Journal of Advances in Computer Research
Quarterly pISSN: 2345-606x eISSN: 2345-6078
Sari Branch, Islamic Azad University, Sari, I.R.Iran
(Vol. 11, No. 4, November 2020), Pages: 57-72
www.jacr.iausari.ac.ir

# Static Task Allocation in Distributed Systems Using Parallel Genetic Algorithm

**Monireh Taheri Sarvtamin** *
Department of Computer Engineering, Kerman Branch, Islamic Azad University, Kerman, Iran
mtaheri@iauk.ac.ir

**Received: 2020/05/19; Accepted: 2020/10/26**

**Abstract**

Over the past two decades, PC speeds have increased from a few instructions per second to several million instructions per second. The tremendous speed of today's networks as well as the increasing need for high-performance systems has made researchers interested in parallel and distributed computing. The rapid growth of distributed systems has led to a variety of problems. Task allocation is a key process for distributed systems to achieve effective system efficiency, which, except for a few cases, is an NP-complete problem. Finding an effective and efficient method for this problem is still sought despite various methods used in studies. Experiments and the results of previous research have shown that NP problems are better solved by exploratory methods than other methods. This study used a parallel genetic algorithm (PGA) to find a solution for proper task allocation to processors in a distributed system. The task allocation policy, obtained using PGA, is much faster than traditional genetic algorithms. The results showed that the proposed algorithm can provide optimal or near-optimal allocations for problems of different sizes. The proposed method also solved large- and medium-sized problems much faster than traditional genetic algorithms and with super linear speedup.

**Keywords:** Distributed System, Task Allocation, Parallel Genetic Algorithm, Static Task Allocation

## 1. Introduction

Today, distributed systems are introduced as a powerful platform for running high-efficiency parallel programs and as an alternative to massive parallel machines. Distributed systems have a set of heterogeneous processors connected through a communication network [1-4]. The efficiency of a parallel program divided into several tasks simultaneously run on different system processors on a distributed system largely depends on task allocation to system processors. If the allocation is not implemented properly, system processors may wait for each other for most of their time instead of doing useful calculations. Optimal allocation is an allocation that can minimize system costs

A distributed system may have static or dynamic allocation policies according to its allocation decision making time. The static task assignment assumes the information about the task and the characteristics of the processor already available before the execution of the task. Static mapping is used for various purposes. It can be used to plan the execution of a set of tasks in the future. Dynamic mapping is done when tasks are mapped online in real-time, for example, tasks are mapped when they arrive at indefinite intervals.

In the general case, the task allocation problem in both cases is an NP-complete problem. The key importance of this problem in efficiency has led many researchers to examine it and introduce many methods for solving it [5-8]. The efficiency of these systems highly require runtime (system cost) reduction and system reliability elevation [9-11].

Various task allocation problem solutions are proposed [10, 12], generally divided into two main classes of exact algorithms and heuristic methods. The graph theory, state-space search [13-15] and mathematical programming [16-18] are used in implementing exact algorithms .In these algorithms, time grows exponentially with the size of the problem, so they are actually used for small problems, while suboptimal solutions can be obtained in heuristic methods providing powerful and fast tools.

Various problems are successfully solved by genetic algorithms (GAs) [5-8]. Due to recent advances in computer systems architecture, as well as the good ability and significant achievements of PGAs, extensive research has been conducted on them [9-12]. These algorithms have performed very well in high-complexity problems [13, 14], which is why they are used in this study.

In this study, PGA was used to map a set of tasks to sources statically in heterogeneous distributed computing environments to reduce the runtime. The results of reducing the runtime in this problem using PGA compared to sequential genetic algorithms as well as the results of evaluating the speedup and efficiency of this algorithm over the sequential version are presented below. The results showed that by encoding the problem as a chromosome, PGA could generate an optimal or near-optimal task assignment, and the proposed method could achieve super linear speedup. Experimental results showed that PGA has a better performance for the task allocation problem even without parallel running than the standard (traditional) genetic algorithm, the causes of which are discussed below. The major contributions of this study are listed below:

1. With the advancement of science, exploratory methods can no longer solve complex problems, necessitating the employment of meta-exploratory methods with better and higher performance in solving problems. Therefore, this study used parallel genetic algorithms, which are more powerful than standard genetic algorithms. In addition to searching in parallel at multiple points in space, these algorithms increase diversity in subpopulations, prevent premature convergence, and reduce entanglement in local optimizations.

