Applied Soft Computing Journal 91 (2020) 106202

Contents lists available at ScienceDirect

Applied Soft Computing Journal

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

A hybrid algorithm for task scheduling on heterogeneous multiprocessor embedded systems

Golnaz Taheri [a,*], Ahmad Khonsari [b,a], Reza Entezari-Maleki [c,a,d], Leonel Sousa [d]

[a] School of Computer Science, Institute for Research in Fundamental Sciences (IPM), Tehran, Iran
[b] School of Electrical and Computer Engineering, University of Tehran, Tehran, Iran
[c] School of Computer Engineering, Iran University of Science and Technology, Tehran, Iran
[d] INESC-ID, Instituto Superior Técnico, Universidade de Lisboa, Lisbon, Portugal

ARTICLE INFO

Article history:
Received 25 April 2019
Received in revised form 7 January 2020
Accepted 23 February 2020
Available online 29 February 2020

Keywords:
Power consumption
Real-time embedded systems
Evolutionary algorithms
Task graph

ABSTRACT

Most of the scheduling algorithms proposed for real-time embedded systems, with energy constraints, try to reduce power consumption. However, reducing the power consumption may decrease the computation speed and impact the makespan. Therefore, for real-time embedded systems, makespan and power consumption need to be considered simultaneously. Since task scheduling is an NP-hard problem, most of the proposed scheduling algorithms are not able to find the multi-objective optimal solution. In this paper, we propose a two-phase hybrid task scheduling algorithm based on decomposition of the input task graph, by applying spectral partitioning. The proposed algorithm, called G-SP, assigns each part of the task graph to a low power processor in order to minimize power consumption. Through experiments, we compare the makespan and power consumption of the G-SP against well-known algorithms of this area for a large set of randomly generated and real-world task graphs with different characteristics. The obtained results show that the G-SP outperforms other algorithms in both metrics, under various conditions, involving different numbers of processors and considering several system configurations.

1. Introduction

For real-time applications, embedded systems have to produce a logically correct solution within a predefined deadline. For these applications, task scheduling in multiprocessor embedded systems is a great challenge. On the other hand, one of the important issues for the design of multiprocessor embedded systems is the trade-off between performance and power consumption [1]. For achieving high performance, suitable scheduling of applications on the resources of heterogeneous computing systems and proposer run-time support are required [2,3].

Real-time task scheduling on multiprocessors corresponds to sequencing the tasks and enabling them to execute on the chosen processors in order to attain a predefined objectives, e.g. satisfying the deadline, minimizing the total execution time or makespan, reducing power consumption, moderating thermal variations, etc. In parallel processing, the tasks are mostly represented by a directed acyclic graph (DAG), being large tasks generally broken down into smaller dependent subtasks. A subtask is generally represented as a node in the DAG and the dependencies between related subtasks are expressed by arcs connecting the nodes [4].

Several studies have been conducted to investigate task scheduling on multiprocessor systems, but there is still a large number of works trying to develop more efficient scheduling algorithms considering different requirements, objectives and system properties. Some of these algorithms are deterministic while others are non-deterministic. In this work, we define the task scheduling problem on heterogeneous multiprocessor systems as an optimization problem subject to some predefined constraints. Our objective is to minimize both the power consumption and makespan considering subtask precedence and task deadline. We propose a two-phase genetic algorithm hybridized with a spectral partitioning, called G-SP, to schedule tasks on processors of heterogeneous multiprocessor systems.

The proposed G-SP algorithm consists of two phases. The first phase is devoted to divide the incoming task graph into two nearly equal size subtask graphs based on the spectral partitioning algorithm. This phase tries to achieve load balancing and minimize data dependencies among subtasks. The second phase then schedules each partition applying genetic algorithm (GA), in order to minimize the makespan without violating precedence constraints. For minimizing the total power consumption, the scheduler tries to assign a selected subtask to a processor

* Corresponding author.
E-mail addresses: golnazthr@ipm.ir (G. Taheri), ak@ipm.ir (A. Khonsari), entezari@iust.ac.ir (R. Entezari-Maleki), las@inesc-id.pt (L. Sousa).

https://doi.org/10.1016/j.asoc.2020.106202
1568-4946/© 2020 Published by Elsevier B.V.

that consumes lower power. If the selected subtask cannot be scheduled on the target lower power processor, because of violation of the deadline, the proposed method selects an alternative processor that consumes higher power to execute the subtask within the deadline. For evaluating the proposed algorithm and comparing it to the state-of-the-art methods, e.g. HEFT-B, HEFT-T and CPOP [5], in terms of the makespan and power consumption, real-world and randomly generated task graphs are used. The results of this comparison, for all of the considered task graphs, show substantial improvement in both the makespan and power consumption.

The remaining of this paper is organized as follows. Section 2 discusses the related state-of-the-art, and Section 3 introduces the background to read this paper. Section 4 proposes the new algorithm for task scheduling with the main goals of minimizing the power consumption and the makespan on heterogeneous computing systems. Section 5 provides the results obtained by applying the proposed algorithm to a real multi-core system. Finally, Section 6 concludes the paper and presents some guidelines for future research.

2. Related work

The different approaches presented for task scheduling can be classified into two main groups: deterministic heuristic-based and guided random based methods [6]. Finding the near optimal solution in appropriate polynomial time that may approximate the exact solution is the main objective in the heuristic based methods [7]. One possible classification of heuristic-based scheduling algorithms groups these algorithms into three categories: list scheduling [4,6], cluster scheduling [8] and duplication-based scheduling [9]. List scheduling algorithms, as one of the most popular classes of heuristic-based methods, has two different phases. A priority is assigned to each task in the first phase, and then a processor is allocated to a task in the second phase.

Adequate sampling of possible solutions is typically needed in guided random based algorithms. Some of the well-known guided random based algorithms widely used for the scheduling problem, are the ant colony optimization (ACO) [10], simulated annealing (SA) [11], tabu search (TS) [12] and genetic algorithm (GA) [13,14]. Energy minimization subject to timing constraints is fundamental for embedded systems. Most of the static scheduling algorithms proposed for embedded systems are based on the worst or average case execution time for each subtask, being pessimistic [15]. In order to shorten the makespan of periodic tasks some uncertainties were taken into account and a probabilistic scheduling, based on the colony optimization, was introduced in [10]. A mathematical model and ant colony optimization algorithm have been proposed in [16] for scheduling with the goal of minimizing the makespan, considering various job sizes and parallel systems. This optimization algorithm can find an appropriate solution in a feasible time compared to other algorithms. A solution for off-line task scheduling and voltage selection using the ant colony optimization method has been proposed in [17]. This solution utilizes the unused time of subtasks that complete their execution earlier than expected time and re-maps the subtasks into processors in order to improve the energy consumption. A hybrid Max-Min ant colony algorithm for real-time task assignment, with two objectives, has been proposed in [18]. The first one is resource-objective, which tries to maximize the number of assigned tasks to the processors. The second one is energy-objective, for minimizing the cumulative energy consumption of all the tasks. In [19], the ant colony based algorithm for task scheduling in heterogeneous multiprocessors has been proposed. This algorithm uses the upward rank value to improve performance in comparison with other ant colony based algorithms. A probabilistic ant colony based algorithm for task scheduling on heterogeneous processors has been proposed in [20]. This algorithm minimizes the average waiting time of tasks. The results of the algorithm were compared with the ones obtained from the first come first served (FCFS), and it was concluded that the algorithm of [20] shows better waiting time. However, that algorithm implies increasing the makespan in comparison with the ant colony algorithm.

In [21], an agent-based power distribution approach for dynamic thermal management has been proposed. The proposed approach achieves about 44% improvement, in comparison with other dynamic thermal management methods, both in what concerns performance and energy consumption. A two-stage heuristic method for multi-satellite scheduling has been proposed in [22], which shows very close results to optimal solutions. A multi-heuristic evolutionary task scheduling algorithm that dynamically maps tasks onto processors in heterogeneous systems has been introduced in [23]. That algorithm uses the GA and combines it with different heuristic methods in order to minimize the total execution time. A fuzzy-based scheduling and load balancing algorithm for cloud service provisioning has been proposed in [24]. This algorithm minimizes the response time of arriving jobs and achieves better average success rate and resource scheduling efficiency. Two hybrid heuristic scheduling algorithms for minimizing the makespan have been presented in [25]. The first algorithm combines ACO with variable neighborhood search, and the second one combines GA with variable neighborhood search. In [26], a hybrid algorithm that uses the combination of the GA and the ant colony method has been proposed. This algorithm uses the ant colony to find an optimum path, and then refines it by applying the GA.

An energy-efficient genetic algorithm for task scheduling on asymmetric multiprocessors has been proposed in [27]. This algorithm adaptively changes the generation strategies to the solution candidates based on the energy consumption and makespan of them. The algorithm uses three different generation strategies instead of regular genetic operators and reduces the power consumption in comparison with two classic genetic-based scheduling algorithms. A genetic-based algorithm for optimizing the energy consumption and performance in multiprocessors has also been proposed in [28]. This algorithm has two different selection operators to reduce energy consumption with increments in the finish time. In contrast to our proposed algorithm, achieving both energy efficiency and performance in that algorithm is a conflicting issue and a trade-off has to be made with respect to system and application requirements. A multi-objective energy-aware scheduling algorithm, based on decomposition for scheduling of the system workflow, has been proposed in [29]. The authors use different conditions, such as various crossover and mutation probabilities, to compare their algorithm with others. Experimental results show that this algorithm has significant improvements in diversity and convergence to the final solutions. However, the final solutions are far from the Pareto optimal.

To reduce the peak temperature, achieve a thermally even distribution, and meeting real-time constraints in the embedded system, a heuristic method for task scheduling and floor-planning has been proposed in [30]. A task scheduling algorithm for real-time multi-core systems has been presented in [31]. This algorithm tries to meet the deadlines while reducing the thermal stress and improving the expected lifetime of the system. An adaptive thermal-aware task scheduling for multi-core systems has been proposed in [32]. Authors of [32] have also proposed an algorithm to prevent the system from overheating, maximizing the utilization of the system and managing the inter-core thermal relation with different variations of dynamic task scheduling.

