: infeasible state, semi-feasible state and feasible state [70].

(1) Infeasible state: In the infeasible state, the population only contains infeasible solutions. In this case, only the degree of constraint violation needs to be considered, and its fitness value can be calculated as follows [70]

$$f_{fit}(x_i) = v(x_i) \tag{26}$$

(2) Semi-feasible state: In the semi-feasible state, the population contains not only several feasible solutions but also some infeasible solutions. In this case, the population is divided into feasible solution set (W_1) and infeasible solution set (W_2). Therefore, the objective function value $f'(x_{i,g})$ of solution $x_{i,g}$ can be converted as [70]

$$f'(x_i^g) = \begin{cases} f(x_i^g), & i \in W_1, \\ \max\{\phi \times f(x_b^g) + (1-\phi) \times f(x_w^g), \\ f(x_i^g)\}, & i \in W_2, \end{cases} \tag{27}$$

where ϕ is the feasible solution ratio of the previous generation population, and x_b^g, x_w^g represent the best and worst solution of feasible solution set W_1, respectively.

Eq.(27) can be normalized as

$$f_{nor}(x_i) = \frac{f'(x_i) - \min\limits_{j \in W_1 \cup W_2} f'(x_j)}{\max\limits_{j \in W_1 \cup W_2} f'(x_j) - \min\limits_{j \in W_1 \cup W_2} f'(x_j)} \tag{28}$$

The degree of constraint violation can be calculated by Eq.(25), then Eq.(25) is normalized as

$$G_{nor}(x_i) = \begin{cases} 0, & i \in W_1, \\ \dfrac{G(x_i) - \min\limits_{j \in W_2} G(x_j)}{\max\limits_{j \in W_2} G(x_j) - \min\limits_{j \in W_2} G(x_j)}, & i \in W_2, \end{cases} \tag{29}$$

Therefore, the fitness value $f_{fit}(x_i)$ can be expressed as

$$f_{fit}(x_i) = f_{nor}(x_i) + g_{nor}(x_i) \tag{30}$$

(3) Feasible state: In the feasible state, all individuals in the population are feasible solutions. At this time, the fitness value can be calculated as follows [70]

$$f_{fit}(x_i) = f(x_i) \tag{31}$$

E. MODIFICATION STRATEGY

Inspired by the literature [1], we propose an improved cost-to-time ratio modification strategy to further reduce energy consumption under time constraint.

The cost-to-time function $Ra(T_i, P_{kr})$ can be defined as follows [1]:

$$Ra(T_i, P_{kr}) = \frac{DiE(T_i, P_{kr})}{DiT(T_i, P_{kr})}, \tag{32}$$

where $DiE(T_i, P_{kr})$ and $DiT(T_i, P_{kr})$ represent respectively the increased energy consumption and execution time when task T_i is moved from the currently assigned processor to P_{kr} [1].

Our strategy works as follows:

- Compute a critical path $cp : T_i \rightsquigarrow T_j$ in G. Apparently, the computation time of the critical path (CP) cp is equal to the makespan of G.
- If $T(G) > S$, we re-allocate the processor of a task selected from the CP to reduce the makespan. In order to obtain the minimal energy consumption and meet the time constraint, if P_{jr} is a new processor for T_i and $DiT(T_i, P_{jr}) < 0$, we select a task with the maximal ratio $Ra(T_i, P_{kl})$ and move it to a processor P_{kl}. Since the increased execution time is negative, for the same amount of reduced execution time, a larger ratio means a smaller increase of energy. For example, assume $DiE(T_i, P_{jr}) = 10$, $DiE(T_i, P_{kr}) = 4$, and $DiT(T_i, P_{jr}) = DiT(T_i, P_{kr}) = -2$, then according to Eq.(32), we can see that $Ra(T_i, P_{jr}) = -5$, and $Ra(T_i, P_{kr}) = -2$. Obviously, in this case, it is better to reassign Ti to resource P_{kr}. After the task assignment adjustment, the algorithm attempts to find a new CP in G

and tries to reduce the completion time until the time constraint is met, or the makespan G cannot be reduced any more.

- If $T(G) \leq S$, we try to reduce the energy consumption by moving a task with the minimum ratio to a new processor and voltage index. In order to reduce energy consumption, the task reassignment must satisfy $DiE(T_i, P_{jr}) < 0$. If there exist $Ra(T_i, P_{jr}) > 0$, which means $DiT(T_i, P_{jr}) < 0$, then we give priority to assign P_{kl} with the smallest positive ratio in CP to T_i. For example, assume $DiE(T_i, P_{jr}) = DiE(T_i, P_{kr}) = -10$, and $DiT(T_i, P_{jr}) = -5$, $DiT(T_i, P_{kr}) = -2$, then according to Eq.(32), we can see that $Ra(T_i, P_{jr}) = 2$, and $Ra(T_i, P_{kr}) = 5$. Obviously, in this case, it is better to reassign Ti to resource P_{jr}. Else if all $Ra(T_i, P_{jr}) \leq 0$, which means $DiT(T_i, P_{jr}) > 0$, then we assign P_{kl} with the smallest negative ratio in CP to T_i. After reassigning a node, the algorithm attempts to find another node and continues this attempt until the energy consumption can no longer be reduced.

The modification strategy iterates for each idle processor for to assign a task with the minimum energy consumption within the time constraint.

The description of the further improvement is depicted in Algorithm 3.

F. TIME AND SPACE COMPLEXITY

We analyze the time complexity of GACSM. It takes $O(Np \times N \times MaxCh)$ time to perform chaotic initialization. In each iteration of Algorithm 1, it needs to perform GACS operation. It takes $O(e \times N_{tv})$ time to evaluate the fitness function, where e, N_{tv} are the number of edges, the total number of the non-idle voltage supply levels, respectively. The time complexity of the modification strategy is $O(N^2 \times N_{tv}^3 \times e + N^2 \times N_{tv}^3 \times e)$. Thus, the time complex of the GACSM algorithm can be calculated as

$$\begin{aligned} O((\text{GACSM}) &= O(N + e \times N_{tv} + N + e \times N_{tv} + N \\ &\quad + e \times N_{tv} + N + e \times N_{tv} + N) \times Np \times Gen \\ &\quad + Np \times N \times MaxCh + 2N^2 \times N_{tv}^3 \times e) \\ &= O(e \times N_{tv} \times Np \times Gen + 2N^2 \times N_{tv}^3 \times e) \end{aligned} \tag{33}$$

where Gen represents the maximum generation.

