The Journal of Supercomputing (2021) 77:2689–2712
https://doi.org/10.1007/s11227-020-03369-w

Scalable parallel implementation of migrating birds optimization for the multi-objective task allocation problem

Dindar Öz[1] · Işıl Öz[2]

Published online: 1 July 2020

Abstract

As the distributed computing systems have been widely used in many research and industrial areas, the problem of allocating tasks to available processors in the system efficiently has been an important concern. Since the problem is proven to be NP-hard, heuristic-based optimization techniques have been proposed to solve the task allocation problem. Particularly, the current cloud-based systems have been grown massively requiring multiple features like lower cost, higher reliability, and higher throughput; therefore, the problem has become more challenging and approximate methods have gained more importance. Migrating birds optimization (MBO) algorithm offers successful solutions, especially for quadratic assignment problems. Inspired by the movement of the birds, it exhibits good results by its population-based approach . Since the algorithm needs to deal with many individuals in the population, and the neighbor solution generation phase takes substantial time for large problem instances, we need parallelism to have execution time improvements and make the algorithm practical for large-scale problems. In this work, we propose a scalable parallel implementation of the MBO algorithm, PMBO, for the multi-objective task allocation problem. We redesigned the implementation of the MBO algorithm so that its computationally heavy independent tasks are executed concurrently in separate threads. We compare our implementation with three parallel island-based approaches. The experimental results demonstrate that our implementation exhibits substantial solution quality improvements for difficult problem instances as the computing resources, namely parallelism, increase. Our scalability analysis also presents that higher parallelism levels offer larger solution improvement for the PMBO over the island-based parallel implementations on very hard problem instances.

Keywords Parallel algorithm · Combinatorial optimization · Task allocation problem · Migrating birds optimization

Işıl Öz
isiloz@iyte.edu.tr

Extended author information available on the last page of the article

1 Introduction

The problem of task allocation in heterogeneous distributed systems tries to assign each task partitioned from an application to the processors in the system. Utilizing the parallel execution units for computation-intensive parallel and distributed applications plays an important role in the efficiency of executing an application on those systems. The problem has been tackled with many different objectives such as maximizing the overall performance of the system, minimizing the probability of the errors during the execution, or maximizing the load balance of the processors in terms of the memory or the workload. Even with a single objective, the task allocation problem is an NP-hard problem for which achieving the optimal solution requires impractical times for most of the real-world instances. Therefore, many studies have been focusing on applying suboptimal algorithms consisting of both problem-specific greedy heuristics and generic metaheuristic solutions for the multi-objective task allocation problem (MOTAP) [7, 10, 17, 39, 43].

By the increase in the usage of cloud-based systems for many industrial and scientific applications, the distributed systems have gained importance [45]. Additionally, the size of those systems has risen to solve more difficult scientific problems and support large companies. With a high number of task execution requests in those large-scale systems, allocating tasks onto available processors has been a more complex problem. Particularly, for a highly available service running on multiple containers, the allocation of system resources becomes more critical. Therefore, the demand for a good solution for the task allocation problem has increased.

Modern hardware systems offer high performance with their parallel execution units [27, 40]. Moreover, parallel programming models provide practical implementation for computationally intensive applications to exploit the parallelism in modern systems. Consequently, the algorithms designed to solve difficult problems started to utilize parallel computation more and more in order to achieve better performance. In this context, metaheuristic algorithms, due to their iterative nature, containing independent execution blocks, are good candidates for applying parallel computation.

In this paper, we design and analyze two different parallelization models of migrating birds optimization algorithm (MBO) for MOTAP. MBO is a population-based metaheuristic algorithm that is proven to be successful on quadratic assignment problem [9]. In the first model, we apply the island model parallelization to the MBO and implement IMBO. In the second model, we present a scalable parallel implementation for the migrating birds optimization algorithm with problem-specific neighboring heuristics. We design and implement a parallel algorithm (PMBO) to reduce the execution time by parallelizing the time-consuming neighboring function calculations. We compare our algorithm performance with island model implementations of migrating birds optimization, genetic algorithm, and simplified swarm optimization algorithm [43]. Our results yield that our parallel implementation exhibits substantial solution quality improvements for very difficult problem instances as the computing resources in the parallel system increase.

The remainder of this paper is organized as follows: Sect. 2 presents the related work on both task allocation problem and parallelization methods for metaheuristic algorithms. Section 3 provides a general description and a formal definition of the multi-objective task allocation problem and the basics of the MBO algorithm. Section 4 presents our parallel implementation by introducing general parallelism constructs. Then, the experimental results are outlined in Sect. 5. Finally, in Sect. 6, we summarize the work with some conclusive remarks.

2 Related work

Task allocation problem (TAP) has been studied in the literature by considering especially the system performance and reliability [5, 9, 12, 21, 31, 32, 35, 36, 44]. Besides the system performance, which is critical for the execution of real-time applications on distributed systems, the system reliability becomes a crucial concern due to high machine and network failure rates in large-scale systems [36]. In safety-critical systems such as power plants, being free from errors and malfunctions becomes the primary objective.

Many research studies have explored the task allocation problem by starting with the work of Stone [37]. While some methods consider performance criteria by minimizing the total sum of execution and communication time [7, 10, 17, 39, 43], or target higher system reliability by minimizing the probability of failure [5, 15, 16, 36], some of them try to optimize both objectives solving the multi-objective task allocation problem [12, 21, 32, 44]. Since the problems with both objectives are NP-hard, many suboptimal heuristic algorithms have been proposed. Mathematical approach-based exact solutions have achieved reasonable times for only smaller instances of the problem [10].

Use of evolutionary metaheuristic optimization algorithms and stochastic search algorithms has been popular for the task allocation problem [5, 9, 11, 13–15, 26, 32, 41, 42]. As the evolutionary algorithms target harder and larger problems, the execution time required to find an acceptable solution increases. Therefore, the parallel implementations have been proposed in order to decrease the execution times [1, 2]. While the island model achieves performance improvement by executing multiple populations in parallel [18, 19, 28], the parallelization of neighborhood search procedures (e.g., mutation or crossover operations) decreases the execution time of the algorithms by a larger amount.

Island model partitions the population into subpopulations in which serial computations are performed in isolation [38]. To introduce diversity into subpopulations, the model exchanges a set of individuals by migrating them into other subpopulations. While each island performs independently in parallel, the migration among the islands provides possible enhancement on solution quality [18]. Alba and Troya [3] propose an island model for a set of optimization problems and conduct an experimental study on a distributed system to analyze the effect of the migration frequency on runtime and diversity. Beside distributed systems, the island model has also been implemented for GPU architectures. Luong et al. [24] propose three different island model schemes for

Weierstrass–Mandelbrot function optimizations. While the first scheme evaluates only the populations in different GPU thread blocks, the others are based on a fully distributed model by offloading the overall computation from CPU to the GPU cores. Pospichal et al. [33] also present a GPU-based island model for parallel execution of a set of benchmark functions. While the island model is successfully implemented in parallel systems to decrease the execution time, it also improves the solution quality without parallel execution support. Limmer and Fey [20] present a comparison study for island model and the global parallelization-based evolutionary algorithms for multi-core CPUs, clusters, GPUs, and grid platforms. Al-Betar et al. [4] propose an island model for the flower pollination algorithm (FPA) which is a swarm-based evolutionary algorithm. Even the proposed model is not implemented in parallel, it can produce better results than the regular FPA by maintaining diversity. Liu and Wang [22] present a scalable parallel genetic algorithm (PGAP) to exploit massively parallel high-end computing resources for solving large problem instances of the generalized assignment problem (GAP). To be able to improve performance by overlapping communication and computation, the authors propose an asynchronous migration strategy for efficient migrations among populations. While the asynchronous communication leads to a reduction in the solution quality due to the loss of good solutions obtained from migration, the technique provides higher speedups in a large-scale parallel computing environment with acceptable solutions.

Luo et al. [23] present a parallel bees algorithm by considering several parallelization concerns. The proposed method not only partitions the colonies (populations) into subcolonies (subpopulations) but also performs multiple instances of local searches in different processing units simultaneously for parallelism.