The key difference between the algorithm proposed in this paper and the previously presented approaches is the adoption

Table 1
Comparison of the proposed algorithm with the related state-of-the-art.

Approach and Goal	Ref.											
	[18]	[19]	[20]	[22]	[23]	[24]	[25]	[26]	[27]	[28]	[29]	$G-SP$
Power-Aware	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓
Performance-Aware	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓	✗	✓
Heuristic approach	✓	✗	✗	✓	✓	✗	✓	✓	✗	✗	✗	✓
Evolutionary approach	✓	✓	✓	✗	✓	✗	✓	✓	✓	✓	✓	✓
Year of publication	2018	2018	2016	2018	2010	2019	2018	2018	2019	2018	2017	–
Model name	H-MMAS	rACS	AS-ACO	WBG	PN	FMRS	ACO-VNS	GAACO	AGATS	PRWS	MOEA/D	G-SP

of an hybrid method that integrates heuristic algorithms with the GA. The idea of the proposed algorithm is to use the advantages of both GA and heuristic-based algorithms, while overcoming their disadvantages, to minimize the total power consumption. Evolutionary algorithms, such as genetic, have some practical advantages for optimization problems. These advantages include self-adaptation, robustness and simplicity. These kinds of algorithms can find solutions for the problems that heuristic methods cannot find suitable results for them. However, there are some types of problems that direct evolutionary algorithms may fail to find optimal solutions for them [33]. This reason leads to the need for hybridization of evolutionary algorithms with other algorithms like heuristic ones. The hybrid algorithms can cover a larger search space in comparison with pure heuristic algorithms. Moreover, they can improve the performance of the solution, e.g. increasing the speed of the convergence to the final result. Table 1 summarizes the differences between our algorithm and the related works, considering both the approach and the objectives.

Table 1 shows that the proposed G-SP algorithm takes advantage of both evolutionary and heuristic approaches. It also considers power and performance at the same time for improving both of them. In all of the related work, listed in Table 1, evolutionary or heuristic approaches were used. The acronyms of the adopted scheduling methods and the year they were proposed are also listed in the table. In some of them that combine both evolutionary and heuristic approaches, the proposed method is not able to simultaneously improve power and performance.

3. Background information

Task scheduling is one of the famous NP-hard problems. A computing system can be homogeneous or heterogeneous. Homogeneous systems use the same kind of processors, while heterogeneous systems use various processors with different processing capabilities. In this work, our target system is heterogeneous with two kinds of processors that exhibit different power and performance capabilities. Different scheduling algorithms have been proposed to solve the scheduling problem in multiprocessor systems. Two of the main categories for task scheduling are deterministic and non-deterministic approaches. These algorithms were listed and the main characteristics introduced in the previous section. This section provides the background information required to analyze those algorithms and the algorithm proposed herein.

3.1. Deterministic approaches

Deterministic algorithms always produce the same answer for a specific input. Most of deterministic scheduling algorithms are based on greedy methods and try to minimize the overall makespan. List scheduling, clustering algorithm and task duplication are three main categories of this approach. It is worthy to mention that these kind of algorithms can only solve the specific problems due to their greedy property.

- **List scheduling algorithm.** The list scheduling algorithms include two phases. In the first phase, the algorithm assigns a priority to each subtask and adds it to a waiting list. In the second phase, the subtask with the highest priority is selected and assigned to an available processor. Heterogeneous earliest finish time (HEFT) and critical path on processors (CPOP) are two of the list based algorithms. It should be noted that the performance of this kind of scheduling algorithms strongly depends on the efficacy and selection of the first heuristic that is applied.
- **Clustering algorithm.** This group of algorithms assumes that an unlimited number of processors exists for executing all of the subtasks. Clustering algorithms use as many processors as possible to reduce the overall makespan. In these algorithms, a set of tasks, that are highly correlated and need to communicate with each other, are grouped into a cluster. Then, each of these clusters is assigned to an available processor. Clustering algorithms encounter a serious problem when the number of clusters is more than the number of available processors. This situation leads to scheduling of multiple clusters on a single processor, which increases the makespan.
- **Task duplication algorithm.** This group of algorithms executes a key subtask on more than one processor in order to reduce the communication delay.

3.2. Non-deterministic approaches

Non-Deterministic algorithms produce various answers depending on the decision they make during the execution time. These algorithms have been used to solve a large number of problems that need more time to search the solution space for the final result. Several non-deterministic algorithms such as genetic algorithms (GA), ant colony optimization (ACO), simulated annealing (SA) and tabu search (TS) have been applied to solve the scheduling problem.

- **Ant Colony Optimization.** ACO algorithm simulates the behavior of ants when they move to find the food. Ants sprinkle the pheromone along the path that they move to find the food to help the other ants to select a path with a greater amount of pheromone with the objective of routing.
- **Tabu Search.** TS iteratively uses a neighborhood search algorithm and dynamically moves from a possible solution to the improved solution. Tabu search actively avoids local search's from visiting the neighbors that have been already visited.
- **Simulated Annealing.** SA is a convenient solution for finding the global optimum with the existence of the numbers of local optimums. It mimics the physical process of the annealing for formation of crystals.
- **Genetic Algorithm.** The term "genetic algorithm" stands for a class of algorithms that uses the natural evolution for solving complex optimization problems. In this algorithm, each solution is represented by a chromosome, and

a set of chromosomes called population, usually initialized randomly and maintained according to a fitness value, is generated in each iteration. A set of reproduction operators, such as selection, mutation and crossover, applied to reproduce new chromosomes and shape a new population. The components of the GA are as follows.

1. **Initial population:** The GA starts with a set of individuals that is called population. Each of these individuals is a possible candidate for the solution. An individual represented by a set of genes is randomly assembled to form a chromosome.
2. **Fitness function:** The fitness function specifies how fit an individual is. This function gives a fitness score to each individual. The probability of selection for each chromosome is also based on this score. In the scheduling problem, this factor can be a throughput, makespan or utilization.
3. **Genetic Operators:** The genetic operators create new individuals based on the current population. Different operators, such as selection, crossover and mutation, are described in the following.

   (a) **Selection:** The selection operator selects the individuals with the highest fitness values and lets these individuals pass their genes to the next generation. Two pairs of individuals named parents are selected based on their fitness values. Actually, the individuals with the higher fitness values or scores have a higher chance to be selected through the selection operator.

   (b) **Crossover:** The crossover is the most important phase in a genetic algorithm. It is a way to stochastically generate new solutions (chromosomes) from an existing population. The new offspring are added to the next generation population.

   (c) **Mutation:** In some of the new offspring, part of their genes can be changed or mutated with a low random probability. It is used to maintain genetic diversity from one generation of a population to the next.

4. The proposed algorithm

Directed acyclic graphs (DAGs) are mostly used to model different applications. A set of subtasks as a sequence, based on the predefined precedence constraints, can be represented by a DAG. The DAG shown in Fig. 1 represents an application with different subtasks. In this figure, each vertex shows a subtask and the edge between vertices shows the execution precedence between them. In this work, the communication time between subtasks is considered to be negligible, as [34–36]. We assume that the execution order of subtasks and the dependencies between them are static, without any change during scheduling. Each subtask of the input DAG can be executed on one of the processors. The target system contains a set of m heterogeneous processors and it is preferred to use the processors with lower power consumption at the beginning.

The proposed scheduling algorithm, G-SP, tries to schedule the input DAG to minimize the power consumption of heterogeneous MPSoC while satisfying timing constraints. The G-SP schedules tasks applying the GA, while it uses a heuristic approach, called spectral partitioning, in the processor mapping phase. The outline of the G-SP for DAG task scheduling on a heterogeneous computing system with two groups of low and high power-performance processors is shown in Algorithm 1.

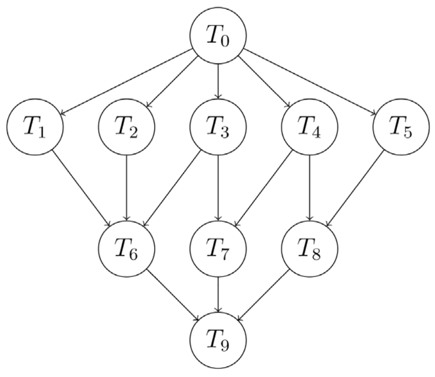

Fig. 1. A simple DAG containing 10 subtasks.

GA is a powerful effective algorithm for solving different optimization problems such as task scheduling. GA provides the advantages of a global search procedure, finding the optimal or near-optimal solution for the scheduling problem. The genetic operators for different problems have to be adapted to the problem under-study. It is noticeable that the generation of the initial population has a major impact on the performance of the final solution. On the other hand, to find better solutions, the hybridization of different algorithms is used. Hybrid algorithms are achieved by merging some features of a heuristics method to another heuristic to find a better solution. Hybrid algorithms usually show better performance compared to respective vanilla heuristic algorithms.

Two main phases can be identified in G-SP algorithm. In the first phase, spectral partitioning is applied by calling Algorithm 2 which divides the input DAG into two separate parts with minimum dependencies between them. In the second phase, the algorithm uses the output of the first phase to perform the task-to-processor mapping. For this mapping, a chromosome is generated for each part of the DAG to minimize the power consumption with respect to the makespan without violating the precedence constraints. Then, the fitness function for each chromosome is evaluated. In line 6 of Algorithm 1, based on the fitness values, the selection method chooses survivals for the next generation with the help of the roulette-wheels method. The selection policy makes the weaker chromosomes disappear and stronger chromosomes survive in order to generate a population of best solutions. In lines 7 and 8, the crossover and mutation operations, which are presented in Section 4.2, are performed. Crossover operation is not always accomplished because it is associated with a predefined crossover probability. The mutation operator also changes each gene in the population with a mutation probability. The mutation operator can generate a chromosome that cannot be produced with the crossover operation, helping to escape from local optimum solutions. Afterwards, the new generation is created and above procedure is done on the new generation. The outer loop is completed when the termination condition is satisfied.