The space complexity of GACSM is $O(Np \times N)$, because we need an array of size N to store each nest and there are at most Np nests.

V. SIMULATION AND RESULTS

A. EXPERIMENT SETUP

In the simulation environment, the target system comprises a set of completely interconnected heterogeneous processors which are DVFS-enabled. In our experiment, processors are uniformly distributed among four different sets of voltage supply levels, which are listed in Table 3. The parameter λ_k of processor P_k is set the same as [51].

Algorithm 4 Modification Strategy

```
if $T(G) > S$ then
  repeat
    find a CP $cp$ in $G$;
    $T_{cp} \leftarrow$ all tasks in $cp$;
    for each $T_i \in T_{cp}$ do
      for each $P_{jr} \in P$ do
        if $P_{jr}$ is a new index for $T_i$ and
        $DiT(T_i, P_{jr}) < 0$ then
          calculate $Ra(T_i, P_{jr})$;
        end if
      end for
    end for
    $Ra(T_i, P_{kl}) \leftarrow$ the maximal ratio in $cp$;
    Assign $P_{kl}$ to $T_i$;
  until $T(G) \leq S$
else
  repeat
    for each $T_i \in G$ do
      for each $P_{jr} \in P$ do
        if $P_{jr}$ is an available index for task $T_i$ and
         $DiE(T_i, P_{jr}) < 0$ and $T(G) \leq S$ then
          calculate $Ra(T_i, P_{jr})$;
        end if
      end for
    end for
    if there exist $Ra(T_i, P_{jr}) > 0$ then
      $Ra(T_i, P_{kl}) \leftarrow$ the smallest positive ratio in CP;
      Assign $P_{kl}$ to $T_i$;
    else
      $Ra(T_i, P_{jr}) \leftarrow$ the smallest negative ratio in CP;
      Assign $P_{kl}$ to $T_i$;
    end if
  until $E(G)$ cannot be reduced
end if
```

We use two sets of graphs to evaluate the algorithms. The first test set is the Modified Molecular Dynamics Code (MMDC) [3]. The second test set is randomly generated task graphs.

The parameters of the random graph generator are set the same as [3]. The graph height of a random DAG is calculated by a uniform distribution with a mean value of $\frac{\sqrt{N}}{\psi}$, where N represents the number of tasks in the DAG [3], and ψ represents a parallelism factor. Let $\overline{D_i}$ be the mean computation amount of task T_i. $\overline{D_i}$ is generated randomly with a uniform distribution of $[0, 2 \times \overline{D_G}]$, where D_G is the average computation amount of the given DAG. The computation amount of task T_i, i.e., $D_w(T_i)$, is in the range $[\overline{D_i} \times (1 - \frac{\delta}{2}), \overline{D_i} \times (1 + \frac{\delta}{2})]$, where δ is the computation capacity heterogeneity factor. Then, the processing time of task T_i on P_k with V_{kr}, i.e., B_{ikr}, is calculated as $B_{ikr} = \frac{D_w(T_i)}{Rs(V_{kr})}$. The communication time among tasks is generated with a uniform distribution $[0, 2 \times \overline{D_G} \times CCR]$, where CCR is the ratio of communication to computation [3].

All simulations are performed on the PC with an Intel Core i7-3770 3.40 GHz CPU and 12.0 GB RAM. The experimental tool is Python 2.7.

B. COMPARISON METRICS

Energy Consumption Ratio (ECR) is an important comparison metric. The ECR value of an algorithm is defined as

$$ECR = \frac{E}{\sum_{i=1}^{n} \min_{P_k \in P}(\lambda_k \times B_{ikh(k)} \times V_{kh(k)}^3)}, \tag{34}$$

where E represents the energy consumption of an algorithm with DVFS, λ_k represents the parameter of processor P_k, $V_{kh(k)}$ is minimal voltage of P_k, and $B_{ikh(k)}$ is the processing time of task T_i on P_k with $V_{kh(k)}$. It can be seen from Eq.(34) that the denominator is the lower bound of the energy consumption of a given task graph.

Energy-saving-ratio (ESR) can also be use to measure the performance of algorithms. The ESR value [27] is expressed as

$$ESR = \frac{E_{HEFT} - E}{E_{HEFT}}, \tag{35}$$

where E_{HEFT} is the energy consumption of all tasks in the HEFT algorithm [38] performed at the highest frequency. The makespan extension can be defined by: $T(G) \leq (1 + \zeta) * MS_b$, where ζ is the makespan extension rate, and MS_b is the makespan of a best effort HEFT schedule. In our experiment, we set the makespan extension rates at 0, 0.1, 0.2, 0.3, 0.4, respectively.

C. PARAMETER SETTING

The setting of parameters will greatly affect the experimental results, however, our main purpose is to illustrate the applicability of GACS to task scheduling. In this paper, the values of all experimental parameters are verified by repeated experiments or set by experience. To reduce randomness, the simulation results of our experiments are the average of 30 independent runs. We use the control variable method to discuss the influence of parameters, i.e., we first fix other parameter values, and then analyze the influence of the studied parameters on the algorithm.

In this paper, we uniformly set the number of population to 40 and the maximum number of iterations to 200. The parameters of GACS are set as follows: step size $\alpha = 0.10$, maximum discovery probability $P_{max} = 0.50$, minimum discovery probability $P_{min} = 0.20$. The ICMPACO [26] parameters, i.e., pheromone factor, heuristic factor, volatility coefficient, pheromone amount, and initial concentration, are set as 1, 5, 0.1, 100, 1.5, respectively. The QHA [3] parameters, i.e., $NumP$, SP, $stasize$ and σ, are set as 4, 10, 20 and 0.05, respectively.