Randall and Lewis [34] propose parallel implementations of ant colony optimization (ACO) based on parallel ants and multiple colonies and conduct experiments for TSP problem instances in a distributed memory architecture. While Chu et al. [8] present a parallel ACO scheme for protein structure prediction, Middendorf et al. [25] introduce multi-colony approach to solve traveling salesperson problem and the quadratic assignment problem. Delevacq et al. [34] adapt parallel ACO for GPU architectures.

ParadisEO [6] presents a framework for the development of parallel and distributed metaheuristic algorithms. It supports different parallel models including parallel distributed evolutionary algorithms and parallel local searches. The developers can implement their own parallel algorithms by using object-oriented components based on standard distributed memory or shared memory libraries.

Duman et al. [9] propose a novel metaheuristic optimization algorithm, namely migrating bird optimization (MBO) for quadratic assumption problem, and achieve promising results. The MBO is further improved by Pan and Dong [31], and Oz [30] by introducing problem-specific neighboring heuristics. While the proposed neighboring functions improve the solution quality, the execution time of the algorithms increases due to the higher computational effort of the functions.

3 Background

3.1 Task allocation problem

In this work, we consider multi-objective task allocation problem that tries to find the efficient allocation of parallel application tasks onto processors of a distributed system. The objectives to be achieved are minimizing the assignment cost and maximizing the system reliability. We use the problem statement given in [5, 13, 44] to formulate the task allocation problem.

3.1.1 System model

The heterogeneous distributed computing system consists of N processors $(P_1, P_2, \dots, P_N)$, and each processor (P_n) has the following computation and reliability attributes:

- C_n: amount of computation resource
- M_n: amount of memory resource
- λ_n: failure rate

We assume that the processors are connected by an interconnection network, and the communication link between two processors (P_n, P_m) has the following attributes:

- DTR_{nm}: data transfer rate between processors
- μ_{nm}: failure rate of the communication path between processors

We further assume that we have a parallel application including K tasks $(T_1, T_2, \dots, T_K)$, as shown in Fig. 1. While the tasks may interact each other, which incurs communication overhead for the tasks executing in different processors, our model does not contain information about precedence relations among the tasks [13]. Each application task (T_k) has the following attributes:

- c_k: computation resource requirement
- m_k: memory resource requirement

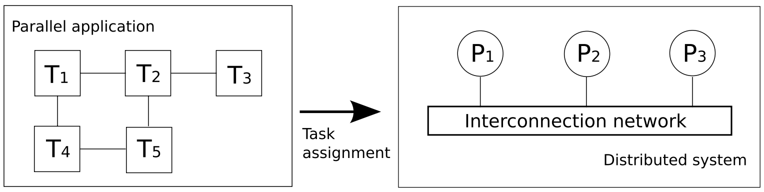

Fig. 1 An example parallel application with several tasks and a target distributed system

Each task pair (T_k, T_l) may communicate each other with an amount of data, and this inter-task communication can be characterized as follows:

– D_{kl}: incurred communication time between tasks T_k and T_l

We assume that the execution time for one task may be different on different processors in a heterogeneous system, and those execution times are known/predicted before execution. ET_{kn} represents the expected execution time of a task T_k on the processor P_n, for each task and processor pair.

The task allocation scheme targets to find a task allocation represented by X, the assignment of K tasks onto N processors, that minimizes the assignment cost and maximizes the system reliability at the same time while satisfying memory and computation resource constraints, where $X_{kn} = 1$ if task T_k is assigned to processor P_n, and $X_{kn} = 0$ otherwise.

3.1.2 Assignment cost

For evaluating the cost for task assignment, we consider the execution and communication times. Given a task allocation X, since the execution time of all tasks in processor P_n is $\sum_{k=1}^{K} X_{kn}ET_{kn}$, the execution cost of all processors is the total execution time of the application, which can be computed as follows:

$$C_{\text{exec}}(X) = \sum_{n=1}^{N}\sum_{k=1}^{K} X_{kn}ET_{kn}.$$

Similarly, since the required time for communications between processors P_n and P_m is $\sum_{k=1}^{K-1}\sum_{l=k+1}^{K} X_{kn}X_{lm}(D_{kl}/DTR_{nm})$, the total system communication cost can be computed as follows:

$$C_{\text{comm}}(X) = \sum_{n=1}^{N-1}\sum_{m>n}\sum_{k=1}^{K-1}\sum_{l=k+1}^{K} X_{kn}X_{lm}(D_{kl}/DTR_{nm}).$$

Then, the assignment cost defined as the sum of the execution and communication costs can be computed as follows:

$$\begin{aligned}\mathbf{C}(X) &= C_{\text{exec}}(X) + C_{\text{comm}}(X)\\ &= \sum_{n=1}^{N}\sum_{k=1}^{K} X_{kn}ET_{kn} + \sum_{n=1}^{N-1}\sum_{m>n}\sum_{k=1}^{K-1}\sum_{l=k+1}^{K} X_{kn}X_{lm}(D_{kl}/DTR_{nm}).\end{aligned} \tag{1}$$

3.1.3 System reliability cost

In order to quantify the reliability of a distributed system, we need to consider the successful execution of all tasks in the system. For successful task execution, we can assume that all processors and communication links are working correctly during execution [36]. Since the reliability of a processor P_n during time t is $e^{-\lambda_n t}$, and the reliability of a processor P_n for a given task allocation X is $e^{-\lambda_n \sum_{k=1}^{K} X_{kn} ET_{kn}}$, the reliability of all processors can be computed as follows:

$$R_{\text{exec}}(X) = \prod_{n=1}^{N} e^{-\lambda_n \sum_{k=1}^{K} X_{kn} ET_{kn}}.$$

The reliability of a communication path between processors P_n and P_m is $e^{-\mu_{nm} \sum_{k=1}^{K-1} \sum_{l=k+1}^{K} X_{km} X_{ln} (D_{kl}/DTR_{mn})}$; the reliability of all communication paths can be computed as follows:

$$R_{\text{comm}}(X) = \prod_{n=1}^{N-1} \prod_{m>n} e^{-\mu_{nm} \sum_{k=1}^{K-1} \sum_{l=k+1}^{K} X_{kn} X_{lm} (D_{kl}/DTR_{nm})}.$$

Then, the system reliability that all involved processors and communication links can be computed as follows:

$$\begin{aligned} R(X) &= R_{\text{exec}}(X) \times R_{\text{comm}}(X) \\ &= \prod_{n=1}^{N} e^{-\lambda_n \sum_{k=1}^{K} X_{kn} ET_{kn}} \times \prod_{n=1}^{N-1} \prod_{m>n} e^{-\mu_{nm} \sum_{k=1}^{K-1} \sum_{l=k+1}^{K} X_{kn} X_{lm} (D_{kl}/DTR_{nm})}. \end{aligned}$$

Since the system reliability ($R(X)$) can be described by the system reliability cost ($RC(X)$) as $R(X) = e^{-RC(X)}$, the system reliability cost can be computed as follows:

$$\mathbf{RC}(X) = \sum_{n=1}^{N} \sum_{k=1}^{K} \lambda_n X_{kn} ET_{kn} + \sum_{n=1}^{N-1} \sum_{m>n} \sum_{k=1}^{K-1} \sum_{l=k+1}^{K} \mu_{nm} X_{kn} X_{lm} (D_{kl}/DTR_{nm}). \tag{2}$$

3.1.4 Multi-objective formulation

Using the formulations for the assignment cost (Eq. 1) and the system reliability cost (Eq. 2), the mathematical formulation of task allocation problem is as follows:

$$\text{Minimize} \quad Z(X) = C(X) + RC(X) \tag{3}$$

Subject to

$$\sum_{n=1}^{N} X_{kn} = 1 \quad \forall k = 1, 2, \dots, K \tag{4}$$

$$\sum_{k=1}^{K} m_k X_{kn} \leq M_k \quad \forall n = 1, 2, \dots, N \tag{5}$$

$$\sum_{k=1}^{K} c_k X_{kn} \leq C_k \quad \forall n = 1, 2, \ldots, N \tag{6}$$

$$X_{kn} \in \{0, 1\} \forall n, k. \tag{7}$$

Equation 3 represents the combined objective function including assignment cost and reliability cost minimization. Equation 4 states that each task should be assigned to exactly one processor. Equation 5, memory constraint, states that the total memory required by all tasks assigned to processor P_n does not exceed the available memory size of the processor. Equation 6, computation resource constraint, provides that the processing load required by all tasks assigned to processor P_n does not exceed the available processing load. The constraint in Eq. 7 guarantees that X_{kn} is binary variable.