4.1. Spectral partitioning

The partitioning of a graph into equal-sized parts while minimizing the number of edges between different parts is NP-Complete problem [37]. This problem, when referred to a graph to be partitioned into two sub-graphs is called Minimum Bisection [38,39].

Definition. Let $G = (V, E)$ denote a simple graph. A partition S is called $(k, 1 + \epsilon)$-separator if S partitions V into k

disjoint subsets of graph G, each subset contains at most $(1+\epsilon)\frac{n}{k}$ vertices and minimizes the number of edges between different components [40].

Proposition 1. *For every α and $k \geqslant 1$, there is a polynomial-time reduction from the problem of finding a clique of size k in a simple graph G, to the problem of finding a (k, α)-separator. Since the problem of finding a clique of size k in a simple graph is an NP-complete problem, finding a (k, α)-separator is also NP-Complete [37].*

Algorithm 1: G-SP algorithm

```
Input :
        Parameters for the genetic algorithm.
        Parameters for task scheduling.
Output:
        A task schedule.
 1. Call Algorithm 2 to create an initial population;
 2. repeat
 3.     Perform task to processor mapping and evaluate the fitness function;
 4.     Copy the elitism with no change and directly into the next population;
 5.     repeat
 6.         Use the roulette-wheel to select candidates as survival to the next population
            according to their fitness values;
 7.         Perform the crossover operation;
 8.         Perform the mutation operation;
 9.     until the new population is complete;
10.     Replace the old population with the new one;
11. until the termination situation is satisfied;
12. return the near optimal schedule as a result.
```

The Proposition 1 shows that the $(2, 1)$-separator, i.e. finding the minimum bisection problem is NP-complete. Spectral partitioning is one of the approximation algorithms to find the $(2, 1)$-separator for a simple graph G, in fact the most popular and successful heuristic algorithm for partitioning the simple graph. This method is widely used in many scientific applications, such as mapping finite elements on a parallel processor, solving linear systems and domain decomposition, web search and image partitioning [37]. In this paper, $G = (V, E)$ is a connected undirected graph with n vertices. A partition of a graph G is the division of the vertices into two disjoint subsets G_1 and G_2. Without loss of generality, we assume that $|G_1| \leqslant |G_2|$. Let $E(G_1, G_2)$ be the set of edges with one endpoint in G_1 and the other one in G_2. The cut size of this partitioning (G_1, G_2) is the number of edges between these two parts $|E(G_1, G_2)|$.

We approximate the balanced partitioning problem with the spectral partitioning approach since finding the balanced bipartite graph or $(2, 1)$-separator is NP-complete. This method is based on the eigenvector and eigenvalues of the Laplacian matrix of a graph. The spectral method recursively bisects the graph by considering the eigenvectors of the Laplacian matrix of the graph. The spectral partitioning has received much attention because of the good quality and efficiency [37]. We model the problem in the following way. In the graph $G = (V, E)$, e_{ij} in E denotes the edge between v_i and v_j. We assign the variable x_i to each vertex v_i such that the following condition is hold:

$$x_i = \pm 1 \text{ and } |\sum_{v_i} x_i| \le 1 \tag{1}$$

The first condition in Eq. (1) puts a partition into two different sets G_1 and G_2, while the second condition implies that the difference between the sizes of G_1 and G_2 should be at most one. The elements of vector $x = (x_1, \dots, x_n)$ indicates the set assignment of each graph vertex. In the next step, the function $f(x) = \frac{1}{4}\sum_{E_{ij}}(x_i - x_j)^2$ counts the number of edges crossing between G_1 and G_2. Let $= [a_{ij}]$ represent the adjacency matrix of graph G such that [41]:

$$a_{ij} = \begin{cases} 1 & if (v_i, v_j) \in E \\ 0 & otherwise \end{cases} \tag{2}$$

We define the diagonal degree matrix of graph G as $D = diag(d_i)$, where d_i is the degree of vertex v_i, which means the number of edges to the vertex v_i. We have the following equation,

$$\sum_{E_{ij}}(x_i - x_j)^2 = \sum_{E_{ij}}({x_i}^2 + {x_j}^2) - \sum_{E_{ij}} 2x_ix_j \tag{3a}$$

$$\sum_{E_{ij}} 2x_ix_j = \sum_{v_i}\sum_{v_j} x_iA_{ij}x_j = \sum_{v_i} x_i \sum_{v_j} A_{ij}x_j = x^TAx \tag{3b}$$

According to Eq. (1), $x_i = \pm 1$, then:

$$\sum_{E_{ij}}({x_i}^2 + {x_j}^2) = \sum_{E_{ij}} 2 = 2|E| = \sum_{v_i} {x_i}^2 d_i = x^TDx \tag{4}$$

We consider the Laplacian matrix of graph G by $L(G) = [l_{ij}]$ where;

$$l_{ij} = \begin{cases} 1 & if (v_i, v_j) \in E \\ d_i & if\, i = j \\ 0 & otherwise \end{cases} \tag{5}$$

$L(G) = D - A$ and $f(x) = \frac{1}{4}x^TLx$ Now we define the discrete bisection problem with respect to the constraints on x,

$$\begin{aligned} &\text{Minimize} && f(x) = \frac{1}{4}x^TLx \\ &\textit{subject to}: && x^T\mathbf{1} \leqslant \mathbf{1}, \quad x_i \in \{-1, 1\} \end{aligned}$$

where $\mathbf{1}$ is the vector $(1, 1, \dots, 1)^T$ with n elements. As we mentioned earlier, the graph partitioning is an NP-complete problem, therefore we relax the discreetness constraint on x and define the continuous version of problem,

$$\begin{aligned} &\text{Minimize} && f(x) = \frac{1}{4}x^TLx \\ &\textit{subject to}: && x^T\mathbf{1} \leqslant \mathbf{1}, \quad xx^T = n \end{aligned}$$

The elements of x can take on any values that satisfy the norm constraint. The continuous version of the problem is an approximation to the discrete version [42]. Therefore, the relaxation part of constraints is the major part of spectral partitioning. We propose our solution for the optimization problem by noting that the Laplacian matrix of a graph has several important properties. Let $(u_1, u_2, \dots, u_n)$be the normalized eigenvectors of $L(G)$ with the corresponding eigenvalues $(\lambda_1, \lambda_2, \dots, \lambda_n)$. Hence, we will have the following proposition [38].

Proposition 2. *The Laplacian matrix $L(G)$ of the graph G has the following properties,*

1. *$L(G)$ is a symmetric positive semidefinite matrix.*
2. *All the eigenvectors are pairwise orthogonal.*
3. *$\lambda_1 = 0$ and $u_1 = n^{-1/2}\mathbf{1}$.*
4. *If G is a connected graph, then the λ_1 is the only zero eigenvalue of $L(G)$.*

Now we express x in terms of eigenvectors of $L(G)$, $x = \sum_i \alpha_i u_i$ such that the α_i is real constants and $x = \sum_i \alpha_i^2 = n$. then,

$$f(x) = \frac{1}{4}(\alpha_2^2\lambda_2 + \alpha_3^2\lambda_3 + \cdots . + \alpha_n^2\lambda_n)$$

clearly,

$$(\alpha_2^2+\alpha_3^2+\cdots.+\alpha_n^2)\lambda_2 \leqslant \alpha_2^2\lambda_2+\alpha_3^2\lambda_3+\cdots.+\alpha_n^2\lambda_n$$

Therefore, $f(x) \geq n\lambda_2/4$. By choosing $x = \sqrt{n}u_2$, we can minimize $f(x) = n\lambda_2/4$. This choice for x satisfies the balance constraints since $x^T\mathbf{1} = u_2^Tu_1 = 0$ and proposes a solution for the continuous problem. It is noticeable that, if $\lambda_2 \neq \lambda_3$, this solution is unique. In this step, the solution of continuous problem should be mapped onto the discrete domain. With graph bipartitioning, there is a simple way to perform this mapping; i.e., according to the median value of u_2, put half of the vertices with the corresponding value greater than u_2 to one set and the other vertices in another set. This solution is the closest solution to the continuous one.

```
Algorithm 2: Spectral partitioning algorithm
Input :
    Parameters for the Input DAG as a workflow.
Output:
    Two nearly equal size sub-workflows with the minimum data dependencies.
1. Compute the Laplacian matrix of the graph from Adjacency matrix of DAG;
2. Compute the eigenvalues of the Laplacian matrix;
3. Compute the eigenvector of Laplacian according to the second smallest eigenvalue and sort its elements;
4. Insert half of the vertices in partition $G_1$ and remainder in partition $G_2$. .
5. return Two nearly equal size sub-workflows $G_1$ and $G_2$ with the minimum data dependencies.
```

After proposing the aforementioned solution, Algorithm 2 provides the method to create the initial population. This algorithm divides the workflow into two nearly equal-sized sub-workflows, by applying the spectral partitioning method in order to minimize the data dependencies between the two resulted sub-workflows. Afterwards, we schedule each of these sub-workflows onto a low-performance processor to maintain the load balancing policy in the system. After scheduling of these sub-workflows, if some subtasks miss the chance of meeting their deadlines, the subtasks move to high-performance processors to be served faster and meet their deadlines. In the following, the proposed spectral partitioning is combined with our GA to improve the performance and power consumption of the system.

4.2. The proposed genetic algorithm

In most of the applications, the state space of the problem is quite large and the convergence to the final solution can be quite slow. Different factors, like the initial configuration, have a major impact on the speed of convergence. The chromosome representation and generation of the initial population are major parts in GA and operations like mutation and crossover help to reach the final population and attain the semi-optimal solution. In this section, the encoding mechanism for the task scheduling algorithm is introduced, representing the problem in the context of genes and chromosomes. It is noticeable that any chromosome represents a unique schedule. After introducing the chromosome representation, other genetic operators are presented.