D. COMPLETE COMPUTING TIME

In this section, we compare the complete computing time of our proposed GACS with two random heuristic algorithms, i.e., QHA and ICMPACO algorithms. Fig.4 depicts the average computing time of different algorithm with respect to different graph sizes under 1000 evaluations for 30 runs. As can be seen from Fig.4, the average complete computing time of GACS is faster than QHA and ICMPACO by (13.8%, 7.4%), (13.7%, 9.1%), (10.7%, 6.0%), (10.1%, 5.3%), (10.3%, 5.5%), for the tasks number of 16, 32, 64, 128, and 256, respectively. The reason is that our proposed GACS algorithm evolves more easily than QHA and ICMPACO, and has fewer parameters to adjust, thus its speed is relatively fast.

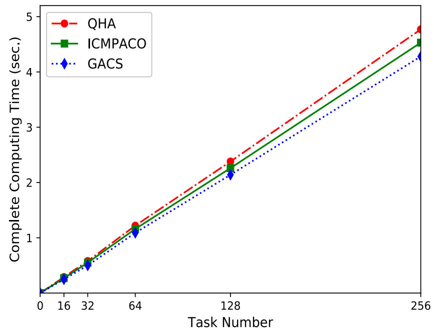


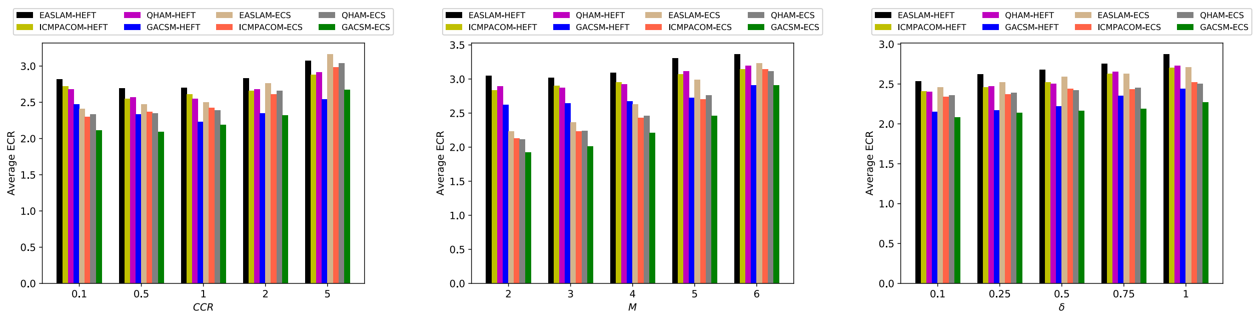
FIGURE 4. Average complete computing time of different algorithm vs. different tasks number.

E. REAL WORLD APPLICATION GRAPHS

We use application graph of real-world problem, the modified molecular dynamic code (MMDC) [3], to evaluate the performance of GACSM.

We test the search effectiveness of GACSM algorithm on MMDC problems. There are three state-of-the-art algorithms for solving performance constraint energy optimization problems, EASLA [27], ICMPACO and QHA. In order to ensure fairness, we first use the same modification strategy mentioned in Algorithm 4 to improve the performance of EASLA, ICMPACO, QHA and GACS, and denote them as EASLAM, ICMPACOM, QHAM and GACSM, respectively. We apply HEFT [38] and ECS [5] to the problem of modified molecular dynamic code, then obtain the makespan of the graph. EASLAM-HEFT, ICMPACOM-HEFT, QHAM-HEFT and GACSM-HEFT are the result of EASLAM, ICMPACOM, QHAM and GACSM simulation constrained by output of HEFT respectively, and EASLAM-ECS, ICMPACOM-ECS, QHAM-ECS and GACSM-ECS are the result of EASLA, ICMPACOM, QHAM and GACSM simulation constrained by output of ECS respectively.

The average ECR on MMDC is are shown in Fig. 5. The result of the algorithms with respect to different CCR values are shown in Fig. 5a. Our proposed GACSM algorithm is superior to EASLAM, ICMPACOM, and QHAM, where the QHAM algorithm is sometimes better or worse than ICMPACOM. The ECR values of the algorithms for different M and δ values are shown in Fig. 5b and Fig. 5c, respectively. Fig. 5 shows that our proposed GACSM algorithm outperforms EASLAM, ICMPACOM, and QHAM algorithms on



(a) (b) (c)

FIGURE 5. Average ECR of MMDC. (a) Average ECR of the algorithms vs. different *CCR* values ($M = 4$, $\delta = 0.5$). (b) Average ECR of the algorithms vs. different *M* values ($CCR = 1.0$, $\delta = 0.5$). (c) Average ECR of the algorithms vs. different δ values ($CCR = 1.0$, $M = 0.5$).

TABLE 4. Random generated instance [3].

instance	N	M	δ	CCR	ψ	S
R1	16	8	0.5	1	1	80
R2	32	8	0.5	1	1	150
R3	64	8	0.5	1	1	250
R4	128	8	0.5	1	1	300
R5	256	8	0.5	1	1	700
R6	128	8	0.5	1	0.5	600
R7	128	8	0.5	1	2	250
R8	128	8	0.1	1	1	300
R9	128	8	0.25	1	1	300
R10	128	8	0.75	1	1	300
R11	128	8	1	1	1	300
R12	128	2	0.5	1	1	800
R13	128	4	0.5	1	1	550
R14	128	16	0.5	1	1	250
R15	128	8	0.5	0.1	1	300
R16	128	8	0.5	0.5	1	300
R17	128	8	0.5	2	1	400
R18	128	8	0.5	10	1	800

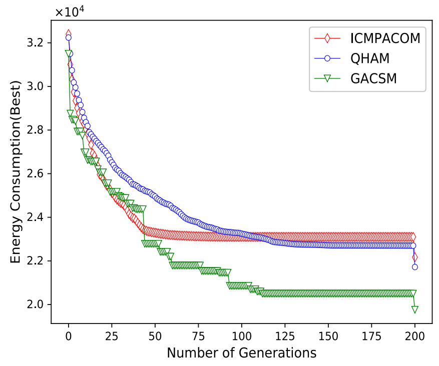


FIGURE 6. Best energy consumption simulation of R2.