The model defines a 0–1 quadratic programming problem which is known as NP-hard [13, 44], which implies that the optimum solution is not practical, especially for big problem sizes.

3.2 Migrating birds optimization algorithm

Getting inspired by the behavioral pattern exhibited by migrating bird flocks, Duman et al. [9] design a population-based metaheuristic algorithm, migrating birds optimization (MBO). Their work and the following studies show that it is particularly successful in quadratic assignment problems [9, 29–31]. The algorithm basically maintains a number of solutions carried by a flock of n birds that are logically organized in V formation as migrating flocks of birds. In this context, each bird (solution) follows the bird placed at its immediate front in the flock. This is achieved by sharing a number of neighbor solutions between the birds at each iteration. The bird in front of all other birds is called the leader bird. Imitating the natural migrating flock behavior, the leader bird is changed after a number of iterations and the old leader goes at the back of the flock, allowing the exploration of different parts of the search space. The iterations are repeated until a predefined condition is satisfied. The algorithm, listed in Algorithm 1, starts with the initialization phase that includes generating the initial solutions of the flock. This is followed by MBO iterations which consist of three stages:

- *Leader improvement* A local search is applied to improve the leader solution by generating its k neighbor according to a move operator. If any of the neighbors improves the leader, then the leader is replaced. This is followed by sharing the remaining x neighbor solutions with the bird next to the leader.
- *Follower improvement* Other birds in the flock are tried to be improved in a similar way as the leader. Each bird, combined with the x solutions shared from the bird it is following, generates $k - x$ new solutions using the move operator. If the best solution improves the bird, the bird updates its current solution. Like the leader bird, each bird in the flock gives their best x neighbor solutions to their followers.

– *Leader change* Mimicking the flock behavior in nature, after a certain number of iterations (m) the leader bird switches to one end of the flock and a new bird becomes leader.

Algorithm 1 MBO algorithm [9].

```
1: Generate $n$ initial solutions
2: while not termination do
3:     for each replication (1 to $m$) do
4:         Generate and evaluate $k$ neighbors for the leader
5:         if there is an improvement then
6:             Update the leader
7:         end if
8:         Move the $x$ leader neighbors to the other solutions
9:         for each follower in the flock do
10:            Generate and evaluate $k$ neighbors
11:            if there is an improvement then
12:                Update the follower
13:            end if
14:        end for
15:    end for
16:    Move the leader to the end and assign a solution as the leader
17: end while
```

3.2.1 Solution representation

Following the previous studies [5, 13, 21, 44], integer vector representation is used where the vector $P = (p_1, p_2, \dots p_K)^T$ stores the processors assigned to each task. In this context, $p[i] = j$ denotes $X_{ij} = 1$ where X_{ij}s are the Boolean variables used in the formal definition of the problem.

3.2.2 Objective function calculation

To avoid infeasible solutions violating the constraints stated in the inequalities (5) and (6), a penalty value is added to the original objective function of the problem (i.e., Eq. 3). The penalty value for a solution P is calculated as follows:

$$\text{Penalty}\,(P) = \sum_{n=1}^{N} \left(max\left(0, \sum_{i|p[i]=n} m_i - M_n\right) + max\left(0, \sum_{i|p[i]=n} c_i - C_n\right)\right). \tag{8}$$

Therefore, the fitness value of a solution P is:

$$\text{Fitness}\,(P) = Z(P) + \omega * \text{Penalty}\,(P) \tag{9}$$

where ω denotes the penalty coefficient used for scaling purposes, as applied in other studies [13, 30].

3.2.3 Problem-specific neighboring for MBO: GR-MR

A problem-specific neighboring function proposed in [30] is used as the neighboring function of the MBO algorithm. This function, Greedy Reassignment Maximum Release (GR-MR), basically tries to reassign a randomly selected task to the processor which causes the maximum amount of fitness improvement. If no improvement possible for a number of attempts, then GR-MR switches to the maximum release phase, and in this phase, it tries to release the mostly loaded processor by removing a task from that processor in order to enable further profitable reassignments. The details of the neighboring function can be found in [30].

4 Parallel MBO

In this study, we redesigned the implementation of the MBO algorithm so that its computationally heavy independent tasks are executed concurrently in separate threads. The largest portion of the execution time in MBO is the generation of the neighbor solutions while trying to improve each bird in the flock. Mainly, the GR-MR function is executed k times for the leader and $n * (k - x)$ times for the other birds. In the original MBO design, the bird improvement is performed sequentially making the birds waiting for each other. This can be justified to an extent, as each bird in the flock requires a number of shared solutions from the bird it is following, making the improvement process nonindependent. However, if the neighbor generation and neighbor sharing are separated from each other, the former operation, which is an independent task, can be performed in parallel. In our parallel MBO implementation (PMBO), at each iteration of the algorithm, each bird generates a number of neighbor solutions simultaneously. After the neighbors are calculated, each bird performs neighbor sharing and any improvement that may possibly be achieved. Figures 2 and 3 depict the design difference between MBO and PMBO algorithms. As shown in Fig. 3, the generation of the neighbors is performed in parallel for all of the birds in the flock, and then they wait for each other before trying to improve via generated and shared neighbors.

To compare the performance of the PMBO, we implement the genetic algorithm (GA), migrating birds optimization (MBO) algorithm [9], and simplified swarm optimization (SSO) algorithm [43]. We also implement the parallel versions of GA, MBO, and SSO algorithms based on the island model (IGA, IMBO, and ISSO, respectively). As mentioned in Sect. 1, the island model partitions the population into subpopulations in which serial computations are performed in isolation [38]. Both serial and parallel executions of the island model-based implementations provide an increase in the population diversity, while the parallel execution reduces the execution time substantially. Figure 4 presents the structure of the IMBO, where multiple MBO populations run concurrently, and they change the leaders for some time migration periods. We implement IGA, IMBO, and ISSO and execute them in parallel with the same number of threads as PMBO executions. All implementations including serial and parallel versions employ GR-MR function as their mutation operator/neighboring function if they are using any.