2. The increasing complexity of scientific computing in recent years has raised the need to increase code execution speed, so if all the processor and system hardware power is used, great success is achieved in reducing runtime and increasing code speed. Due to advances in CPU manufacturing and multi-core processors, as well as the advent of GPUs, this study has tried to use the potential power of multi-core processors to execute the proposed algorithm in parallel, to achieve the desired response in much less time.

This paper presents related previous works, task allocation problem definition, the proposed method explanation, experimental design, experimental results, and conclusions in sections 2 to 7, respectively.

## 2. Related works

New effective techniques are always sought for best possible solutions of intractable task allocations problems in an acceptable computational time. The literature is full of studies on the task allocation problem providing several methods [25, 3]. The issue under investigation was first introduced by Stone [15]. Stone's main work was to develop the Task Interaction Graph (TIG) model for executive task description. In the introduced model, each processor ran exactly one task at a time.

Aggarwal et al. provided an efficient way to allocate resources using hybrid planning and optimization in a distributed system. To achieve smaller execution intervals with a lower error rate probability, they used ant colony optimization and Round Robin scheduling provided an effective scheduling solution and optimizing resource management [27].

In an article titled "A survey of task allocation and load balancing in distributed systems", Jiang examined the general characteristics of distributed systems. He examined studies on task allocation and load balancing in various aspects and discussed future research paths [16].

In an article titled "A reformed task scheduling algorithm for heterogeneous distributed systems with energy consumption constraints", Hu et al. proposed a task scheduling program to satisfy energy constraints in heterogeneous distribution systems and minimize the scheduling time of parallel programs. According to the experimental results, better planning length was achieve through the new algorithm in energy consumption limitations compared to existing algorithms [29].

Akbari and Rashidi proposed a multifunctional scheduling algorithm based on cuckoo optimization for the task allocation problem when compiling in heterogeneous systems. They ensured that local optimization is avoided. Their proposed algorithm, while allowing global search in the problem area to speed up global search optimization, provides a relatively optimal timing with the least number of repetitions [2].

Yadav et al. presented a heuristic method for optimal task allocation, to analyze the cost of heterogeneous distributed computing systems. The proposed algorithm in this study , used Communication Link Sum (CLS) for one by one task allocation to processors , which reduces the Inter-Process Communication (IPC) and thus minimizes system costs [17].

Hamed presented a GA seeking task allocation optimal solution in heterogeneous distributed systems to balance the load assigned to each processor. Experimental results indicated that the GA was more effective than conventional algorithms [18]. In a recent study entitled "Designing a task allocation framework for distributed systems", Chaubey et al. tried to design a task allocator framework to implement different task allocation algorithms. Using this task allocator framework, one can replace the task allocation mechanism with a new mechanism when necessary [19].

In a study, Akbari used the TRIZ method to improve the genetic algorithm performance and solved the problem of assigning and scheduling tasks in heterogeneous distributed systems. In his proposed method, he has improved the genetic algorithm performance by altering and manipulating genetic functions [20]. In another study, Hosseini Shirvani used a shuffled genetic algorithm to schedule tasks in distributed systems such as grids and cloud computing. This study used the Heterogeneous Earliest Finish Time (HEFT) method to intelligently generate the initial population. The method

also uses a shuffled operator to take advantage of both exploration and extraction in finding promising individuals in the search space. The results of the proposed method were compared with HEFT and QGARAR, indicating that it has better performance in terms of mean makespan[21].

In an article entitled "Heuristic algorithm for scheduling tasks in cloud computing using hybrid particle swarm optimization and bat algorithms", Barzegar et al. developed a hybrid algorithm called PSOBat-Greedy to schedule tasks and provide appropriate responses in cloud computing. Their proposed method has somewhat increased the resource efficiency compared to the particle swarm optimization algorithm and the bat algorithm [22].