In our task scheduling problem, each chromosome corresponds to a unique schedule. A chromosome contains a permutation of integers $0, 1, 2, \ldots, n-1$, representing a list of n subtasks. The tuple $(T_0, T_1, T_2, \ldots, T_i, \ldots, T_{n-1})$ represents a set of all subtasks of a given DAG application. Each gene of the chromosome represents one of these subtasks and the order of these subtasks should be a valid one according to their topological order. A valid chromosome in the proposed algorithm schedules all subtasks of the given DAG and the resulting schedule satisfies all precedence constraints.

The initial population size should be constant through different generations. The idea of selecting the suitable size is always a trade-off between effectiveness and efficiency. Initially, a chromosome is generated with respect to the output of Algorithm 2. Let $T(h)$ be the set of subtasks with the height h in each subworkflow. We choose r random subtasks $0 < r < T(h)$ that are the first r subtasks to be assigned to the processor. Then, we remove these r subtasks form $T(h)$ and repeat this step to schedule all subtasks. By selecting subtasks from the lowest to the highest height in each step, the proposed algorithm never violates the precedence constraints.

After producing the initial population, fitness value of each chromosome is computed and the chromosomes are sorted according to their fitness values. Afterwards, the selection method is applied to the chromosomes and the strongest individuals are selected to be paired up and copied to the next generation with the hope of producing stronger population.

4.2.1. Subtask-to-processor mapping

In this section, we first describe the heuristic subtask to processor mapping, based on spectral partitioning method for DAG applications, with timing and precedence constraints. Then, the method for computing the fitness value, that has an important role to generate the next generation population, is presented.

For the subtask-to-processor mapping, a heuristic approach is presented to minimize power consumption. With the proposed heuristic-based algorithm for the subtask-to-processor assignment, subtasks are selected through spectral partitioning and the selected part is assigned to the low power consumption processors. We schedule each of these sub-workflows on a low-performance processor to keep up the load balancing. Afterwards, if some of the subtasks miss their deadlines, they are moved to high-performance processors to serve faster and meet their deadlines with a predefined penalty in the fitness function.

The fitness value plays an important role in generating the next-generation population. For our task scheduling problem, the goal is to obtain an assignment with the minimum overall power consumption and minimum makespan without violating the precedence constraints. Therefore, our solution has two concerns: the first one is to minimize the makespan, and the second one is to minimize the power consumption. For the first concern, we define a *Score* value as a penalty for a subtask which misses its deadline, as Eq. (6).

$$Score = Score - Max(|(delta, 100)|) \tag{6}$$

The parameter *delta* shows amount of time that a subtask misses the predefined deadline. If a subtask is served before its deadline, we add this time to its score.

$$Score = Score + delta$$

For the second concern, we add the doubled value of total execution time on low power processors ($T_{low-power}$) to the score value. For the high power processors ($T_{high-power}$), we reduce the total execution time from *Score* as a cost. The fitness value for each scheduling is the summation of the score for all of its genes.

$$Score = Score + 2 * (T_{low-power})$$

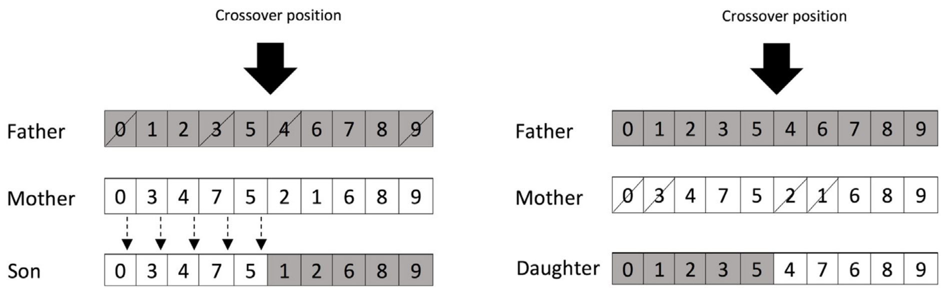

Fig. 2. Single-point crossover operator.

4.2.2. Genetic operators

The genetic operators, namely selection, crossover and mutation, used in the proposed algorithm are described in the following.

The selection has a major impact on the convergence. We use the roulette-wheel selection, one of the most used methods for selection purpose. In this method, the chromosomes with higher fitness value have a higher chance to be selected. The probability p_i of each chromosome ($i \in \{1, 2, \dots, n\}$ where n is the number of individuals in the population) is defined by Eq. (7)

$$p_i = \frac{fitness(i)}{\sum_{j=1}^{n} fitness(j)} \tag{7}$$

and

$$Select(i) = \sum_{j=1}^{i} p_j \tag{8}$$

where $Select(i)$ is the sum of p_j from $j = 1$ to i. For a random number $R \in [0, 1]$, if $Select(i) > R$, its related individual is selected. With this method, the chromosome with a higher fitness value is selected with a higher probability. The crossover operation merges two valid parents to generate two new offspring that are topologically ordered. The mutation operator changes each gene such that the diversity of population covers the larger search space. The mutation operation takes place with a certain probability, it helps the search phase of the algorithm to escape from local sub-optimum points.

Since there are precedence relations between subtasks in a DAG, not all permutation of n subtasks is a valid schedule. During all phases of the GA, the encoder and the decoder should always yield a valid solution to the scheduling problem. Therefore, a valid schedule should have two conditions. First, the precedence constraints of a DAG application should be satisfied. Second, every subtask should be presented only one time in the schedule.

Lemma 1. *For a DAG application, a solution for the task scheduling problem is a topological order for the subtasks that respects the precedence conditions. When a subtask T_j is deleted from the topological order, the remaining subtasks still have a topological order without violating precedence constraints.*

Lemma 2. *For a DAG application, a subtask T_j can be inserted into any position between successor and predecessor of subtask T_j, which can generate a new topological order without violating precedence constraints.*

As it can be seen in the example of Fig. 2, we have 10 subtasks from Fig. 1 to schedule. The corresponding chromosome to this problem is coded as an integer. For $i = 5$, two parents exchange some genes to generate two children, with the crossover operator. The left segment of the new child is derived from the corresponding part of parent 1 (0, 1, 2, 3, 5) or parent 2 (0, 3, 4, 7, 5).

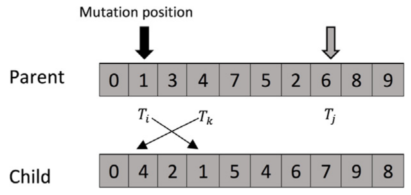

Fig. 3. Mutation operator.

According to Lemma 1, we can delete some of the subtasks from the topological order and the remaining subtasks still maintain a topological order without violating precedence constraints. For example, the left part of parent 2 still has a topological order $(4, 7, 6, 8, 9)$ when we delete the subtasks on the left part of parent 1 $(0, 1, 2, 3, 5)$. Similarly, the left part of parent 1 is still in a topological order $(1, 2, 5, 8, 9)$ when we delete the subtasks on the left part of parent 1 $(0, 3, 4, 7, 5)$. Finally, according to Lemma 2, the left part of the new offspring is copied from parent 1 $(4, 7, 6, 8, 9)$ or parent 2 $(1, 2, 6, 8, 9)$ by putting the genes one by one from left to right.

The mutation operation tries to insert some variations into the individuals to achieve a better solution and escape from local optimal points. According to Lemmas 1 and 2, we can exchange two genes without violating precedence constraints. A subtask or gene can move to any position between its successor and predecessor in a valid chromosome. For example, lets T_i denote a randomly selected subtask. Then, we can find T_j, the first successor of subtask T_i for $\exists k \in [i+1, j+1]$ and the predecessor of T_k, where T_i and T_k can be integrated. This method can generate a new offspring without violating the precedent constraints. Fig. 3 shows this operation.

5. Experimental results

In order to comparatively evaluate the proposed algorithm to the state-of-the-art, we compare the makespan and power consumption of the G-SP algorithm with the previously proposed heuristic-based algorithms HEFT-B, HEFT-T and CPOP [5]. For this purpose, we consider two kinds of graphs as workloads, real-word task graphs and randomly generated task graphs. We use the Intel i7-3610QM processor to perform the experiments. It is a fast quad core processor based on the Ivy Bridge architecture. Intel i7-3610QM processor has a base frequency of 2.3 GHz which may increase to 3.3 GHz using the Intel Turbo Boost Technology. The proposed algorithm, G-SP, is programmed in Python and all input parameters of the algorithm are according to the values used in prior studies as reported in Table 2.

5.1. Real-world applications graphs

The first set of task graphs considered herein are task graphs for real-world problems, e.g. *Fast Fourier Transformation* (FFT) [43],

Table 2
Experimental Parameters.

Parameter	Description
The population size	Ten times the number of subtasks.
The crossover probability	0.75
The mutation probability	0.15
The stopping criteria	The results stays unchanged after 50 generation.

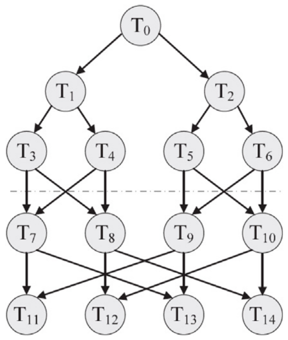

Fig. 4. Task graph of the FFT algorithm with four points.

Gaussian Elimination, *Laplace Equations*, *Sparse Matrix Solver* and *Robot Control*.

Algorithm 3 computes a one-dimensional FFT and its corresponding task graph is shown in Fig. 4 [43]. In this figure, M is an array of size j that holds the coefficients of the polynomial and F is the array of samples/results. The algorithm has two main sections: the first one performs the recursive calling and the second one is the butterfly operation. We can divide the task graph of Fig. 4 into two parts, indicated by a dashed line. The nodes above the dashed line represent the recursive call subtasks while the nodes below the dashed line represent butterfly operation subtasks. For a sample vector of size j, there are $2j-1$ recursive call subtasks and $j\log j$ butterfly operation subtasks, the computation cost of the subtasks is identical.