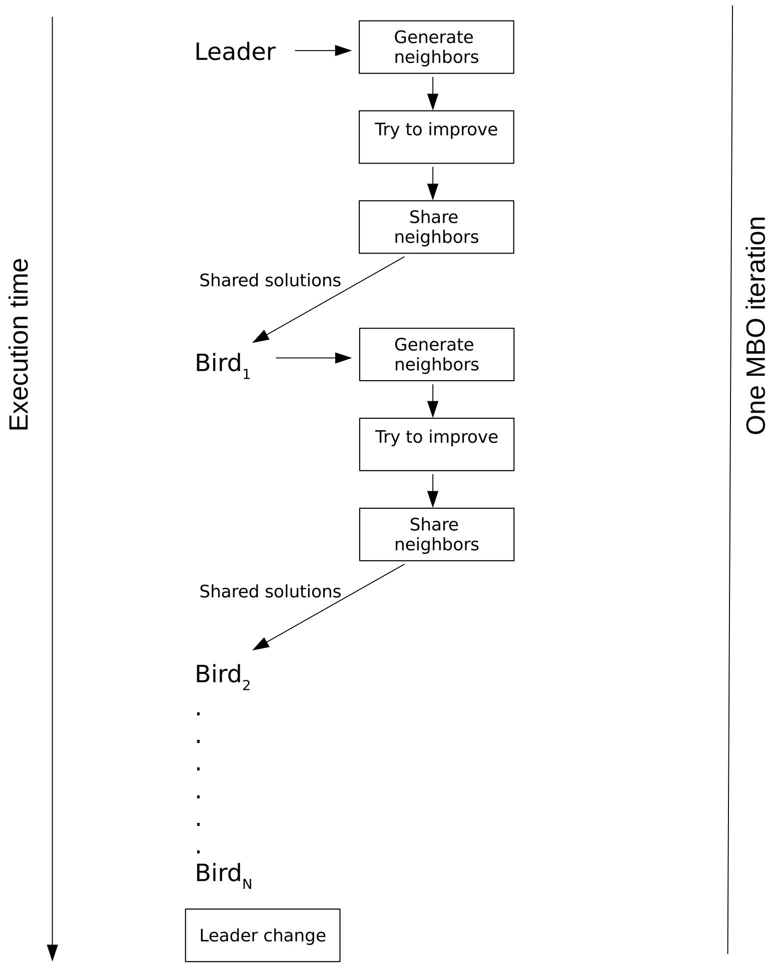

Fig. 2 MBO algorithm flow

5 Experimental study

5.1 Experimental setup

We execute our parallel implementations in a multicore platform, which is based on Intel Xeon Gold 6148 processors (40 cores), located in the National Center for High-Performance Computing (UHeM).

We conduct experiments for a set of randomly generated problem instances due to the lack of a standard benchmark for the multi-objective task allocation problem. We generate a set of test cases by using the same methodology as proposed by the work in the literature [13, 30]. The main configuration parameters for the problem instances and their values are given in Table 1, where task interaction density (D)

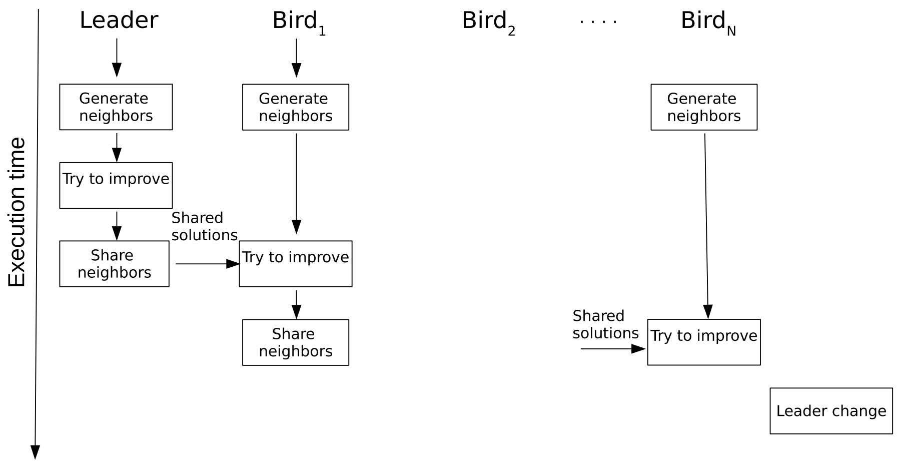

Fig. 3 Parallel MBO algorithm flow

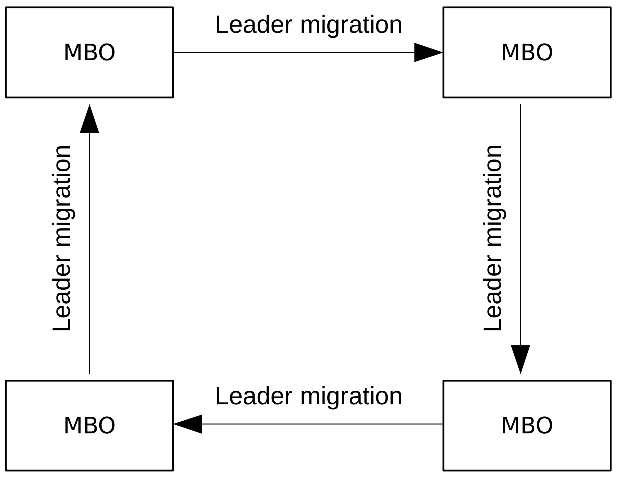

Fig. 4 Island-based MBO structure

Table 1 Configuration parameters for problem instances

Parameter name	Parameter value
Number of tasks (K)	20, 30, 40
Number of processors (N)	8
Task interaction density (D)	0.3, 0.5, 0.8
Communication-to-computation time ratio (CCR)	0.5, 1.0, 2.0

quantifies the ratio of the inter-task communication demands for a task interaction graph, and communication-to-computation time ratio (CCR) represents the amount of resulting communication between tasks in a task graph comprising a parallel application.

In our experiments, 10 problem instances are generated for each parameter configuration, resulting in a total of 270 different problem instances for evaluating the performance of the parallel algorithm. We execute each problem instance for 10 times and take the average of the cost and the running time values. The values of the other system parameters are generated at random from the uniform distributions between the following ranges: the expected execution times of tasks on different processors: 15–25, the data transfer rate between processors: 1–10, the amount of data to be transferred between tasks: such that the CCR is 0.5, 1.0, or 2.0, the failure rate of the processors and the communication path: 0.00005–0.00010 and 0.00015–0.00030, memory resource requirement of each task: 5–10, amount of memory resource of each processor: $x/N - 2x/N$ where $x = \sum_{i=1}^{K} m_i$, computation resource requirement of each task: 5–10, amount of computation resource of each processor: $x/N - 2x/N$, where $x = \sum_{i=1}^{K} c_i$.

5.2 Experimental results

To evaluate the performance of the PMBO and compare it to the other algorithms, we conduct three main experiments:

– Comparison on normal instances
– Comparison on hard instances
– Scalability analysis on very hard instances

Before conducting the performance evaluation experiments, we execute our target implementations by different parameters and fix the parameters that yield the best performance results for the performance comparison experiments.

5.2.1 Parameter tuning

Since the previous work [9, 30] conducts detailed experiments to find the optimal parameters of the MBO algorithm for quadratic assignment problem, we take those values as the following: $GreedyAttemptCount = 3$, $k = 3$, $x = 1$, $n = 51$, $m = 10$.

We also take the values determined as the best for MOTAP by the previous work [13] for GA algorithm:

– Population size = number of tasks $\times$ 2
– Crossover rate = 0.8
– Mutation rate = 0.2

As for the parameters of SSO algorithm, we use the values that are picked in [43]:

- C_g= 0.50
- C_p= 0.85
- C_w= 0.95
- Population size = 50

In order to decide the optimal parameters for island model-based implementations, we conduct experiments by considering multiple values for each parameter including population count, which is the number of islands; immigration period, which represents the number of iterations between two migrations; and immigration count (for IMBO, it is always 1), which represents the number of individuals to be migrated to the other island. The values for each parameter that we execute our IGA, IMBO, and ISSO are as follows:

- Population count = 2, 4, 8, 16
- Immigration period = 5, 10, 20, 50
- Immigration count = 2, 3, 4, 5

We fix the number of threads to eight in this pre-experimentation phase.

The values that achieved the best results for the IGA, IMBO, and ISSO are given in Table 2.

5.2.2 Comparison on normal instances

In the first set of the comparison experiments, we run each algorithm on the normal instances which have a relatively easy configuration of the problem (the allocation of 20/30/40 tasks onto 8 processors). Our aim is to see how much improvement we gain by the parallelization of the algorithm in terms of the time required to explore a certain amount of the search space. To achieve this, we run each algorithm for the same number of neighboring function calls.