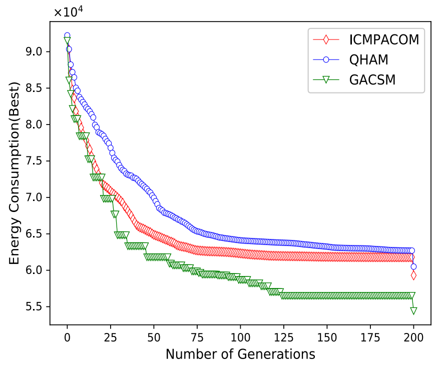


FIGURE 7. Best energy consumption simulation of R4.

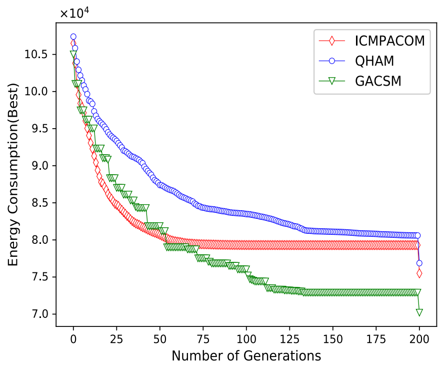


FIGURE 8. Best energy consumption simulation of R6.

the average *ECR* value by 15.0%, 10.1% and 10.4%, respectively. The result shows that the strategy of our GACSM algorithm increases the diversity of population and improves the accuracy of the algorithm effectively.

F. RANDOM GENERATED APPLICATION GRAPHS

In this section, we use 18 randomly generated DAG instances to evaluate the performance of GACSM (see Table 4). The methods and parameters used for them are same as to those used by [3]. In these instances, we consider the impact of different application graphs and the number of processors. There are three state-of-the-art algorithms for solving the performance constraint energy optimization problem, EASLA [27], ICMPACO [26] and QHA [3]. In order to ensure fairness, we first use the same modification strategy described in Algorithm 4 to improve the performance of EASLA, ICMPACO, and QHA, and denote them as EASLAM, ICMPACOM, and QHAM, respectively. We perform ICMPACO and QHA algorithms in the same number of iterations as GACSM.

TABLE 5. Result of random generated instance.

	instance	R1	R2	R3	R4	R5	R6	R7	R8	R9
ICMPACOM	Feasible rate (%)	96	100	100	100	100	100	100	100	100
	Best	3.4068	2.6726	2.4183	2.8139	2.8243	3.2594	2.7315	3.0251	2.9143
	Worst	*	3.0906	2.8125	3.3604	3.2417	3.7543	3.4927	3.6317	3.5141
	Mean	*	2.8796	2.5880	3.1113	3.0485	3.5623	3.0944	3.3592	3.2063
	std	*	4.24E-2	4.59E-2	5.31E-2	4.16E-2	6.74E-2	6.20E-2	2.85E-2	4.05E-2
QHAM	Feasible rate (%)	98	100	100	100	100	100	100	100	100
	Best	3.4623	2.6175	2.3408	2.8671	2.7829	3.3187	2.7631	3.0810	2.8703
	Worst	*	3.0524	2.8025	3.4209	3.1937	3.8540	3.4274	3.5844	3.5439
	Mean	*	2.8391	2.5530	3.1532	3.0034	3.5895	3.1307	3.3182	3.2270
	std	*	2.97E-2	3.06E-2	3.54E-2	2.75E-2	2.17E-2	4.36E-2	4.57E-2	1.53E-2
GACSM	Feasible rate (%)	**98**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**
	Best	**3.3475**	**2.3815**	**2.2194**	**2.5773**	**2.6942**	**3.0307**	**2.6576**	**2.9412**	**2.8289**
	Worst	*	**2.9321**	**2.6391**	**3.2830**	**3.1721**	**3.4755**	3.4475	**3.5427**	**3.3755**
	Mean	*	**2.6656**	**2.4285**	**2.9315**	**2.9517**	**3.2669**	**3.0584**	**3.2617**	**3.1246**
	std	*	**8.74E-1**	**7.09E-1**	**5.76E-1**	3.28E-2	5.14E-2	**2.69E-2**	3.86E-2	2.37E-2
	instance	R10	R11	R12	R13	R14	R15	R16	R17	R18
ICMPACOM	Feasible rate (%)	100	100	100	100	100	100	100	100	100
	Best	3.1813	3.3011	2.8141	2.1181	3.5349	2.4739	2.6305	2.6913	4.7013
	Worst	3.7126	3.8835	3.2356	2.5042	4.3510	2.9865	3.5122	3.5489	5.4516
	Mean	3.4467	3.5845	3.0512	2.3127	3.9441	2.7446	3.0748	3.1294	5.0907
	std	2.26E-2	2.34E-2	4.87E-2	4.63E-2	8.97E-1	4.37E-2	7.32E-1	5.39E-2	5.84E-2
QHAM	Feasible rate (%)	100	100	100	100	100	100	100	100	100
	Best	3.1033	3.2695	2.8537	2.1393	3.4917	2.4787	2.6412	2.6130	4.6452
	Worst	3.7425	3.8730	3.2172	2.5250	4.3124	3.0926	3.5569	3.5243	5.5842
	Mean	3.4374	3.5819	3.0472	2.3303	3.9203	2.7931	3.1027	3.0790	5.1071
	std	3.58E-2	6.89E-2	2.96E-2	2.09E-2	2.85E-2	2.17E-2	5.79E-2	5.79E-2	4.23E-2
GACSM	Feasible rate (%)	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**
	Best	**3.0413**	**3.1137**	**2.7019**	**2.0670**	**3.3708**	**2.3893**	**2.4816**	**2.5573**	**4.4739**
	Worst	**3.6384**	**3.6950**	**3.1342**	**2.4712**	**4.2067**	3.0412	**3.4225**	**3.4046**	**5.3202**
	Mean	**3.3379**	**3.4119**	**2.9185**	**2.2755**	**3.7793**	**2.7297**	**2.9568**	**2.9737**	**4.9074**
	std	**7.27E-1**	3.57E-2	**4.70E-1**	3.62E-2	**6.75E-1**	3.50E-2	1.79E-2	**8.05E-1**	**9.23E-1**