Algorithm 3: FFT algorithm

```
FFT ($M, \varphi$) function :
    1. $n$ = length ($M$)
    2. if ($n = 1$) return ($M$)
    3. $(F^{(0)}) = FFT((M[0], M[2], ...M[n-2]), \varphi^2)$
    4. $(F^{(0)}) = FFT((M[1], M[3], ...M[n-1]), \varphi^2)$
    5. for ($i = 1$) to $n/2 - 1$ do
    6. $F[i] = F^{(0)}[i] + \varphi^i * F^{(1)}[i]$
    7. $F[i+j/2] = F^{(0)}[i] + \varphi^i * F^{(1)}[i]$
    8. endfor
    9. return ($F$)
```

Gaussian Elimination is used to calculate the solution for a set of linear equations. The related task graph is shown in Fig. 5 [43]. In this experiment, a Gaussian Elimination algorithm with the size of 5 was used. The total number of subtasks in the graph of this set is 19, and the largest number of subtasks at the same level, showing the parallelism degree, is 5.

The Gauss–Seidel algorithm is used to solve the Laplace Equation and this method converges faster than the Jacobi algorithm [43]. The following equation shows the update procedure for the step k:

$$A_{i,j}{}^{(k)} = \frac{1}{4}[A_{(i-1),j}{}^{(k)} + A_{i,(j-1)}{}^{(k)} + A_{(i+1),j}{}^{(k-1)} + A_{i,(j+1)}{}^{(k-1)}] \qquad (9)$$

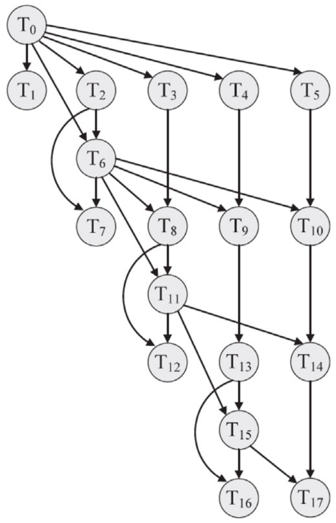

Fig. 5. Task graph of the Gaussian Elimination with 19 points.

Each value $A_{i,j}$ is updated in iteration k, from two newly generated values of iteration k and two old values generated in iteration $k-1$; therefore, this algorithm follows an updating sequence. We also used two other mostly used task graphs, Robot Control and Sparse Matrix Solver [44], in our experiments.

The proposed algorithm (G-SP) is compared with other algorithms in terms of makespan and power consumption for the above-mentioned task graphs. As shown in Figs. 6 and 7, the G-SP algorithm outperforms HEFT-B, HEFT-T, and CPOP both in makespan and power consumption. The reason for the proposed algorithm outperforming the others is that it searches a wider range of the solution space for the optimal scheduling, while HEFT-B, HEFT-T, and CPOP narrow the search space down to the small portion of the solution space by means of their initial heuristics.

5.2. Random generated applications graphs

In another experiment, we used randomly generated task graphs to evaluate the power consumption and performance of the proposed algorithm. We implemented a random graph generator which generates random graphs with different characteristics. The input parameters for this random task graph generator are the number of nodes in the graph, the computation cost of each node, and the number of successors of each node. We have generated a set of random task graphs with different characteristics and applied the considered and proposed scheduling algorithms to schedule them on a heterogeneous computing system. We have evaluated the performance of the algorithms with different numbers of subtasks and compared the G-SP with the other algorithms in terms of power consumption and makespan. The number of subtasks generated in a DAG is 50, 100 and 150. Fig. 8 shows that G-SP outperforms HEFT-B, HEFT-T and CPOP, as it was expected. This may occur because when the number of subtasks becomes higher, the problem becomes more complicated and more difficult for heuristic algorithms to find a good solution. As we can see in Fig. 8, G-SP achieves better makespan whenever the number of subtasks of the DAG increases. Fig. 9 shows that G-SP also outperforms HEFT-B, HEFT-T and CPOP in terms of power consumption. As we mentioned earlier, the proposed algorithm tries to minimize the power consumption and Fig. 9 shows that G-SP reaches this goal by consuming lower power in all case studies.

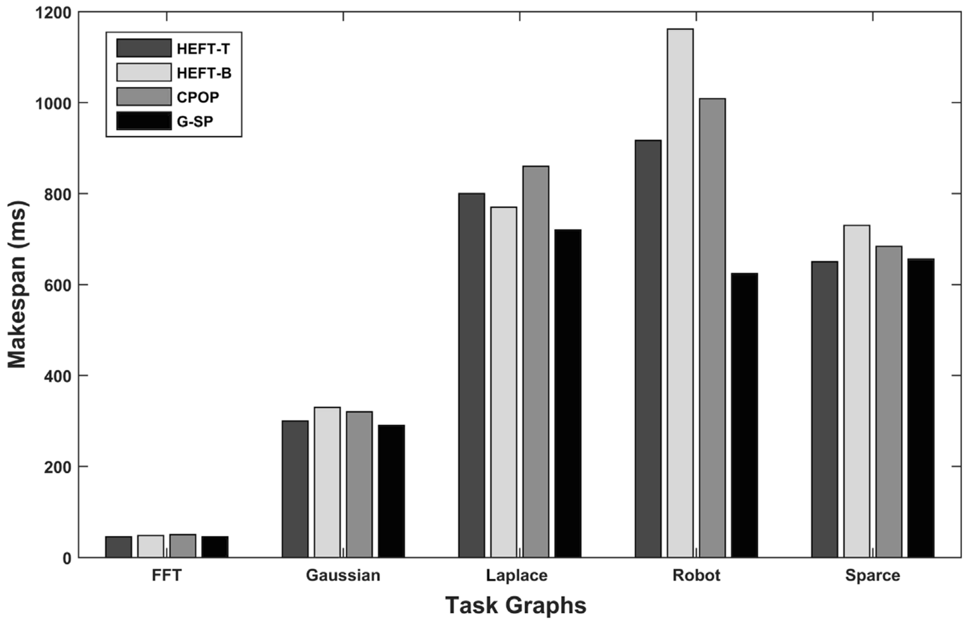

Fig. 6. Makespan of real-world applications on a system with 2 low-power (performance) and 2 high-power (performance) cores.

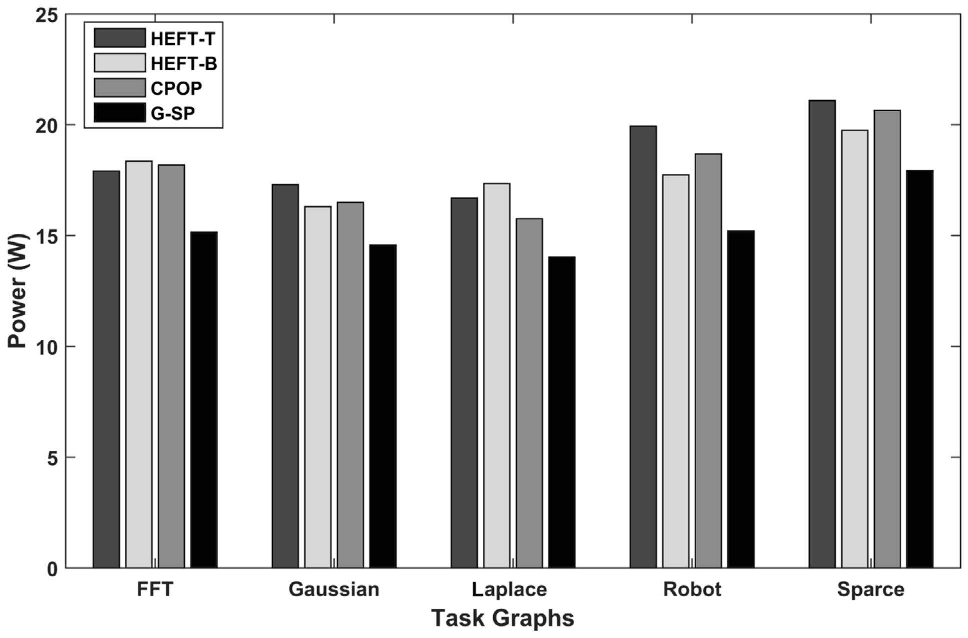

Fig. 7. Power consumption of real-world applications on a system with 2 low-power (performance) and 2 high-power (performance) cores.

In all of the above-mentioned experiments, various number of iterations is used to determine the termination condition of the G-SP algorithm. However, according to the experiments, the iteration number equal to 500 is an acceptable value for all of the task graphs studied herein. Fig. 10 represents the convergence of the G-SP for a random DAG with 100 subtasks. It shows that the convergence speed of the proposed G-SP algorithm is acceptable.

5.3. Different number of processors

Herein, we compare the makespan and power consumption of G-SP with those of benchmarks HEFT-B, HEFT-T, and CPOP, for different number of processors. Figs. 11 and 12 shows the makespan for real-world and randomly generated task graphs for a system with 2 low-power and 4 high-power cores, respectively. Figs. 13 and 14 represent the power consumption of the real-world and randomly generated task graphs on the same system. Figs. 15 and 16 show the makespan results for a system with 2 low-power and 6 high-power cores, and Figs. 17 and 18 show the power consumption for the same system. As it can be concluded from Figs. 11 to 18, G-SP outperforms HEFT-B, HEFT-T, and CPOP in terms of the makespan and power consumption in all cases.