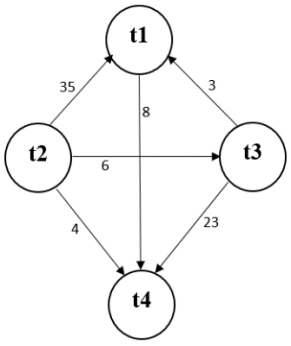
In another study on task allocation in distributed systems, Neelakantan and Sreekanth minimized task response time by transferring tasks from loaded computers to computers under load using load balancing techniques. The present study balanced tasks among computers on a distributed system because diffusion technique strength lies in all port communication model and asynchronous implementation [1].

## 3. Problem definition

The problem in this study relates to an optimal static task allocation of a parallel program on processors in a distributed system. A distributed system is made up of two sets, the set of heterogeneous processors ($P=P_1, P_2, \ldots, P_m$) interconnected by communication lines, and the set of program tasks ($T=t_1, t_2, \ldots, t_n$) that together form a common goal. The cost of executing an executable task varies across different processors, and the execution costs are expressed as an n×m matrix called Execution Cost Matrix (ECM). Similarly, a symmetric n×n matrix called Inter Task Communication Cost Matrix (ITCCM) is used to show the connection cost between two tasks.

Figure 1 shows a distributed system with different processors, Task Interaction Graph (TIG), and execution cost and connection cost tables. Part (a) shows a parallel program by a task interaction graph G (V,E). In the task interaction graph, vertices represent tasks and edges represent relationships between tasks. In this graph, V defines the task set and E the edges set. Each i ϵ V task has a certain processing load. Part (b) shows the costs or the runtimes of three different processors with an ECM.

Part (c) shows the matrix of communication costs obtained from the TIG graph. In short, there is a set of N tasks stating that a parallel program must execute on a distributed system with M processors.




**Execution Cost Matrix**

| | $p_1$ | $p_2$ | $p_3$ |
|---|---|---|---|
| $t_1$ | 31 | 4 | 14 |
| $t_2$ | 1 | 5 | 6 |
| $t_3$ | 2 | 4 | 24 |
| $t_4$ | 3 | 28 | 10 |

**Communication Cost Matrix**

| | t1 | t2 | t3 | t4 |
|---|---|---|---|---|
| **t1** | 0 | 0 | 0 | 8 |
| **t2** | 35 | 0 | 6 | 4 |
| **t3** | 3 | 0 | 0 | 23 |
| **t4** | 0 | 0 | 0 | 0 |

**Figure 1. (a) Task interaction graph; (b) Execution cost matrix; (c) Communication cost matrix**

### 3.1 Problem formulation

The formulation is a mathematical model for a static task allocation problem involving two steps: (i) cost function formulation for representing the main purpose; (ii) formulating some limitations or inequalities according to tasks needs and system resources availability. To do that, X is a binary matrix of N×M corresponding to the allocating N tasks to M processors so that:

$$X_{ip} = \begin{cases} 0 \\ 1 \end{cases} \quad (1)$$

If task i is assigned to processor p, $X_{ip}=1$, otherwise, $X_{ip}=0$.

**Location limitation**

Each task should only be allocated to one processor that executes completely and without preemption. The following Equation must be followed for each task.

$$\sum_{p} X_{ip} = 1 \quad (2)$$

**Cost function**

According to Equation 3, task allocation to processors is defined as an X function as follows:

$$X : T \rightarrow B \text{ such that } X(i) = k \quad (3)$$

If the $i^{th}$ task is assigned to the $k^{th}$ processor.