Tables 3 and 4 present time and cost values of the algorithms for the normal instances, respectively. The tables include the results for serial genetic algorithm (GA), serial migrating birds optimization (MBO), serial simplified swarm optimization (SSO) algorithm, parallel migrating birds optimization (PMBO), island-based parallel genetic algorithm (IGA), island-based parallel migrating birds optimization (IMBO), and island-based simplified swarm optimization (ISSO) algorithm. Each row presents the time/cost values for each problem instance, while the columns

Table 2 The optimal values for island-based implementations observed in parameter tuning experiments

	Population count	Immigration period	Immigration count
IGA	16	20	5
IMBO	16	50	1
ISSO	16	100	5

Table 3 Time values on the normal instances (Task: 20, 30, 40 Processor: 8) with fixed neighboring function count

	GA	MBO	SSO	PMBO					IGA					IMBO					ISSO				
				1	2	4	8	16	1	2	4	8	16	1	2	4	8	16	1	2	4	8	16
20	3.39	3.57	3.85	6.54	3.68	2.83	2.10	2.35	1.41	0.89	0.62	0.61	0.65	3.95	2.03	1.12	0.61	0.38	3.95	4.38	6.00	7.73	7.68
20	3.40	3.53	3.85	6.54	3.69	2.41	2.07	2.36	1.41	0.91	0.62	0.62	0.66	3.93	1.98	1.09	0.59	0.35	3.96	4.47	6.33	7.73	7.63
20	3.46	3.55	3.84	6.46	3.68	2.43	2.06	2.34	1.41	0.89	0.62	0.61	0.66	3.96	1.99	1.06	0.61	0.47	3.97	4.23	6.17	7.71	7.55
20	3.39	3.80	3.91	6.81	3.87	2.47	2.05	2.32	1.46	0.93	0.62	0.61	0.67	4.18	2.08	1.14	0.60	0.37	4.00	4.12	6.15	7.71	7.63
20	3.59	3.75	3.91	6.82	3.82	2.49	2.08	2.36	1.49	0.93	0.63	0.61	0.67	4.20	2.12	1.12	0.59	0.37	4.03	4.47	6.13	7.86	7.64
20	3.60	3.78	3.90	6.79	3.83	2.51	2.05	2.31	1.47	0.91	0.63	0.62	0.66	4.23	2.13	1.14	0.59	0.43	4.03	4.63	5.91	7.78	7.68
20	3.50	3.86	3.95	6.85	3.85	2.50	2.03	2.30	1.52	0.93	0.64	0.62	0.64	4.28	2.18	1.14	0.63	0.38	4.04	4.15	5.90	7.85	7.69
20	3.61	3.82	3.97	6.75	3.79	2.50	2.09	2.31	1.54	0.97	0.64	0.62	0.67	4.21	2.12	1.11	0.60	0.37	4.04	4.24	5.99	7.80	7.65
20	3.71	3.89	3.97	6.81	3.83	2.49	2.07	2.33	1.55	0.95	0.64	0.61	0.67	4.29	2.15	1.13	0.61	0.37	4.09	4.82	5.99	7.78	7.56
30	7.53	10.45	10.04	15.30	8.17	4.98	3.40	3.61	4.70	2.70	1.62	1.10	1.02	11.35	5.68	3.03	1.56	0.86	11.07	9.99	13.38	18.44	18.04
30	7.78	10.45	10.11	15.32	8.18	4.95	3.40	3.63	4.76	2.72	1.64	1.11	1.02	11.39	5.71	3.05	1.56	0.89	11.18	9.78	13.76	18.06	18.18
30	7.89	10.55	10.02	15.38	8.20	4.94	3.37	3.61	4.81	2.69	1.64	1.11	1.04	11.52	5.81	3.12	1.59	0.91	11.32	10.35	13.63	18.39	18.31
30	7.74	10.19	10.08	15.07	8.07	4.88	3.37	3.64	4.79	2.73	1.64	1.10	1.03	11.22	5.67	2.95	1.53	0.85	11.14	9.96	12.85	18.11	18.01
30	8.04	10.68	10.12	15.01	8.02	4.88	3.34	3.59	4.79	2.73	1.65	1.12	1.04	11.48	5.75	2.98	1.63	0.86	11.30	9.93	13.37	18.20	18.05
30	8.36	10.75	10.10	15.08	8.06	4.88	3.33	3.57	4.85	2.76	1.68	1.12	1.03	11.58	5.80	3.01	1.58	0.87	11.52	10.11	13.50	18.09	17.95
30	8.16	10.97	10.23	15.41	8.27	4.97	3.37	3.61	5.02	2.90	1.73	1.15	1.04	11.75	5.84	3.02	1.60	0.90	11.44	10.19	13.12	18.17	18.21
30	8.56	11.23	10.31	15.44	8.20	4.98	3.38	3.59	5.13	2.93	1.75	1.16	1.04	11.93	5.92	3.10	1.63	0.89	11.75	10.39	13.58	18.19	18.10
30	8.79	11.19	10.30	15.38	8.18	4.96	3.36	3.56	5.08	2.98	1.76	1.16	1.05	11.88	5.93	3.08	1.66	1.14	11.82	10.06	13.10	18.34	18.27
40	15.11	22.96	21.20	27.78	14.38	8.38	5.26	5.02	12.33	6.74	3.87	2.37	1.60	23.90	11.88	6.21	3.43	1.90	21.96	18.20	20.70	30.25	29.99
40	15.68	23.01	21.02	27.70	14.35	8.36	5.24	5.03	12.34	6.84	3.93	2.39	1.61	24.11	11.89	6.21	3.40	1.79	22.35	18.31	21.04	30.31	30.30
40	16.30	23.13	21.08	27.44	14.35	8.36	5.23	4.96	12.60	6.84	3.92	2.40	1.63	24.33	12.03	6.24	3.42	1.79	22.74	18.60	21.21	30.71	30.14
40	15.71	22.57	21.11	27.14	14.18	8.32	5.19	5.03	12.18	6.71	3.87	2.35	1.61	23.58	11.79	6.11	3.33	1.75	22.20	17.97	21.08	31.95	29.97
40	16.02	23.04	21.10	27.23	14.13	8.28	5.20	4.97	12.31	6.92	3.91	2.39	1.66	23.86	11.85	6.23	3.38	1.80	22.63	18.31	20.87	31.21	29.97

Table 3 (continued)

	GA	MBO	SSO	PMBO					IGA					IMBO					ISSO				
				1	2	4	8	16	1	2	4	8	16	1	2	4	8	16	1	2	4	8	16
40	16.77	23.15	20.94	27.20	14.14	8.27	5.20	4.82	12.47	6.83	3.93	2.39	1.61	24.08	12.01	6.22	3.37	1.82	23.17	18.59	21.37	31.56	29.79
40	17.05	24.25	21.71	28.27	14.66	8.58	5.32	4.91	13.12	7.36	4.18	2.52	1.67	25.54	12.53	6.53	3.57	1.84	23.34	18.69	21.63	31.34	30.19
40	17.23	24.20	21.59	28.20	14.66	8.53	5.32	4.90	13.24	7.37	4.17	2.51	1.68	25.34	12.57	6.50	3.64	1.83	23.76	18.59	21.11	31.39	30.28
40	17.69	24.37	21.67	28.28	14.68	9.61	5.31	4.83	13.28	7.39	4.22	2.79	1.70	25.46	12.65	6.74	3.54	1.85	23.94	18.65	20.30	31.02	29.40

Table 4 Cost values on the normal instances (Task: 20, 30, 40 Processor: 8) with fixed neighboring function count