6. Conclusion and future work

In this paper, a hybrid algorithm was proposed for multi-objective task scheduling on heterogeneous computing systems. The proposed G-SP algorithm simultaneously minimizes both the power consumption and makespan, with the help of GA and spectral partitioning. The G-SP algorithm can cover a large search

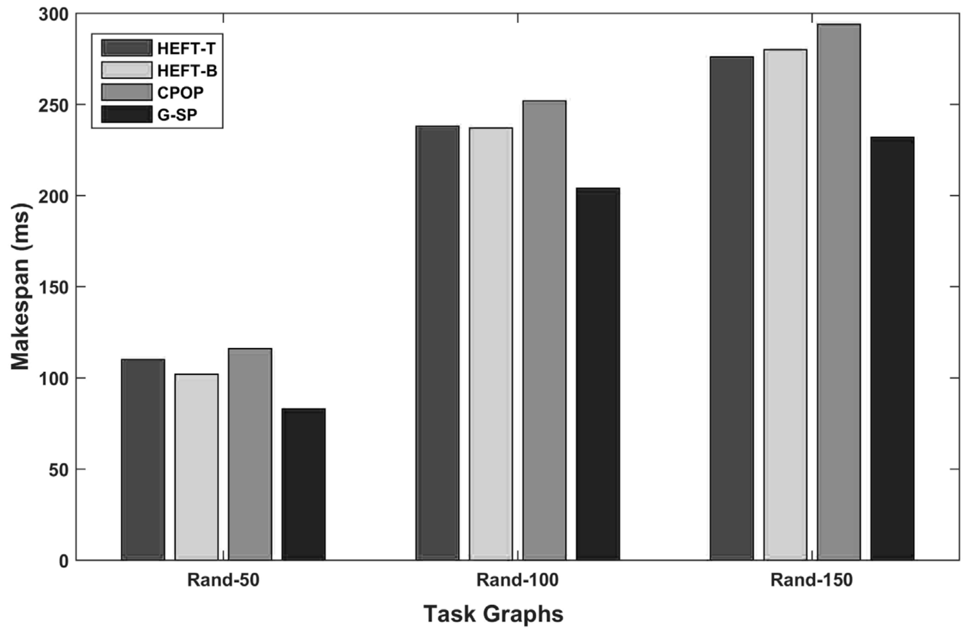

Fig. 8. Makespan of random generated applications on a system with 2 low-power (performance) and 2 high-power (performance) cores.

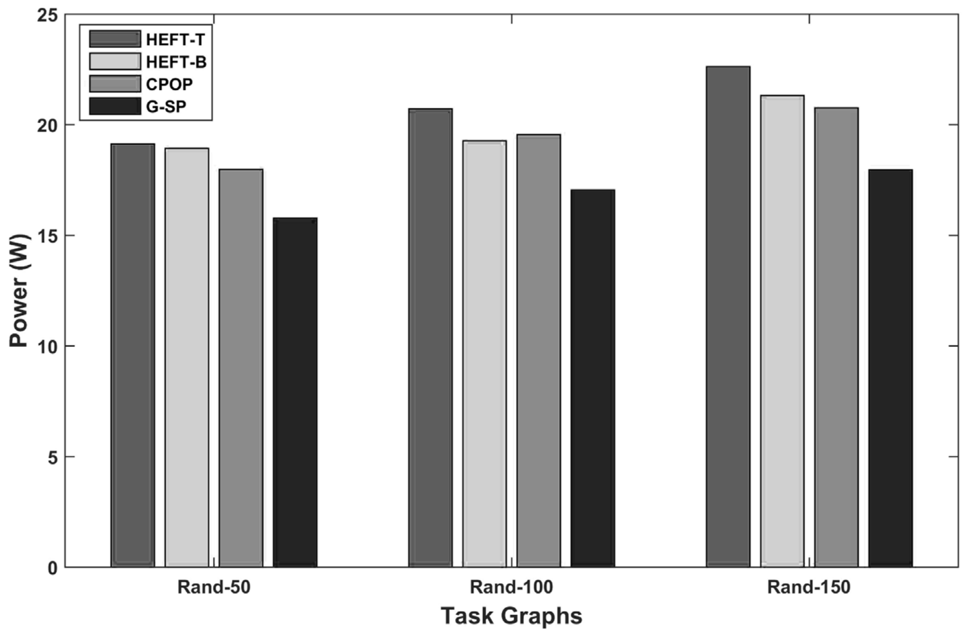

Fig. 9. Power consumption of random generated applications on a system with 2 low-power (performance) and 2 high-power (performance) cores.

space in comparison with heuristic algorithms without incurring significant computational cost. We compared the makespan and power consumption of G-SP, when applied to different number of processors, with three state-of-the-art algorithms, HEFT-B, HEFT-T, and CPOP. Experimental results showed that G-SP outperforms the other algorithms in terms of both makespan and power consumption.

One of the future prospects of this work includes applying dynamic voltage and frequency scaling (DVFS) techniques to achieve better power consumption. In this way, the voltage/frequency of a processor dynamically increases when the task graph execution needs more computational power and reduces when the low-power processors can serve the incoming tasks. The other direction to continue this work include thermal consideration for the scheduling algorithm, trying to minimize the overall thermal profile of the system.

Declaration of competing interest

No author associated with this paper has disclosed any potential or pertinent conflicts which may be perceived to have impending conflict with this work. For full disclosure statements refer to https://doi.org/10.1016/j.asoc.2020.106202.

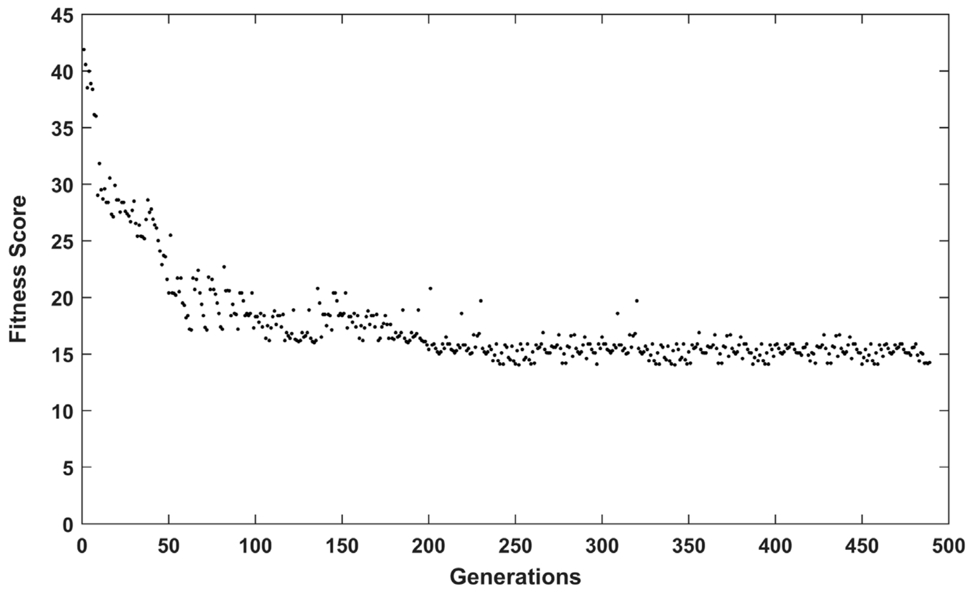

Fig. 10. The convergence of the proposed algorithm for a random DAG with 100 subtasks.

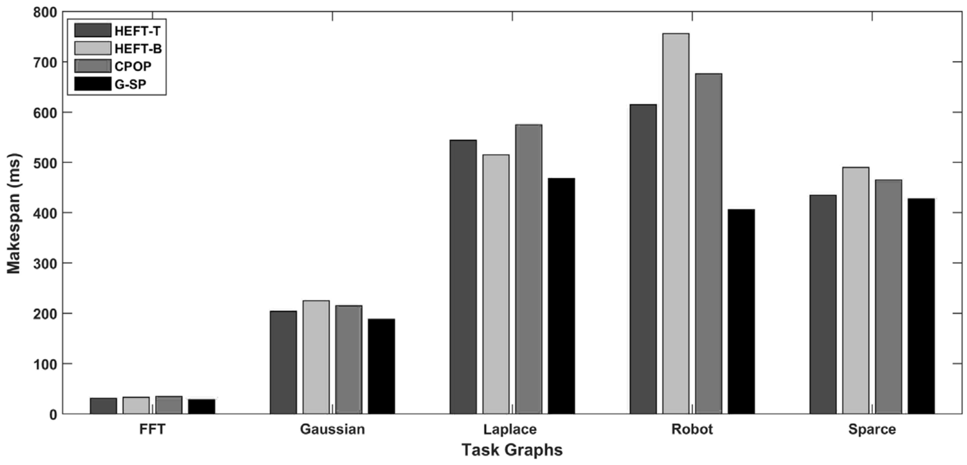

Fig. 11. Makespan of real-world applications on a system with 2 low-power (performance) and 4 high-power (performance) cores.

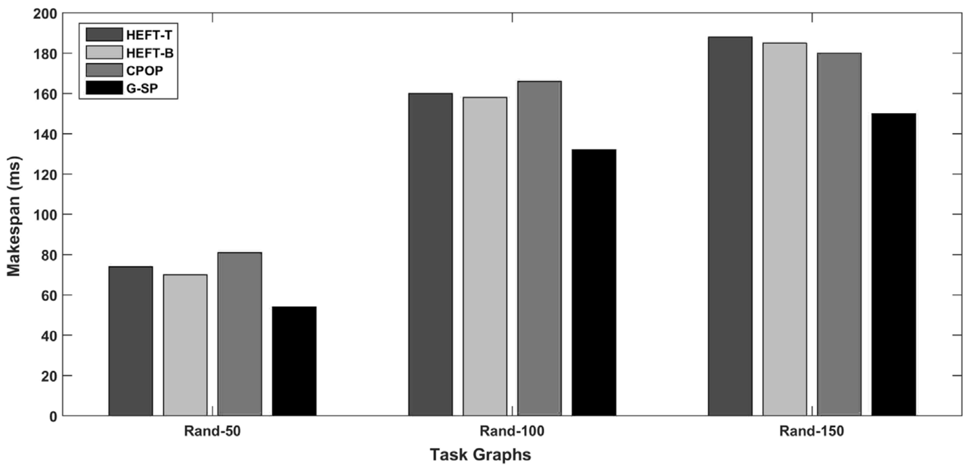

Fig. 12. Makespan of random generated applications on a system with 2 low-power (performance) and 4 high-power (performance) cores.