#### A. Processor Execution Cost (PEC)

PEC of $t_i$ task on a $P_k$ processor is determined by $ec_{ik}$, which is the total cost required in the execution process to execute $t_i$ on that processor. The corresponding execution cost of a tasked that cannot be executed on a particular processor is assumed infinite (∞), and Equation 4 shows how to calculate the processor execution cost in an X task allocation, for all tasks assigned to the processor k [23]:

$$PEC(X)_K = \sum_{i=1}^{m} ec_{ik} x_{ik} \quad (4)$$

#### B. Inter Processor Communication Cost (IPCC)

IPCC of $cc_{ij}$ is calculated when data is transferred from one task to another due to the relationship between tasks; and this cost is added to the calculations when the $t_i$ and $t_j$ tasks are deployed on separate processors during the execution process. When two tasks are executed on one processor, $cc_{ij}=0$; hence in an X task allocation, IPCC for the $k^{th}$ processor k is calculated as follows [23]:

$$IPCC(X)_K = \sum_{i=1}^{m} \sum_{j>i}^{m} (cc_{ij}) x_{ik} x_{jb} \quad (5)$$

In an X task allocation, PEC and IPCC for $k^{th}$ processor make up the total cost of the $k^{th}$ processor [23]:

$$T_{cost}(X)_k = PEC(X)_k + IPCC(X)_k \quad (6)$$

and the system total cost is calculated as follows [23]:

$$S_{cost}(x) = \sum_{k=1}^{n} T_{cost}(X)_k \quad (7)$$

**C. System Cost Model**

Taking into account system resource limitations, the task allocation for system costs is formulated as follows [23]:

$$\min . S_{cost}(X) \quad (8)$$

$$s.t. \sum_{k=1}^{n} X_{ik} = 1 \;\forall\; i = 1,2,3,...,m \quad (9)$$

$$X_{ik} \in \{0,1\} \;\forall\; i,k \quad (10)$$

In this model, the limitation is set to 9 to assign each task exactly to one processor, and limitation 10 ensures that $X_{ik}$ is a decision variable.

## 4. Proposed Methods

The proposed methods in this paper is to use PGAs. PGAs are not just a parallel version of the old sequential genetic algorithm. In fact, they reach the ideal goal of a parallel algorithm with better behavior than that of the sum of its components. [13] Several parameters justify this. First, PGAs are naturally parallel, because the operations on the threads are independent. Besides, the entire population (panmixia) is geographically structured to focus on competitively selecting among thread subsets, which result in better algorithms. Higher efficiency and greater diversity further reinforce the importance of research advances in PGAs. Figure 2 shows the pseudocode of island genetic algorithms and Figure 3 shows the cell genetic algorithm.

Algorithm 1. Island model with migration interval of τ

1: Initialize a population made up of subpopulations or islands, P(0) = (P1(0),…, Pm(0)).
2. Let t:=1.
3: loop
4: for each island i do in parallel
5: if t mod τ = 0 then
6: Send selected individuals from island Pi(t) to selected neighboring islands.
7: Receive immigrants Ii(t) from islands for which island Pi(t) is a neighbor.
8: Replace Pi(t) by a subpopulation resulting from a selection among Pi(t) and Ii(t)
9: end if
10: Produce Pi(t+1) by applying reproduction operators and selection to Pi(t).
11: end for
12: Let t:= t + 1.
13: end loop

**Figure 2. Pseudo-code of a parallel island genetic algorithm with migration intervals of $\tau$ [24].**

Algorithm 2. Cellular genetic algorithm

1: Initialize all cells to form a population P(0) = (P1(0),...,Pm(0)). Let t:= 0.
2: loop
3: for each cell i do in parallel
4: Select a set Si of individuals from Pi(t) out of all cells neighboring to cell i.
5: Create a set Ri by applying reproduction operators to Si.
6: Create Pi(t+1) by selecting an individual from (Pi(t) ∪ Ri.
7: end for
8: Let t:= t + 1.
9: end loop

**Figure 3. Pseudo-code of parallel cellular genetic algorithms[24].**

### 4.1 Presenting the problem genotype

Finding a proper mapping between the genotype and solution of a problem is a key elements in designing a PGA[9]. In the case of the PGA, each genotype corresponds to a candidate solution to the problem, which represents each task allocation with a genotype using the numerical vector with M elements, where position i refers to a processor number that task i is allocated to: Genotype[i] =p, pϵ (1, 2, …, n).