	GA	MBO	SSO	PMBO					IGA					IMBO					ISSO				
				1	2	4	8	16	1	2	4	8	16	1	2	4	8	16	1	2	4	8	16
20	57.5	54.8	56.0	53.8	53.8	53.8	53.7	53.7	56.8	57.3	56.9	56.8	56.8	53.8	53.7	53.7	53.7	53.7	54.5	52.5	52.6	53.1	53.4
20	73.5	70.6	71.8	66.6	66.8	66.6	66.7	66.8	73.4	73.5	73.8	73.4	73.1	67.8	67.6	67.5	67.5	67.4	70.2	67.4	67.8	68.0	68.5
20	80.1	76.6	78.8	74.0	74.1	74.1	74.1	74.2	79.0	79.0	79.5	79.5	79.4	74.0	74.1	74.1	74.1	74.1	76.9	73.3	73.8	74.6	75.1
20	64.5	62.5	63.7	61.0	61.1	60.9	61.0	60.9	63.5	63.2	63.1	63.4	63.7	61.1	61.2	61.3	61.2	61.2	62.2	59.9	59.8	60.2	60.6
20	87.8	83.2	84.1	79.2	79.3	79.4	79.4	79.3	86.0	85.9	86.4	86.6	86.7	79.6	79.7	79.6	79.5	79.7	82.8	79.8	79.3	80.6	80.3
20	109.4	105.9	109.2	102.7	102.4	102.6	102.7	102.6	108.5	109.0	109.1	109.1	109.2	102.6	102.5	102.6	102.4	102.5	105.9	102.7	102.9	103.7	104.3
20	79.8	78.2	80.0	76.2	76.2	76.3	76.2	76.2	79.3	79.0	79.2	79.4	79.5	76.2	76.3	76.2	76.2	76.2	77.0	74.1	74.0	74.6	75.5
20	97.5	93.9	95.5	90.5	90.5	90.6	90.4	90.5	96.9	96.2	97.1	96.8	96.6	91.0	91.2	91.2	90.9	91.0	93.4	90.4	90.1	90.6	91.2
20	157.9	151.8	156.6	145.7	145.6	146.0	145.7	145.6	156.2	156.8	157.3	157.5	156.7	146.5	146.2	146.4	146.5	146.5	153.3	147.0	148.1	148.8	149.6
30	87.8	85.6	86.4	84.5	84.4	84.5	84.5	84.5	88.9	88.6	89.3	88.9	88.7	84.4	84.4	84.5	84.4	84.4	85.1	82.3	82.3	82.6	83.0
30	106.8	102.7	104.6	100.8	100.6	100.6	100.7	100.6	108.5	108.5	109.0	108.2	108.7	100.7	100.7	100.6	100.7	100.6	101.4	99.0	98.5	99.5	99.3
30	138.7	132.5	135.9	129.4	129.5	129.8	129.4	129.6	140.1	140.4	139.5	140.7	140.7	129.6	129.4	129.6	129.7	129.6	132.4	128.6	128.5	129.0	129.4
30	106.3	102.0	103.2	100.0	100.1	100.0	100.0	100.0	107.9	107.4	107.4	107.6	107.1	100.1	100.1	100.1	100.1	100.1	100.8	98.3	98.0	98.4	98.8
30	140.0	135.6	136.9	133.0	133.0	133.1	133.0	133.1	142.1	141.6	141.8	141.7	141.8	133.3	133.4	133.3	133.2	133.2	134.5	131.0	130.9	131.9	131.8
30	193.6	184.3	187.7	179.6	179.6	179.6	179.6	179.7	196.7	196.7	197.5	197.2	196.7	180.1	180.3	180.2	180.2	180.3	184.1	180.5	180.4	180.4	180.7
30	143.0	139.0	140.0	136.5	136.2	136.3	136.2	136.4	144.9	145.3	145.1	145.1	145.3	136.5	136.5	136.6	136.6	136.6	137.4	133.6	133.3	134.1	134.6
30	217.4	209.0	208.8	198.8	198.7	199.0	199.0	198.9	221.0	221.4	221.3	221.4	220.6	201.1	201.8	201.6	201.8	201.4	209.7	202.4	202.5	203.5	203.8
30	332.4	319.3	315.1	302.5	302.8	302.9	303.2	303.5	336.2	337.8	336.4	338.0	338.9	309.4	310.4	309.9	309.8	309.8	320.9	309.8	311.1	312.9	314.3
40	117.3	114.9	115.7	114.2	114.2	114.2	114.2	114.2	120.9	120.7	120.7	120.7	120.7	114.1	114.2	114.1	114.1	114.1	114.3	112.1	111.6	112.1	112.5
40	159.0	154.4	156.0	152.3	151.9	152.0	152.0	152.0	163.5	163.9	164.0	164.4	163.9	152.1	152.1	152.1	152.1	152.1	153.7	151.3	150.5	150.3	150.8
40	222.4	212.2	214.0	207.5	207.3	207.3	207.6	207.5	231.5	232.5	231.6	232.4	231.8	208.0	207.9	207.8	208.0	208.1	214.1	210.7	209.8	209.7	210.4
40	160.4	155.0	155.7	152.9	153.0	152.8	152.9	153.1	165.9	166.1	165.9	166.6	166.0	153.0	152.9	152.9	153.0	153.0	154.3	151.9	150.5	151.5	151.7
40	235.7	225.4	226.6	218.3	218.1	218.3	218.3	217.9	244.8	245.4	246.2	246.1	246.0	219.8	220.0	219.7	220.0	219.7	224.2	220.4	219.3	219.5	219.5
40	334.6	317.9	318.1	308.2	307.8	307.9	307.7	308.0	348.6	348.9	348.5	347.5	347.5	310.6	311.3	310.2	310.6	310.4	323.6	316.9	317.2	318.1	317.4
40	217.1	210.0	209.2	203.4	203.2	203.2	203.1	203.3	225.0	224.4	224.4	223.8	224.9	204.6	204.3	204.2	204.5	204.4	208.2	203.4	204.0	203.3	203.9

Table 4 (continued)

	GA	MBO	SSO	PMBO					IGA					IMBO					ISSO				
				1	2	4	8	16	1	2	4	8	16	1	2	4	8	16	1	2	4	8	16
40	320.2	306.6	304.5	293.9	294.8	293.5	294.5	294.0	329.5	329.8	330.6	329.4	330.9	299.1	298.9	299.4	299.9	299.5	307.1	300.8	299.8	302.4	301.1
40	498.2	480.0	469.4	457.9	457.8	457.5	458.2	458.6	512.3	511.9	511.1	513.9	512.6	466.6	467.9	467.4	466.8	468.2	487.2	474.4	472.6	473.6	474.2

represent the different algorithm executions. For parallel versions, we include the executions with 1, 2, 4, 8, and 16 threads.

If we look at the results, we can see that IGA and IMBO complete their execution in shorter time; however, PMBO achieves the best cost values especially for larger problem instances (allocating 40 tasks) with reasonable times. By this observation, PMBO seems promising for better solutions for much larger problems.

Since our aim is to find out the performance effect of the parallel implementations, we present the average execution time values for each parallel algorithm (namely PMBO, IMBO, IGA, ISSO) in Fig. 5. While we expect lower execution times as the thread count increases, the results do not have this trend for the first two cases (8_20 and 8_30, allocating 20 tasks and 30 tasks onto 8 processors, respectively). Since threading incurs some cost in terms of time, and the total execution time is low for smaller problem sizes, threading cost increases the total execution time for the larger thread counts for those cases. However, as the problem size increases, total execution time increases for each algorithm, and the ratio of the threading cost decreases. Thus, we have smaller total execution times for larger thread counts in the last case (8_40).

5.2.3 Comparison on hard instances

To observe the performance of the algorithms on the cost values in a given time, we perform a set of experiments for hard(er) problem instances by limiting the