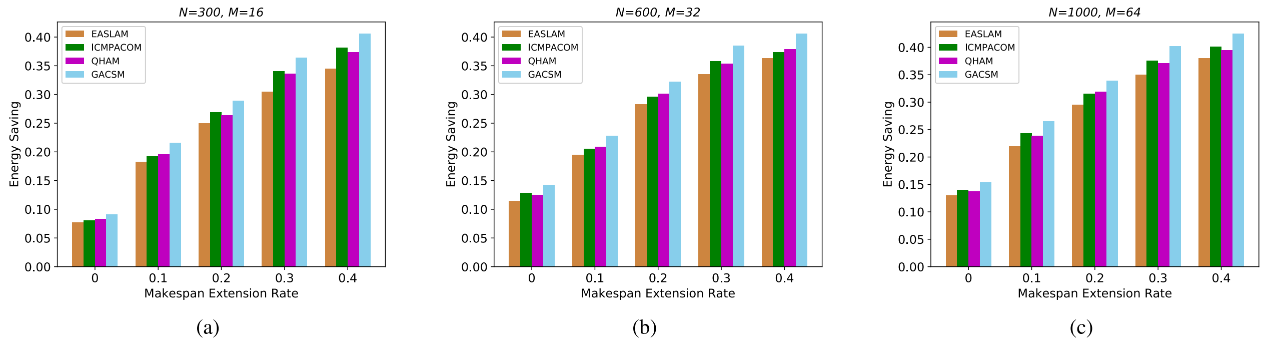


FIGURE 9. Energy saving of the algorithms. (a) Energy saving when $N = 300$, $M = 16$ vs. different makespan extension rates. (b) Energy saving when $N = 600$, $M = 32$ vs. different makespan extension rates. (c) Energy saving when $N = 1000$, $M = 64$ vs. different makespan extension rates.

The experimental results of randomly generated application graphs are shown in Table 5, which reports the statistical performance comparison of four algorithms, where "-" indicates that the data is infeasible. Table 5 shows that GACSM achieves better performance than ICMPACOM and QHAM in many of the test instances, such as R2, R3, R4, R10, R12, R14, R17 and R18. In contrast, ICMPACOM and QHAM does not outperform GACSM in any instance. In addition, the feasible rate and mean values of GACSM is better than ICMPACOM and QHAM (see Table 5), which shows that our GACSM algorithm has strong global search performance and high robustness.

Figure 6-8 plot the convergence of energy consumption for processing the R2, R4, and R6 test cases, which are taken as representatives of 18 test cases. Figure 6-8 also show that their convergence speeds are rather different, i.e., the GACSM algorithm converges faster than ICMPACOM and QHAM. It can be observed from the figures that the final energy consumption achieved by GACSM is better than the other two algorithms. The reason behind lies in that our GACSM algorithm uses the optimal individual-guided population search strategy of Gauss random walk, thus its search speed is fast; and GACSM has strong local search ability in the later stage.

In what follows, we use another parameter ESR to compare the energy saving of EASLAM, ICMPACOM, and QHAM with our proposed algorithm. The energy saving results of the algorithms with respect to various makespan extension rates are shown in Fig. 9. The number of tasks is set at 300, 600 and

1000, respectively, and the number of processors is set at 16, 32 and 64, respectively. We can see from Fig. 9 that our GACSM algorithm outperforms the other three algorithms under different conditions. As the makespan extension rate increases, the energy saving results of four algorithms also increase. Our GACSM algorithm can improve on energy consumption by 14.9%, 6.9%, 8.4% than the EASLAM, ICMPACOM, and QHAM algorithms respectively when $\zeta = 0.3, N = 1000$ and $M = 64$. GACSM, ICMPACOM, and QHAM algorithms are outperform EASLAM in all case. The reason behind this is that EASLAM algorithm adopts heuristic strategy, and it is not easy to find good solution in complicated task graph; while GACSM, ICMPACOM, and QHAM algorithms adopt random search strategy, which can be used to solve complicated problems and have better search ability in large solution space, thus they can find better results than EASLAM.

VI. CONCLUSION

In this paper, we address the problem of energy-aware scheduling on heterogeneous computing systems with time constraint. We propose an improved cuckoo search algorithm incorporating a heuristic strategy to solve task scheduling on heterogeneous multiprocessor systems with DVFS. We first present an improved cuckoo search algorithm based on Gaussian random walk and adaptive discovery probability to establish the mapping of tasks and processor voltage states. We then give a downward rank heuristic strategy to find the corresponding scheduling sequence. Finally, we present a cost-to-ratio modification strategy to further improve the performance of the GACS. The simulation results show that our proposed algorithm exhibits better performance than the state-of-the-art algorithms.

In the future, we plan to consider new guided random search algorithms to solve the DVFS-based task scheduling problem. Moreover, we plan to find more effective and efficient scheduling algorithms which can reduce time complexity and improve energy efficiency.

REFERENCES

[1] Y. Wang, K. Li, H. Chen, L. He, and K. Li, "Energy-aware data allocation and task scheduling on heterogeneous multiprocessor systems with time constraints," *IEEE Trans. Emerg. Topics Comput.*, vol. 2, no. 2, pp. 134–148, Jun. 2014.

[2] H. Fu, J. Liao, J. Yang, L. Wang, Z. Song, X. Huang, C. Yang, W. Xue, F. Liu, and F. Qiao, "The sunway taihulight supercomputer: System and applications," *Sci. China Inf. Sci.*, vol. 59, no. 7, 2016, Art. no. 072001.

[3] S. Chen, Z. Li, B. Yang, and G. Rudolph, "Quantum-inspired hyper-heuristics for energy-aware scheduling on heterogeneous computing systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 6, pp. 1796–1810, Jun. 2016.