Figure 4 shows a visual example of the genotype and refers to the allocation of five tasks to three processors. For instance, Genotype[1]=2 indicates that task 1 is assigned to processor 2, and so on.

| Task | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| Processor | 2 | 1 | 3 | 3 | 1 |

**Figure 4. An example of the problem genotype**

### 4.2 Genetic algorithm operators

The selection process is done by the roulette wheel, and the most suitable individuals have a higher chance than the weaker ones. A single-point, two-point, and uniform crossover methods together form a crossover operator. Each of these methods is given a different probability, and at each call to the crossover function, one of these three methods is selected using the roulette wheel. A mutation operator is also used. Here, the mutation mechanism switches the selective processor from the selected task to another randomly selected processor.

The condition for stopping is reaching a defined repetitions number. This number is considered different for different problem sizes, but this number is chosen in such a way that the algorithm becomes convergent and an answer is found. The replacement method here is to create a temporary population of new individuals and old individuals, then sort them according to fitness and in descending order, and then select the best individuals from this new temporary population and replace the previous population.

The fitness function calculates the sum of the execution and communication costs for a produced solution (individual).Table 1 shows the parallel genetic algorithm parameters.

**Table 1. Parallel genetic algorithm parameters**

| Parameters | Value |
|---|---|
| Pcrossover | 0.7 |
| Pmutation | 0.02 |
| Population Size | 100 |
| Iterations | 1000-8000 |

### 4.3 Island genetic algorithm parameters

Determining the parameters for the migration of individuals from one population to another is necessary for an island model genetic algorithm (IGA). One-way circular topology is used here. Ten percent of the best individuals in each population migrate and replace the worst individuals in the destination population. The migration interval is considered at 20.

### 4.4 Temporal complexity of genetic algorithms

Calculating the temporal complexity of a genetic algorithm depends on many factors, including population size, genotype length, algorithm termination conditions, population selection type, fitness function, crossover function, and mutation function. If the algorithm termination condition is considered as a certain number of iterations, the time complexity of the genetic algorithm can be written as follows:

O (G * (N * M * O (Fitness) + G * (N * Pc * O (Crossover) + G * (N * Pm * O (Mutation))

Where

G: Generations Number
N: Population Size
Pc: Probability of Crossover
Pm: Probability of Mutation

Given the common options such as point mutation, single-point crossover, and roulette wheel selection, the complexity of these functions is considered constant, and Pc and Pm are also constant, so the complexity of the genetic algorithm is simplified as follows:

$$O (G * (N * M + N * M + N)) = O (GNM)$$

Space complexity is considered as O (N), considering that twice the population size is needed to maintain the old and new populations, and N is the population size. In genetic algorithm parallelism, the main population is divided into several subpopulations and genetic operators are applied to them. The temporal complexity of the parallel genetic algorithm can be considered as follows:

$$O ((GNM) / Np)$$

Where Np is the number of processors. The acceleration algorithm is obtained by dividing the execution time of the genetic algorithm serially by its parallel execution time.

## 5. Experimental design

The proposed method was executed using MATLAB R2012a software on a PC with CPU=Core i3, 2.00GHz, RAM=4.00GB. Two system configurations were considered for evaluation: a 4-computer distributed system and a 6-computer distributed system. In addition, four random population sizes with values of N=8, 12, 16, and 20 were used.

### 5.1 Efficiency measure

According to a category introduced by Alba [25], the effectiveness of evolutionary serial and parallel algorithms are assessed with "Single machine/Panmixia" and "Orthodox" comparing methods. In the orthodox comparing method, a parallel algorithm is run on m machines, and then the same algorithm is run on one machine, then the results are compared. In the Panmixia comparing method, the results of running the parallel algorithm and running the standard version of the same algorithm on one machine are compared.

### 5.2 Paralleling with MATLAB parallel computing toolbox

The increasing complexity of scientific computing in recent years has raised the need to increase code execution speed more than ever. Thus, MATLAB has provided a parallel computing toolbox so that users can employ multiple processors and GPU to increase the speed of their program execution and save time[26]. The parallel

computing toolbox allows users to process high-volume, time-consuming data using multiprocessor systems, multi-core processors, GPUs, and computer networks.