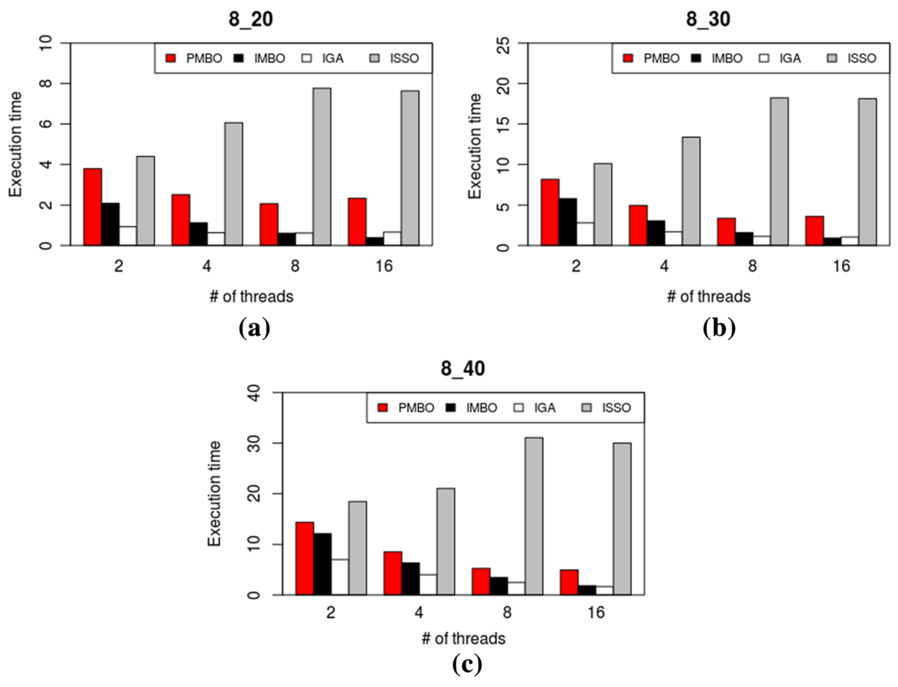

Fig. 5 Average execution time values on the normal instances

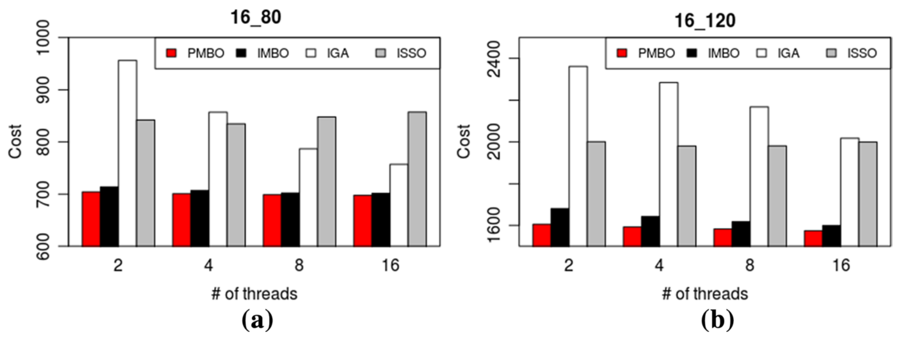

Fig. 6 Average cost values on the hard instances

execution time to 60 seconds. We aim to see how much improvement the parallelism provides on solution qualities if we have a limited time budget. Figure 6 demonstrates the average cost values for two problem instances, assigning 80 tasks onto 16 processors (16_80) and assigning 120 tasks onto 16 processors (16_120).

While the number of threads increases, the cost values for PMBO, IMBO, and IGA tend to decrease while PMBO maintains the smallest values. However, ISSO yields almost the same values for the 2-thread, 4-thread, 8-thread execution scenarios; it even produces larger cost values for the 16-thread execution, which is not promising for larger parallelism.

5.2.4 Scalability analysis

To be able to understand the behavior of our parallel implementations for the larger level of parallelism (larger thread counts), we perform a set of experiments for very hard problem instances on a 40-core parallel system, as one node of our cluster. We execute our parallel implementations for three problem instances (32_240, 40_320, 64_480) by 8-thread, 16-thread, 32-thread, and 40-thread configurations. Figure 7 presents the average cost values for three problem instances. The results demonstrate that PMBO maintains a reduction in the cost values as the thread counts increase for very hard problem instances. (It is much clear in the 40_320, 64_480 instances.)

To see the relative performance of PMBO over other parallel implementations, we also present cost improvement rate of PMBO in Fig. 8. We specifically include the values calculated by the division of the cost values of the IGA,

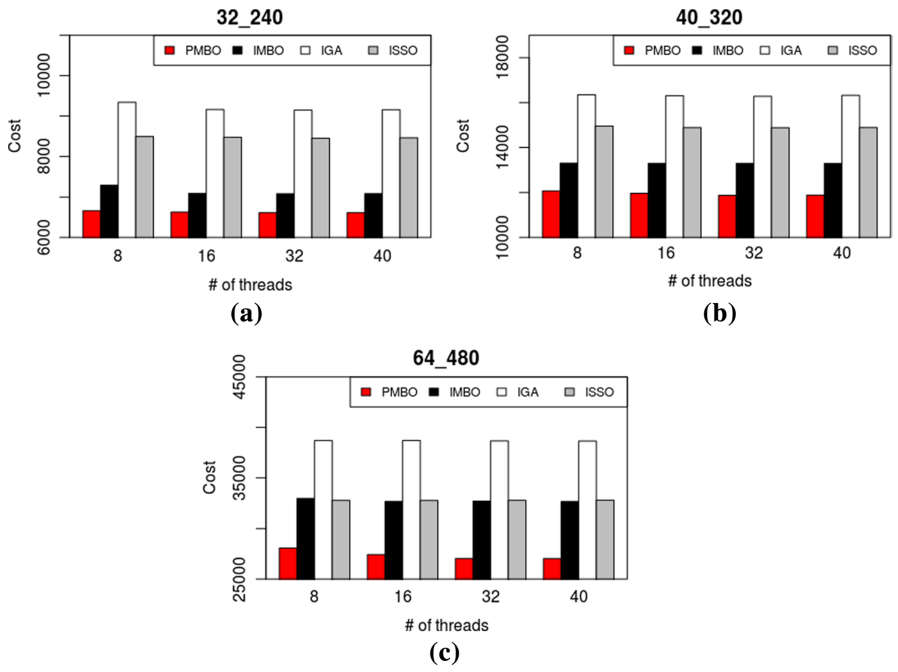

Fig. 7 Average cost values on the very hard instances

IMBO, and ISSO by the cost values of the PMBO. The results demonstrate that as the parallelism level increases, the efficiency of PMBO also increases relatively if we consider the cost values produced by the island-based methods. These results indicate the scalability of the PMBO, in which the higher parallelism level, namely higher computational resources, provides an improvement on the solution quality.

6 Conclusion

In this work, we propose a parallel implementation for migrating birds optimization algorithm to solve the task allocation problem with performance and reliability constraints. Our scalable algorithm provides an efficient execution in a parallel environment providing concurrent execution of multiple threads. The scalability analysis, we perform as part of our experimental study, demonstrates that high parallelism provides substantial solution improvement for hard problem instances with a large number of tasks and processors.

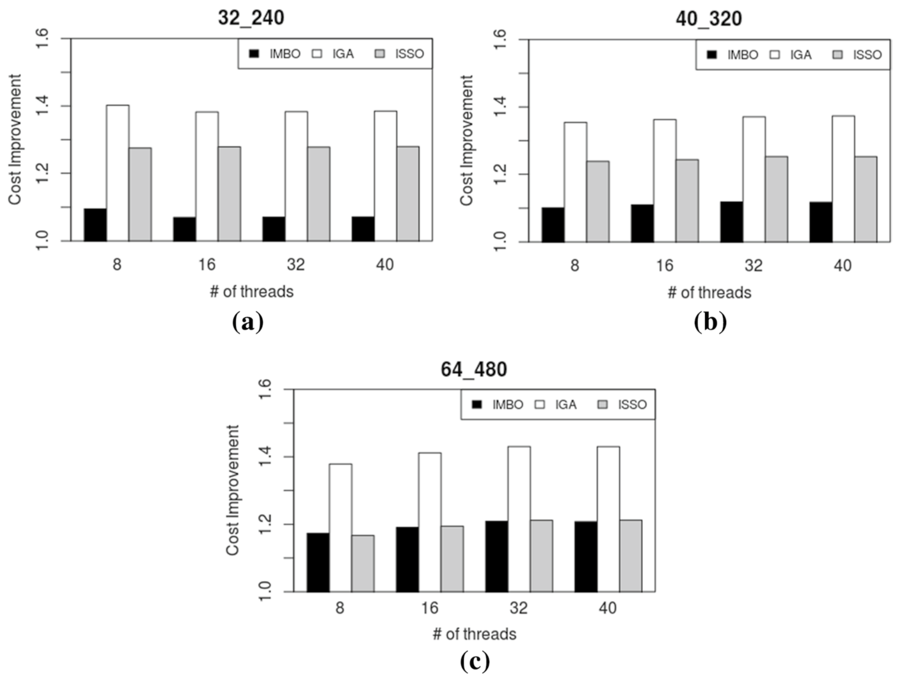

Fig. 8 Average cost improvement values of PMBO over island-based parallel algorithms on the very hard instances

Acknowledgements Computing resources used in this work were provided by the National Center for High-Performance Computing of Turkey (UHeM) under Grant Number 1006722019. This work was supported within the scope of the scientific research project which was accepted by the Project Evaluation Commission of Yasar University under the project number of BAP071 and the title of "Parallelization of Evolutionary Algorithms for The Multi-objective Task Allocation Problem."