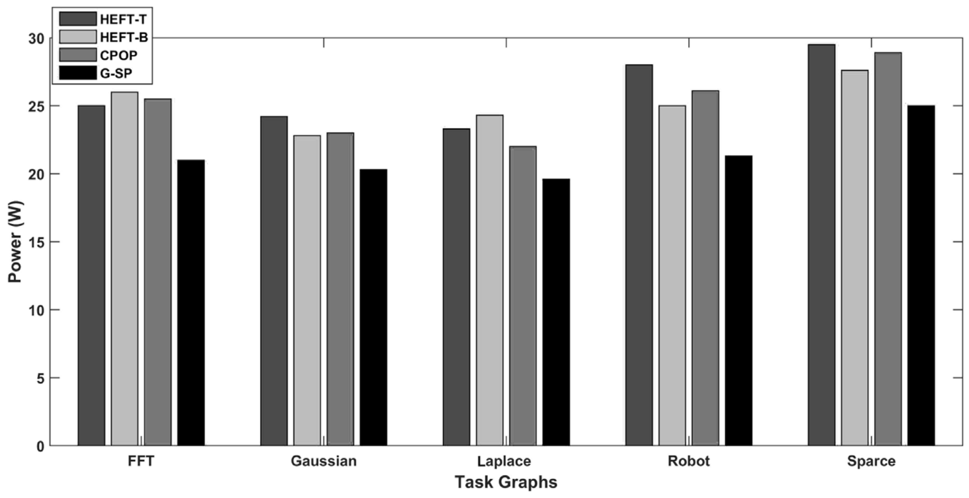

Fig. 13. Power consumption of real-world applications on a system with 2 low-power (performance) and 4 high-power (performance) cores.

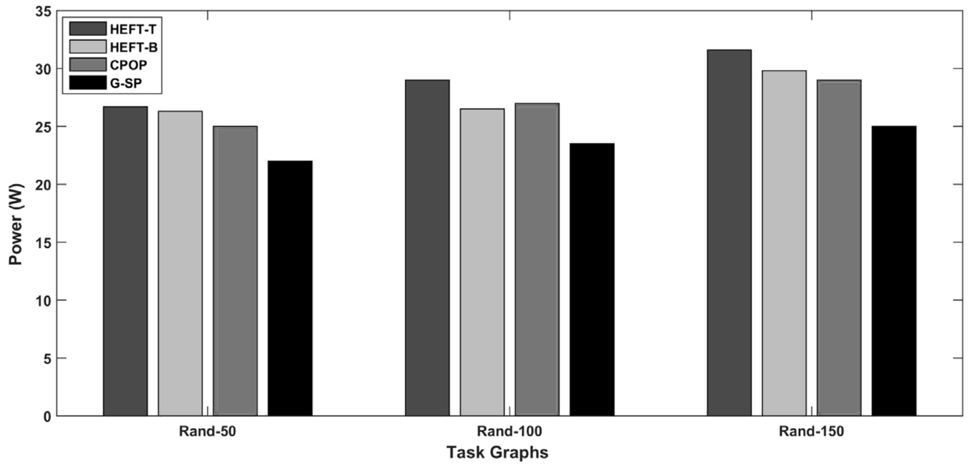

Fig. 14. Power consumption of random generated applications on a system with 2 low-power (performance) and 4 high-power (performance) cores.

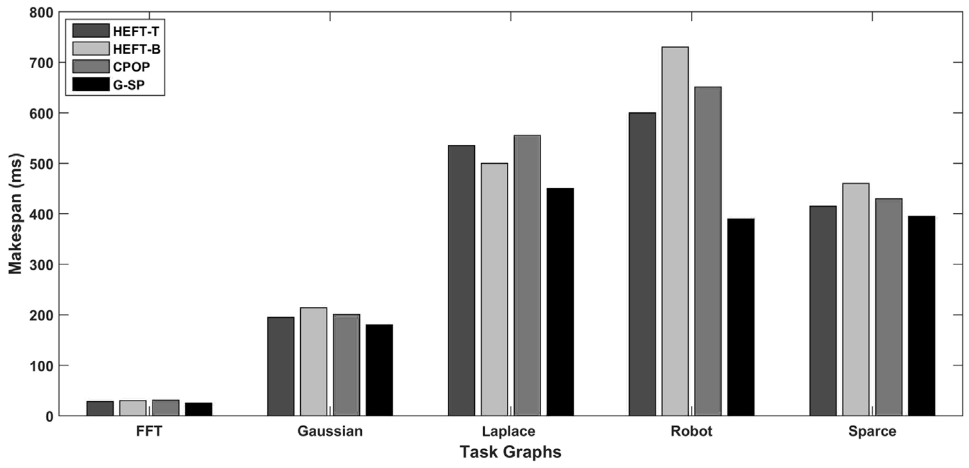

Fig. 15. Makespan of real-world applications on a system with 2 low-power (performance) and 6 high-power (performance) cores.

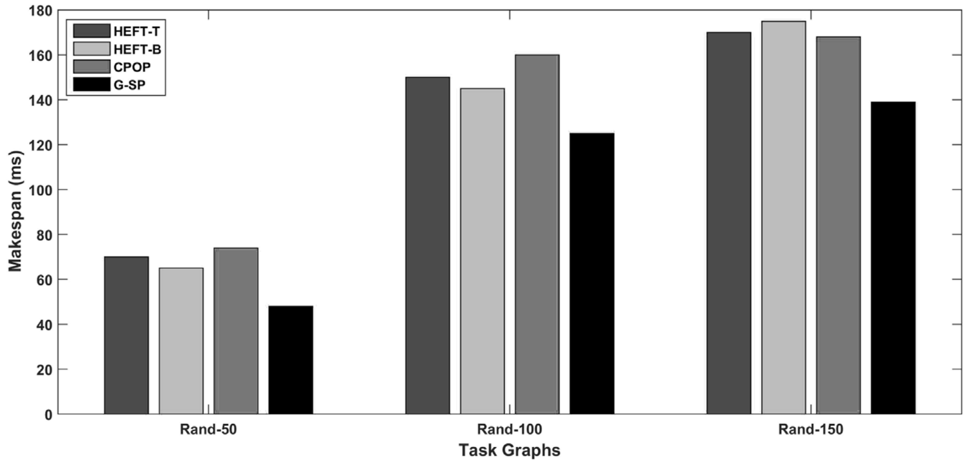

Fig. 16. Makespan of random generated applications on a system with 2 low-power (performance) and 6 high-power (performance) cores.

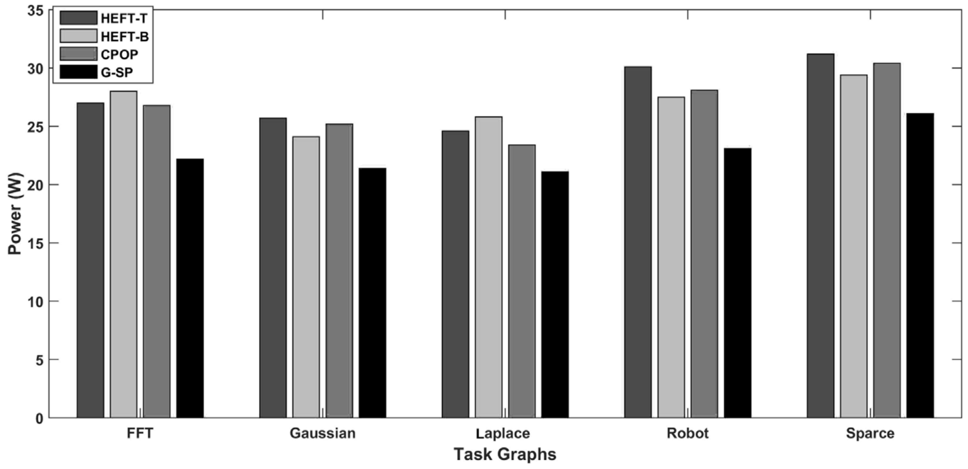

Fig. 17. Power consumption of real-world applications on a system with 2 low-power (performance) and 6 high-power (performance) cores.

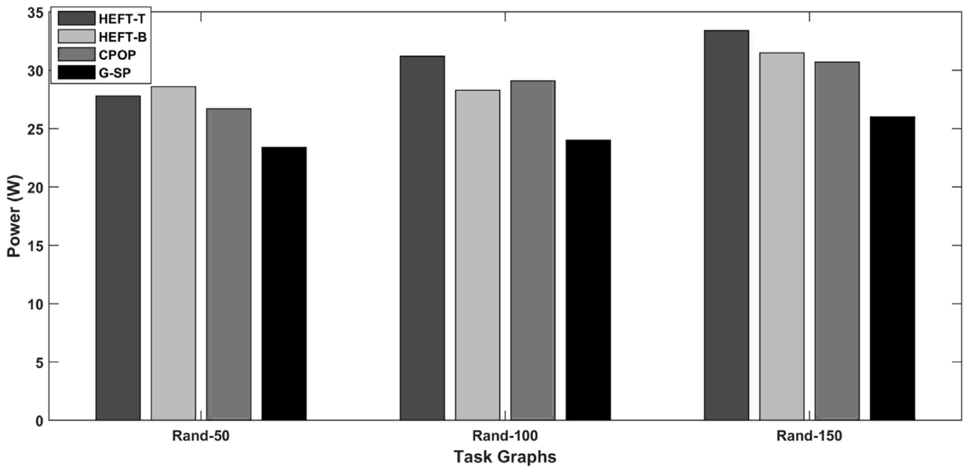

Fig. 18. Power consumption of random generated applications on a system with 2 low-power (performance) and 6 high-power (performance) cores.

CRediT authorship contribution statement

Golnaz Taheri: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Data curation, Writing - original draft. **Ahmad Khonsari:** Conceptualization, Methodology, Supervision. **Reza Entezari-Maleki:** Conceptualization, Methodology, Writing - review & editing. **Leonel Sousa:** Conceptualization, Methodology, Writing - review & editing.