MATLAB has also made it possible to use all the power of a desktop computer with its parallel computing toolbox, allowing all processors of a multi-processor system and multi-core processor cores to be used via MATLAB workers. MATLAB has also made it possible to run parallel codes on a cluster of computers using MATLAB's distributed computing server, by running the code that is written, troubleshot, and tested in parallel on a multi-core desktop computer or multi-core system, on a cluster of computers.

### 5.3 Dataset

As there is generally no accepted standard basic dataset for evaluating task allocation algorithms aimed at reducing system costs in distributed computing systems, in this study we produced a range of similar samples that other researchers have used [27-29] to test PGAs by simulation. A set of different parameters are used to make a large simulation dataset that determines problem instances properties. The main parameters are:

- Number of tasks in a TIG program (M)
- Number of processors in a distributed computing system (N)
- Costs of executing the program on different processors
- Costs of communication between tasks

In this study, the N value means the number of distributed computing system processors between 4 and 6 variables. The number of M tasks varies by values of 8, 12, 16 and 20 to check the algorithm with different scales.

## 6. Experimental results

### 6.1 Performance evaluation of the proposed method based on orthodox comparison method

Based on the orthodox method, IGA and CGA were run in parallel on two processor cores, then the same algorithms were run on one processor core and their runtime in both cases was recorded to evaluate their performance. Table 2 shows the runtimes of IGA and CGA for various problem sizes. Note that the results in part (a) show the table for the distributed system with 4 computers and part (b) shows the table for the distributed system with 6 computers.

**Table 2. Problem runtime using different algorithms (in seconds). (a) Distributed system with 4 computers; (b) Distributed system with 6 computers**

| (M,N) | $T_{SIGA}$* | $T_{PIGA}$** | $T_{SCGA}$*** | $T_{PCGA}$**** |
|---|---|---|---|---|
| (8,4) | 0.92s | 0.81s | 1.99s | 1.36s |
| (12,4) | 1.19s | 0.88s | 2.13s | 1.39s |
| (16,4) | 3.71s | 2.18s | 11.18s | 6.47s |
| (20,4) | 28.89s | 16.05s | 63.59s | 34.89s |

| (M,N) | $T_{SIGA}$ | $T_{PIGA}$ | $T_{SCGA}$ | $T_{PCGA}$ |
|---|---|---|---|---|
| (8,6) | 12.28s | 6.84s | 42.4s | 23.45s |
| (12,6) | 13.43s | 7.34s | 44.86s | 24.59s |
| (16,6) | 27.03s | 14.61s | 63.05s | 34.48s |
| (20,6) | 28.11s | 14.79s | 64.08s | 35.04s |

*$T_{SIGA}$: Runtime of island genetic algorithm on one processor core
**$T_{PIGA}$: Runtime of island genetic algorithm on two processor cores in parallel
***$T_{SCGA}$: Runtime of cellular genetic algorithm on one processor core
****$T_{PCGA}$: Runtime of cellular genetic algorithm on two processor cores in parallel

As Table 2 shows, both IGA and CGA can solve the problem of static task allocation to processors in distributed systems much faster than sequential genetic algorithms. The speedup graphs of PGAs for different problem sizes is presented in Figure 5. As Figure 5 shows, PGAs speedup rate increases by problem size.