[4] J. Singh, S. Betha, B. Mangipudi, and N. Auluck, "Contention aware energy efficient scheduling on heterogeneous multiprocessors," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 5, pp. 1251–1264, May 2015.

[5] Y. C. Lee and A. Y. Zomaya, "Energy conscious scheduling for distributed computing systems under different operating conditions," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 8, pp. 1374–1381, Aug. 2011.

[6] D. L. Li and J. Wu, "Minimizing energy consumption for frame-based tasks on heterogeneous multiprocessor platforms," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 3, pp. 810–823, Mar. 2015.

[7] G. Xie, G. Zeng, X. Xiao, R. Li, and K. Li, "Energy-efficient scheduling algorithms for real-time parallel applications on heterogeneous distributed embedded systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 12, pp. 3426–3442, Dec. 2017.

[8] Y. Zhang, Y. Wang, X. Tang, X. Yuan, and Y. Xu, "Energy-efficient task scheduling on heterogeneous computing systems by linear programming," *Concurrency Comput., Pract. Exper.*, vol. 30, no. 19, p. e4731, Oct. 2018.

[9] K. Huang, X. Jiang, X. Zhang, R. Yan, K. Wang, D. Xiong, and X. Yan, "Energy-efficient fault-tolerant mapping and scheduling on heterogeneous multiprocessor real-time systems," *IEEE Access*, vol. 6, pp. 57614–57630, 2018.

[10] K. Li, "Scheduling precedence constrained tasks with reduced processor energy on multiprocessor computers," *IEEE Trans. Comput.*, vol. 61, no. 12, pp. 1668–1681, Dec. 2012.

[11] A. Gandhi, M. Harchol-Balter, R. Das, and C. Lefurgy, "Optimal power allocation in server farms," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 37, pp. 157–168, Jun. 2009.

[12] Y. Wen, H. Xu, and J. Yang, "A heuristic-based hybrid genetic-variable neighborhood search algorithm for task scheduling in heterogeneous multiprocessor system," *Inf. Sci.*, vol. 181, no. 3, pp. 567–581, Feb. 2011.

[13] Y. Wang, K. Li, and K. Li, "Dynamic data allocation and task scheduling on multiprocessor systems with NVM-based SPM," *IEEE Access*, vol. 7, pp. 1548–1559, 2019.

[14] F. A. Omara and M. M. Arafa, "Genetic algorithms for task scheduling problem," *J. Parallel Distrib. Comput.*, vol. 70, no. 1, pp. 13–22, 2010.

[15] Y. Xu, K. Li, J. Hu, and K. Li, "A genetic algorithm for task scheduling on heterogeneous computing systems using multiple priority queues," *Inf. Sci.*, vol. 270, pp. 255–287, Jun. 2014.

[16] D. Konar, S. Bhattacharyya, K. Sharma, S. Sharma, and S. R. Pradhan, "An improved hybrid quantum-inspired genetic algorithm (HQIGA) for scheduling of real-time task in multiprocessor system," *Appl. Soft Comput.*, vol. 53, pp. 296–307, Apr. 2017.

[17] P. K. Muhuri, A. Rauniyar, and R. Nath, "On arrival scheduling of real-time precedence constrained tasks on multi-processor systems using genetic algorithm," *Future Generation Comput. Syst.*, vol. 93, pp. 702–726, Apr. 2019.

[18] H. Zhang, J. Xie, J. Ge, Z. Zhang, and B. Zong, "A hybrid adaptively genetic algorithm for task scheduling problem in the phased array radar," *Eur. J. Oper. Res.*, vol. 272, no. 3, pp. 868–878, 2019.

[19] S. Basu, M. Karuppiah, K. Selvakumar, K.-C. Li, S. H. Islam, M. M. Hassan, and M. Z. A. Bhuiyan, "An intelligent/cognitive model of task scheduling for IoT applications in cloud computing environment," *Future Gener. Comput. Syst.*, vol. 88, pp. 254–261, Nov. 2018.

[20] A. Arunarani, D. Manjula, and V. Sugumaran, "Task scheduling techniques in cloud computing: A literature survey," *Future Gener. Comput. Syst.*, vol. 91, pp. 407–415, Feb. 2019.

[21] H. Yuan, J. Bi, and M. Zhou, "Spatial task scheduling for cost minimization in distributed green cloud data centers," *IEEE Trans. Autom. Sci. Eng.*, vol. 16, no. 2, pp. 729–740, Apr. 2019.

[22] H. Yuan, J. Bi, M. Zhou, and A. C. Ammari, "Time-aware multi-application task scheduling with guaranteed delay constraints in green data center," *IEEE Trans. Autom. Sci. Eng.*, vol. 15, no. 3, pp. 1138–1151, Jul. 2018.

[23] M. Kurdi, "Ant colony system with a novel Non-DaemonActions procedure for multiprocessor task scheduling in multistage hybrid flow shop," *Swarm Evol. Comput.*, vol. 44, pp. 987–1002, Feb. 2019.

[24] W. Li, B. Jia, H. Xu, Z. Zong, and T. Watanabe, "A multi-task scheduling mechanism based on ACO for maximizing workers' benefits in mobile crowdsensing service markets with the Internet of Things," *IEEE Access*, vol. 7, pp. 41463–41469, 2019.

[25] Z. Zhang, F. Hu, and N. Zhang, "Ant colony algorithm for satellite control resource scheduling problem," *Appl. Intell.*, vol. 48, no. 10, pp. 3295–3305, Oct. 2018.

[26] W. Deng, J. Xu, and H. Zhao, "An improved ant colony optimization algorithm based on hybrid strategies for scheduling problem," *IEEE Access*, vol. 7, pp. 20281–20292, 2019.

[27] Y. Hu, C. Liu, K. Li, X. Chen, and K. Li, "Slack allocation algorithm for energy minimization in cluster systems," *Future Gener. Comput. Syst.*, vol. 74, pp. 119–131, Sep. 2017.

[28] X.-Y. Yang and S. Deb, "Cuckoo Search via Lévy flights," in *Proc. (NaBIC)*, Dec. 2009, pp. 210–214.