Acknowledgments

This work was supported by the Iran National Science Foundation (INSF) under grant No. 98004886, and by the national funds through Fundação para a Ciência e a Tecnologia (FCT), Portugal under project number UIDB/50021/2020.

References

[1] A.K. Coskun, T.S. Rosing, K. Gross, Proactive temperature balancing for low cost thermal management in MPSoCs, in: International Conference on Computer-Aided Design, ICCAD, San Jose, CA, 2008, pp. 250–257.
[2] X. Zhong, C. Xu, System-wide energy minimization for real-time tasks: Lower bound and approximation, ACM Trans. Embed. Comput. Syst. 7 (3) (2008) 28–31.
[3] G. Taheri, A. Khonsari, R. Entezari-Maleki, M. Baharloo, L. Sousa, Temperature-aware dynamic voltage and frequency scaling enabled MPSoC modeling using stochastic activity networks, Microprocess. Microsyst. 60 (1) (2018) 15–23.
[4] J. Ullman, D. Jeffrey, NP-complete scheduling problems, J. Comput. Syst. Sci. 10 (3) (1975) 384–393.
[5] H. Topcuoglu, S. Hariri, M. Wu, Performance-effective and low-complexity task scheduling for heterogeneous computing, IEEE Trans. Parallel Distrib. Syst. 13 (3) (2002) 260–274.
[6] K. Chronaki, A. Rico, M. Casas, M. Moretó, R. Badia, E. Ayguadé, J. Labarta, M. Valero, Task scheduling techniques for asymmetric multi-core systems, IEEE Trans. Parallel Distrib. Syst. 28 (7) (2017) 2074–2087.
[7] S. AlEbrahim, I. Ahmad, Task scheduling for heterogeneous computing systems, J. Supercomput. 73 (6) (2017) 2313–2338.
[8] K. He, X. Meng, Z. Pan, L. Yuan, P. Zhou, A novel task-duplication based clustering algorithm for heterogeneous computing environments, IEEE Trans. Parallel Distrib. Syst. 30 (1) (2018) 2–14.
[9] W. Zhang, Y. Hu, H. He, Y. Liu, A. Chen, Linear and dynamic programming algorithms for real-time task scheduling with task duplication, J. Supercomput. 75 (2) (2019) 494–509.
[10] F. Ferrandi, P.L. Lanzi, C. Pilato, D. Sciuto, A. Tumeo, Ant colony heuristic for mapping and scheduling tasks and communications on heterogeneous embedded systems, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 29 (6) (2010) 911–924.
[11] M. Al-zuwaini, A. Shaker, M. Marwa, Solving machine scheduling problem under fuzzy processing time using the simulated annealing method, J. Progress. Res. Math. 14 (1) (2018) 2308–2317.
[12] S. Memeti, S. Pllana, A. Binotto, J. Kołodziej, I. Brandic, A review of machine learning and meta-heuristic methods for scheduling parallel computing systems, in: International Conference on Learning and Optimization Algorithms: Theory and Applications, Rabat, Morocco, 2018, pp. 5:1–5:6.
[13] A.S. Pillai, K. Singh, V. Saravanan, A. Anpalagan, I. Woungang, L. Barolli, A genetic algorithm-based method for optimizing the energy consumption and performance of multiprocessor systems, Soft Comput. 22 (10) (2018) 3271–3285.
[14] O. Sinnen, L.A. Sousa, Communication contention in task scheduling, IEEE Trans. Parallel Distrib. Syst. 16 (6) (2005) 503–515.
[15] K. Qinma, H. Hong, A novel discrete particle swarm optimization algorithm for meta-task assignment in heterogeneous computing systems, Microprocess. Microsyst. 35 (1) (2011) 10–17.
[16] Z. Jia, J. Yan, J.Y.T. Leung, K. Li, H. Chen, Ant colony optimization algorithm for scheduling jobs with fuzzy processing time on parallel batch machines with different capacities, Appl. Soft Comput. 75 (1) (2019) 548–561.
[17] K. HyunJin, K. Sungho, Communication-aware task scheduling and voltage selection for total energy minimization in a multiprocessor system using ant colony optimization, Inform. Sci. 181 (18) (2011) 3995–4008.
[18] P. Marimuthu, R. Arumugam, J. Ali, Hybrid metaheuristic algorithm for real time task assignment problem in heterogeneous multiprocessors., Int. Arab J. Inf. Technol. 15 (3) (2018) 445–453.
[19] N. Edward, J. Elcock, Task scheduling in heterogeneous multiprocessor environments–An efficient ACO-based approach, Indones. J. Electr. Eng. Comput. Sci. 10 (1) (2018) 320–329.
[20] U. Srikanth, U. Maheswari, S. Palaniswami, A. Siromoney, Task scheduling using probabilistic ant colony heuristics, Int. Arab J. Inf. Technol. 13 (4) (2016) 375–379.
[21] T. Ebi, A. Faruque, M. Abdullah, J. Henkel, TAPE: thermal-aware agent-based power economy for multi/many-core architectures, in: International Conference on Computer-Aided Design, ICCAD, San Jose, CA, 2009, pp. 302–309.
[22] X. Chen, G. Reinelt, G. Dai, M. Wang, Priority-based and conflict-avoidance heuristics for multi-satellite scheduling, Appl. Soft Comput. 69 (1) (2018) 177–191.
[23] A.J. Page, T.M. Keane, T.J. Naughton, Multi-heuristic dynamic task allocation using genetic algorithms in a heterogeneous distributed system, IEEE Trans. Parallel Distrib. Syst. 70 (7) (2010) 758–766.
[24] V. Priya, C.S. Kumar, R. Kannan, Resource scheduling algorithm with load balancing for cloud service provisioning, Appl. Soft Comput. 76 (1) (2019) 416–424.
[25] M.T. Younis, S. Yang, Hybrid meta-heuristic algorithms for independent job scheduling in grid computing, Appl. Soft Comput. 72 (1) (2018) 498–517.
[26] S. Basu, M. Karuppiah, K. Selvakumar, K. Li, S. Islam, M. Hassan, M.B.A. Zakirul, An intelligent/cognitive model of task scheduling for IoT applications in cloud computing environment, Future Gener. Comput. Syst. 88 (2018) 254–261.
[27] Y. Yun, E.J. Hwang, Y.H. Kim, Adaptive genetic algorithm for energy-efficient task scheduling on asymmetric multiprocessor system-on-chip, Microprocess. Microsyst. 66 (1) (2019) 19–30.
[28] A.S. Pillai, K. Singh, V. Saravanan, A. Anpalagan, I. Woungang, L. Barolli, A genetic algorithm-based method for optimizing the energy consumption and performance of multiprocessor systems, Soft Comput. 22 (10) (2018) 3271–3285.
[29] S. Yuan, G. Deng, Q. Feng, P. Zheng, T. Song, Multi-objective evolutionary algorithm based on decomposition for energy-aware scheduling in heterogeneous computing systems, J.UCS 23 (7) (2017) 636–651.
[30] Y. Xie, W. Hung, Temperature-aware task allocation and scheduling for embedded multiprocessor systems-on-chip (MPSoC) design, J. VLSI Signal Process. Syst. Signal Image Video Technol. 45 (3) (2006) 177–189.
[31] S. Xu, I. Koren, C. Krishna, Thermal-aware task allocation and scheduling for heterogeneous multi-core cyber-physical systems, in: International Conference on Embedded Systems, Cyber-Physical Systems, and Applications, ESCS'16, Las Vegas, USA, 2016, pp. 10–16.
[32] H. Chu, Y. Kao, Y. Chen, Adaptive thermal-aware task scheduling for multi-core systems, J. Syst. Softw. 99 (1) (2015) 155–174.
[33] L.Y. Tseng, S.C. Liang, A hybrid metaheuristic for the quadratic assignment problem, Comput. Optim. Appl. 34 (1) (2006) 85–113.
[34] G. Buttazzo, E. Bini, Y. Wu, Partitioning real-time applications over multicore reservations, IEEE Trans. Ind. Inf. 7 (2) (2011) 302–315.
[35] D. Li, J. Wu, Minimizing energy consumption for frame-based tasks on heterogeneous multiprocessor platforms, IEEE Trans. Parallel Distrib. Syst. 26 (3) (2015) 810–823.
[36] K. Cao, J. Zhou, P. Cong, L. Li, T. Wei, M. Chen, S. Hu, X.S. Hu, Affinity-driven modeling and scheduling for makespan optimization in heterogeneous multiprocessor systems, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 1 (1) (2018) 1.
[37] W. Donath, A. Hoffman, Lower bounds for the partitioning of graphs, IBM J. Res. Dev. 17 (5) (1973) 420–425.
[38] G. Taheri, M. Ayati, L. Wong, C. Eslahchi, Two scenarios for overcoming drug resistance by co–targeting, Int. J. Bioinform. Res. Appl. 11 (1) (2015) 72–89.
[39] G. Taheri, M. Habibi, L. Wong, C. Eslahchi, Disruption of protein complexes, J. Bioinform. Comput. Biol. 11 (03) (2013) 254–261.
[40] D. Cohen-Steiner, W. Kong, C. Sohler, G. Valiant, Approximating the spectrum of a graph, in: International Conference on Knowledge Discovery & Data Mining, London, United Kingdom, 2018, pp. 1263–1271.
[41] C.J. Alpert, A.B. Kahng, S.-Z. Yao, Spectral partitioning with multiple eigenvectors, Discrete Appl. Math. 90 (1–3) (1999) 3–26.
[42] B. Hendrickson, R. Leland, An improved spectral graph partitioning algorithm for mapping parallel computations, SIAM J. Sci. Comput. 16 (2) (1995) 452–469.
[43] I. Ahmad, Y. Kwok, M. Wu, W. Shu, CASCH: a tool for computer-aided scheduling, IEEE Concurr. 8 (4) (2000) 21–33.
[44] The standard task graph set, 2019, URL http://www.kasahara.elec.waseda.ac.jp/schedule/.