References

1. Alba E, Tomassini M (2002) Parallelism and evolutionary algorithms. IEEE Trans Evol Comput 6(5):443–462
2. Alba E, Troya JM (1999) A survey of parallel distributed genetic algorithms. Complexity 4(4):31–52
3. Alba E, Troya JM (2000) Influence of the migration policy in parallel distributed gas with structured and panmictic populations. Appl Intell 12:163–181
4. Al-Betar MA, Awadallah MA, Doush IA, Hammouri AI, Mafarja M, Alyasseri ZAA (2019) Island flower pollination algorithm for global optimization. J Supercomput 75(8):5280–5323
5. Attiya G, Hamam Y (2006) Task allocation for maximizing reliability of distributed systems: a simulated annealing approach. J Parallel Distrib Comput 66(10):1259–1266
6. Cahon S, Melab N, Talbi EG (2004) Paradiseo: a framework for the reusable design of parallel and distributed metaheuristics. J Heuristics 10(3):357–380
7. Chen WH, Lin CS (2000) A hybrid heuristic to solve a task allocation problem. Comput Oper Res 27(3):287–303
8. Chu D, Till M, Zomaya A (2005) Parallel ant colony optimization for 3d protein structure prediction using the hp lattice model. In: 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS)

9. Duman E, Uysal M, Alkaya AF (2012) Migrating birds optimization: a new metaheuristic approach and its performance on quadratic assignment problem. Inf Sci 217:65–77
10. Ernst A, Jiang H, Krishnamoorthy M (2006) Exact solutions to task allocation problems. Manage Sci 52(10):1634–1646
11. Hadj-Alouane A (1996) A hybrid genetic/optimization algorithm for a task allocation problem
12. Kang Q, He H, Deng R (1997) Bi-objective task assignment in heterogeneous distributed systems using honeybee mating optimization. In: IBM Microelectronics Division
13. Kang Q, He H, Deng R (2012) Bi-objective task assignment in heterogeneous distributed systems using honeybee mating optimization. Appl Math Comput 219(5):2589–2600
14. Kang Q, He H, Wei J (2013) An effective iterated greedy algorithm for reliability-oriented task allocation in distributed computing systems. J Parallel Distrib Comput 73(8):1106–1115
15. Kang QM, He H, Song HM, Deng R (2010) Task allocation for maximizing reliability of distributed computing systems using honeybee mating optimization. J Syst Softw 83(11):2165–2174
16. Kartik S, Murthy CSR (1997) Task allocation algorithms for maximizing reliability of distributed computing systems. IEEE Trans Comput 46(6):719–724
17. Lai CM, Yeh WC, Huang YC (2017) Entropic simplified swarm optimization for the task assignment problem. Appl Soft Comput 58:115–127
18. Lassig J, Sudholt D (2010) The benefit of migration in parallel evolutionary algorithms. In: Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation (GECCO)
19. Lassig J, Sudholt D (2013) Design and analysis of migration in parallel evolutionary algorithms. Soft Comput 17(7):1121–1144
20. Limmer S, Fey D (2017) Comparison of common parallel architectures for the execution of the island model and the global parallelization of evolutionary algorithms. Concurr Comput Practice Exp 29(9):e3797
21. Lin SW, Ying KC, Huang CY (2013) Multiprocessor task scheduling in multistage hybrid flowshops: a hybrid artificial bee colony algorithm with bi-directional planning. Comput Oper Res 40(5):1186–1195
22. Liu YY, Wang S (2015) A scalable parallel genetic algorithm for the generalized assignment problem. Parallel Comput 46((C)):98–119
23. Luo GH, Huang SK, Chang YS, Yuan SM (2014) A parallel bees algorithm implementation on gpu. J Syst Architect 60:271–279
24. Luong TV, Melab N, Talbi EG (2010) Gpu-based island model for evolutionary algorithms. In: Annual Conference on Genetic and Evolutionary Computation (GECCO)
25. Middendorf M, Reischle F, Schmeck H (2002) Multi colony ant algorithms. J Heuristics 8(3):305–320
26. Mirzazadeh M, Shirdel GH, Masoumi B (2011) A honey bee algorithm to solve quadratic assignment problem. J Optim Ind Eng 4(9):27–36
27. Mittal S (2016) A survey of techniques for architecting and managing asymmetric multicore processors. ACM Comput Surv 48:3
28. Neumann F, Oliveto PS, Rudolph G, Sudholt D (2011) On the effectiveness of crossover for migration in parallel evolutionary algorithms. In: Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation (GECCO)
29. Niroomand S, Hadi-Vencheh A, şahin R, Vizvári B (2015) Modified migrating birds optimization algorithm for closed loop layout with exact distances in flexible manufacturing systems. Expert Syst Appl 42(19):6586–6597
30. Oz D (2017) An improvement on the migrating birds optimization with a problem-specific neighboring function for the multi-objective task allocation problem. Expert Syst Appl 67:304–311
31. Pan QK, Dong Y (2014) An improved migrating birds optimisation for a hybrid flowshop scheduling with total flowtime minimisation. Inf Sci 277:643–655
32. Pendharkar PC (2015) An ant colony optimization heuristic for constrained task allocation problem. J Comput Sci 7:37–47
33. Pospichal P, Jaros J, Schwarz J (2010) Parallel genetic algorithm on the cuda architecture. In: European Conference on the Applications of Evolutionary Computation (EvoApplications)
34. Randall M, Lewis A (2002) A parallel implementation of ant colony optimization. J Parallel Distrib Comput 62:1421–1432
35. Salman A, Ahmad I, Al-Madani S (2002) Particle swarm optimization for task assignment problem. Microprocess Microsyst 26(8):363–371

36. Shatz S, Wang JP, Goto M (1992) Task allocation for maximizing reliability of distributed computer systems. IEEE Trans Comput 41(9):1156–1168
37. Stone HS (1977) Multiprocessor scheduling with the aid of network flow algorithms. IEEE Trans Software Eng 3(1):85–93
38. Tanese R (1989) Distributed genetic algorithms. In: Proceedings of the 3rd International Conference on Genetic Algorithms (ICGA)
39. Ucar B, Aykanat C, Kaya K, Ikinci M (2006) Task assignment in heterogeneous computing systems. J Parallel Distrib Comput 66(1):32–46
40. Vajda A (2011) Programming Many-Core chips, 1st edn. Springer, Berlin
41. Vidyarthi DP, Tripathi AK (2001) Maximizing reliability of distributed computing system with task allocation using simple genetic algorithm. J Syst Architect 47(6):549–554
42. Yadav PK, Singh MP, Sharma K (2011) Article: an optimal task allocation model for system cost analysis in heterogeneous distributed computing systems: a heuristic approach. Int J Comput Appl 28(4):30–37
43. Yeh WC, Lai CM, Huang YC, Cheng TW, Huang HP, Jiang Y (2017) Simplified swarm optimization for task assignment problem in distributed computing system. In: International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)
44. Yin PY, Yu SS, Wang PP, Wang YT (2007) Multi-objective task allocation in distributed computing systems by hybrid particle swarm optimization. Appl Math Comput 184(2):407–420
45. Zhang Q, Cheng L, Boutaba R (2010) loud computing: state-of-the-art and research challenges. J Internet Serv Appl 1:7–18

Affiliations

Dindar Öz[1] · **Işıl Öz**[2]

Dindar Öz
dindar.oz@yasar.edu.tr

[1] Software Engineering Department, Yaşar University, Izmir, Turkey

[2] Computer Engineering Department, Izmir Institute of Technology, Izmir, Turkey