[29] X. S. Yang and S. Deb, "Engineering optimisation by cuckoo search," *Int. J. Math. Model. Numer. Optim.*, vol. 1, no. 4, p. 330, 2010.

[30] P. Civicioglu and E. Besdok, "A conceptual comparison of the cuckoo-search, particle swarm optimization, differential evolution and artificial bee colony algorithms," *Artif. Intell. Rev.*, vol. 39, no. 4, pp. 315–346, 2013.

[31] M. Shehab, A. T. Khader, and M. A. Al-Betar, "A survey on applications and variants of the cuckoo search algorithm," *Appl. Soft Comput..*, vol. 61, pp. 1041–1059, Dec. 2017.

[32] G. Sun, Y. Liu, Z. Chen, S. Liang, A. Wang, and Y. Zhang, "Radiation beam pattern synthesis of concentric circular antenna arrays using hybrid approach based on cuckoo search," *IEEE Trans. Antennas Propag.*, vol. 66, no. 9, pp. 4563–4576, Sep. 2018.

[33] J. Cheng, L. Wang, Q. Jiang, and Y. Xiong, "A novel cuckoo search algorithm with multiple update rules," *Appl. Intell.*, vol. 48, no. 11, pp. 4192–4211, Nov. 2018.

[34] J. Chrouta, W. Chakchouk, A. Zaafouri, and M. Jemli, "Modeling and control of an irrigation station process using heterogeneous cuckoo search algorithm and fuzzy logic controller," *IEEE Trans. Ind. Appl.*, vol. 55, no. 1, pp. 976–990, Jan. 2019.

[35] H. Zhu, X. Qi, F. Chen, X. He, L. Chen, and Z. Zhang, "Quantum-inspired cuckoo co-search algorithm for no-wait flow shop scheduling," *Appl. Intell.*, vol. 49, no. 2, pp. 791–803, Feb. 2019.

[36] K. Thirugnanasambandam, S. Prakash, V. Subramanian, S. Pothula, and V. Thirumal, "Reinforced cuckoo search algorithm-based multimodal optimization," *Appl. Intell.*, vol. 49, no. 6, pp. 2059–2083, Jun. 2019.

[37] T. Hosseinalizadeh, S. M. Salamati, S. A. Salamati, and G. B. Gharehpetian, "Improvement of identification procedure using hybrid cuckoo search algorithm for turbine-governor and excitation system," *IEEE Trans. Energy Convers.*, vol. 34, no. 2, pp. 585–593, Jun. 2019.

[38] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 3, pp. 260–274, Mar. 2002.

[39] R. Tariq, F. Aadil, M. F. Malik, S. Ejaz, M. U. Khan, and M. F. Khan, "Directed acyclic graph based task scheduling algorithm for heterogeneous systems," in *Proc. SAI Intell. Syst. Conf.* Cham, Switzerland: Springer, 2018, pp. 936–947.

[40] J. Li, Y. Liu, H. Li, Z. Yuan, C. Fu, J. Yue, X. Feng, C. J. Xue, J. Hu, and H. Yang, "PATH: Performance-aware task scheduling for energy-harvesting nonvolatile processors," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 26, no. 9, pp. 1671–1684, Sep. 2018.

[41] F. Juarez, J. Ejarque, and R. M. Badia, "Dynamic energy-aware scheduling for parallel task-based application in cloud computing," *Future Gener. Comput. Syst.*, vol. 78, pp. 257–271, Jan. 2018.

[42] X. Tang and X. Liao, "Application-aware deadline constraint job scheduling mechanism on large-scale computational grid," *PLoS ONE*, vol. 13, no. 11, Nov. 2018, Art. no. e0207596.

[43] X. Tang, X. Liao, J. Zheng, and X. Yang, "Energy efficient job scheduling with workload prediction on cloud data center," *Cluster Comput.*, vol. 21, no. 3, pp. 1581–1593, Sep. 2018.

[44] X. Tang, K. Li, and G. Liao, "An effective reliability-driven technique of allocating tasks on heterogeneous cluster systems," *Cluster Comput.*, vol. 17, no. 4, pp. 1413–1425, Dec. 2014.

[45] H. Izadkhah, "Learning based genetic algorithm for task graph scheduling," *Appl. Comput. Intell. Soft Comput.*, vol. 2019, pp. 1–15, Feb. 2019.

[46] A. Mishra and P. Trivedi, "Benchmarking the contention aware nature inspired metaheuristic task scheduling algorithms," *Cluster Comput.*, vol. 22, pp. 1–17, May 2019.

[47] R. M. Sahoo and S. K. Padhy, "Improved crow search optimization for multiprocessor task scheduling: A novel approach," in *Proc. Int. Conf. Appl. Robot. Ind. Using Adv. Mech.* Cham, Switzerland: Springer, 2019, pp. 1–13.

[48] X. Tang, X. Li, and Z. Fu, "Budget-constraint stochastic task scheduling on heterogeneous cloud systems," *Concurrency Comput., Pract. Exper.*, vol. 29, no. 19, Oct. 2017, Art. no. e4210.

[49] K. Li, "Performance analysis of power-aware task scheduling algorithms on multiprocessor computers with dynamic voltage and speed," *IEEE Trans. Parallel Distrib. Syst.*, vol. 19, no. 11, pp. 1484–1497, Nov. 2008.

[50] Y. Zhang, Y. Wang, and H. Wang, "Energy-efficient task scheduling for DVFS-enabled heterogeneous computing systems using a linear programming approach," in *Proc. IEEE 35th Int. Perform. Comput. Commun. Conf. (IPCCC)*, Dec. 2016, pp. 1–8.

[51] S. Nesmachnow, B. Dorronsoro, J. E. Pecero, and P. Bouvry, "Energy-aware scheduling on multicore heterogeneous grid computing systems," *J. Grid Comput.*, vol. 11, no. 4, pp. 653–680, Dec. 2013.

[52] K. Li, X. Tang, and K. Li, "Energy-efficient stochastic task scheduling on heterogeneous computing systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 11, pp. 2867–2876, Nov. 2014.

[53] M. Mezmaz, N. Melab, Y. Kessaci, Y. Lee, E.-G. Talbi, A. Zomaya, and D. Tuyttens, "A parallel bi-objective hybrid metaheuristic for energy-aware scheduling for cloud computing systems," *J. Parallel Distrib. Comput.*, vol. 71, no. 11, pp. 1497–1508, Nov. 2011.

[54] L. Mashayekhy, M. M. Nejad, D. Grosu, Q. Zhang, and W. Shi, "Energy-aware scheduling of MapReduce jobs for big data applications," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 10, pp. 2720–2733, Oct. 2015.

[55] J. Kang and S. Ranka, "Slack allocation algorithm for parallel machines," *J. Parallel Distrib. Comput.*, vol. 70, no. 1, pp. 23–34, Jan. 2010.

[56] Q. Huang, S. Su, J. Li, P. Xu, K. Shuang, and X. Huang, "Enhanced energy-efficient scheduling for parallel applications in cloud," in *Proc. 12th IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput. (CCGRID)*, May 2012, pp. 781–786.

[57] Z. Tang, L. Qi, Z. Cheng, K. Li, S. U. Khan, and K. Li, "An energy-efficient task scheduling algorithm in DVFS-enabled cloud environment," *J. Grid Comput.*, vol. 14, no. 1, pp. 55–74, Mar. 2016.

[58] K. Li, "Energy and time constrained task scheduling on multiprocessor computers with discrete speed levels," *J. Parallel Distrib. Comput.*, vol. 95, pp. 15–28, Sep. 2016.

[59] X. Xiao, G. Xie, C. Xu, C. Fan, R. Li, and K. Li, "Maximizing reliability of energy constrained parallel applications on heterogeneous distributed systems," *J. Comput. Sci.*, vol. 26, pp. 344–353, May 2018.

[60] V. Shekar and B. Izadi, "Energy aware scheduling for DAG structured applications on heterogeneous and DVS enabled processors," in *Proc. Int. Conf. Green Comput.*, Aug. 2010, pp. 495–502.

[61] G. Terzopoulos and H. D. Karatza, "Bag-of-task scheduling on power-aware clusters using a DVFS-based mechanism," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops*, May 2014, pp. 833–840.

[62] L. Zhang, K. Li, Y. Xu, J. Mei, F. Zhang, and K. Li, "Maximizing reliability with energy conservation for parallel task scheduling in a heterogeneous cluster," *Inf. Sci.*, vol. 319, pp. 113–131, Oct. 2015.

[63] S. Chen, Z. Li, B. Yang, and Y. Li, "Hyper-heuristic energy-aware scheduling algorithm on cloud computation system," *Comput. Eng. Appl.* , vol. 52, no. 2, pp. 74–80, 2016.

[64] X. Feng, R. Ge, and K. Cameron, "Power and energy profiling of scientific applications on distributed systems," in *Proc. 19th IEEE Int. Parallel Distrib. Process. Symp.*, Apr. 2005, p. 10.

[65] R. Ge, X. Feng, S. Song, H.-C. Chang, D. Li, and K. W. Cameron, "PowerPack: Energy profiling and analysis of high-performance systems and applications," *IEEE Trans. Parallel Distrib. Syst.*, vol. 21, no. 5, pp. 658–671, May 2010.

[66] F. Zheng, K. Zhang, and G. Wu, "Architecture techniques of many-core processor for energy-efficient in high performance computing," *China J. Comput.*, vol. 37, no. 10, pp. 2176–2186, 2014.

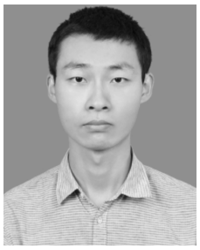
[67] Ş. Doğan, "A new data hiding method based on chaos embedded genetic algorithm for color image," *Artif. Intell. Rev.*, vol. 46, no. 1, pp. 129–143, Jun. 2016.

[68] A. Qin and P. Suganthan, "Self-adaptive differential evolution algorithm for numerical optimization," in *Proc. IEEE Congr. Evol. Comput.*, vol. 2, Dec. 2005, pp. 1785–1791.

[69] M. Li, H. Zhao, X. Weng, and T. Han, "Differential evolution based on optimal Gaussian random walk and individual selection strategies," *Control Decis.*, vol. 31, no. 8, pp. 1379–1386, 2016.

[70] W. Gong, Z. Cai, and D. Liang, "Engineering optimization by means of an improved constrained differential evolution," *Comput. Methods Appl. Mech. Eng.*, vol. 268, pp. 884–904, Jan. 2014.

ZEXI DENG received the B.S. and M.S. degrees from Guangxi University for Nationalities. He is currently pursuing the Ph.D. degree with Sun Yat-sen University. His research interests include high-performance computing, cloud computing, machine learning, and deep learning.



ZIHAN YAN received the B.S. degree from Zhejiang Gongshang University. He is currently pursuing the M.S. degree with Sun Yat-sen University. His research interests include high-performance computing and machine learning.

HUIMIN HUANG received the B.S. degree from Shihezi University and the M.S. degree from Chongqing Jiaotong University. She is currently pursuing the Ph.D. degree with Sun Yat-sen University. Her research interests include influence diffusion in social networks, recommendation systems, and machine learning.

HONG SHEN received the B.Eng. degree from the Beijing University of Science and Technology, the M.Eng. degree from the University of Science and Technology of China, and the Ph.Lic. and Ph.D. degrees from Abo Akademi University, Finland, all in computer science. He was a Professor and the Chair of the Computer Networks Laboratory, Japan Advanced Institute of Science and Technology, from 2001 to 2006. He has been a Professor (Chair) of computer science with Griffith University, Australia, where he taught nine years, since 1992. He is currently a specially appointed Professor with Sun Yat-sen University, China, and a tenured Professor (Chair) of computer science with The University of Adelaide, Adelaide, SA, Australia. He has published more than 300 articles, including more than 100 articles in international journals, such as a variety of IEEE and ACM transactions. His main research interests include parallel and distributed computing, algorithms, data mining, privacy preserving computing, high-performance networks, and multimedia systems. He was a recipient of many honours and awards.