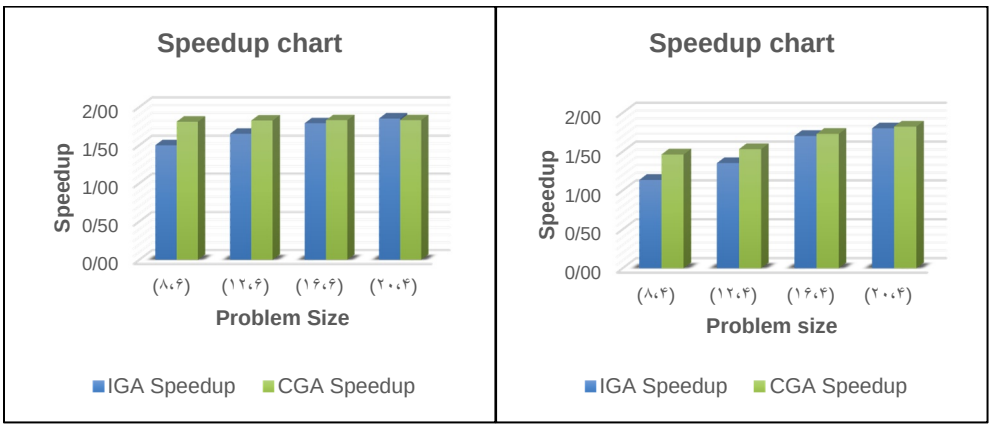



**Figure 5. Speedup graph of island and cellular genetic algorithms (CGAs) for the problem of static task allocation to processors by formal comparison method. (a) In a distributed system with 4 computers; (b) In a distributed system with 6 computers**

As Figure 5 shows, the speedup rate of parallel genetic algorithms increases by the number of tasks. Also, comparison of the two graphs (A and B) indicate that graph B, in which the number of processors is considered more, has a higher speedup rate than graph A. Therefore, increasing the size of the problem, including the tasks and processors number, generally increases parallel genetic algorithms efficiency, and these methods perform better for large and time-consuming problems.

Code and data transfer, as well as competition for resources between workers and the operating system, can be attributed to the declining speedup rate and, of course, efficiency in some experiments. Also, the low efficiency of PGAs in small-size problems is justified by the fact that opening and closing the parallelization tool is time-consuming, so PGAs are not recommended for small problems.

### 6.2 Evaluation of the proposed algorithm by single machine comparison method

In this experiment, genetics parallel algorithms were run on dual-core processors in parallel using MATLAB parallel computing toolbox, and their results were compared with the traditional (standard) genetic algorithm.

Table 3 shows the speedup and efficiency of PGAs for different problem sizes. Parts (a) and (b) of Table 3 show the performance of island and CGAs in a distributed computing system with 4 computers, respectively.

**Table 3. Runtime (in seconds), speedup and performance of PGAs compared to a static task allocation problem solved a standard genetic algorithm in a distributed computer system with 4 computers; (a) The island genetic algorithm; (b) The cellular genetic algorithm**

| (M,N) | $T_{GA}$* | $T_{IGA}$** | Speedup | Efficiency |
|---|---|---|---|---|
| (8,4) | 0.53s | 0.69s | 0.76 | 0.38 |
| (12,4) | 0.56s | 0.74s | 0.76 | 0.38 |
| (16,4) | 9.63s | 2.62s | 3.67 | 1.83 |
| (20,4) | 57.14s | 16.33s | 3.49 | 1.75 |

| (M,N) | $T_{GA}$ | $T_{CGA}$*** | Speedup | Efficiency |
|---|---|---|---|---|
| (8,4) | 0.53s | 1.39s | 0.38 | 0.19 |
| (12,4) | 0.56s | 1.42s | 0.39 | 0.19 |
| (16,4) | 9.63s | 6.4s | 1.5 | 0.75 |
| (20,4) | 57.14s | 34.61s | 1.65 | 0.82 |

*GA: Genetic Algorithm
**IGA: Island Genetic Algorithm
***CGA: Cellular Genetic Algorithm

Table 3 results show that both island genetic algorithms in the single machine comparison for large problems can achieve super-linear speedup. Experiments indicated that standard genetic algorithms need more iterations than PGAs to achieve the answer which increases their runtime. The reasons for the need of standard genetic algorithms to iterations can be explained as follows: (1) PGAs are usually faster and less inclined to finding suboptimal solutions; (2) Standard genetic algorithms are slow and rapidly lose variation; (3) PGAs perform very well in many applications and evolve very rapidly. Figure 6 shows the speedup rate of the two Parallel Island and CGAs for the problem under consideration by the single-machine comparison method.

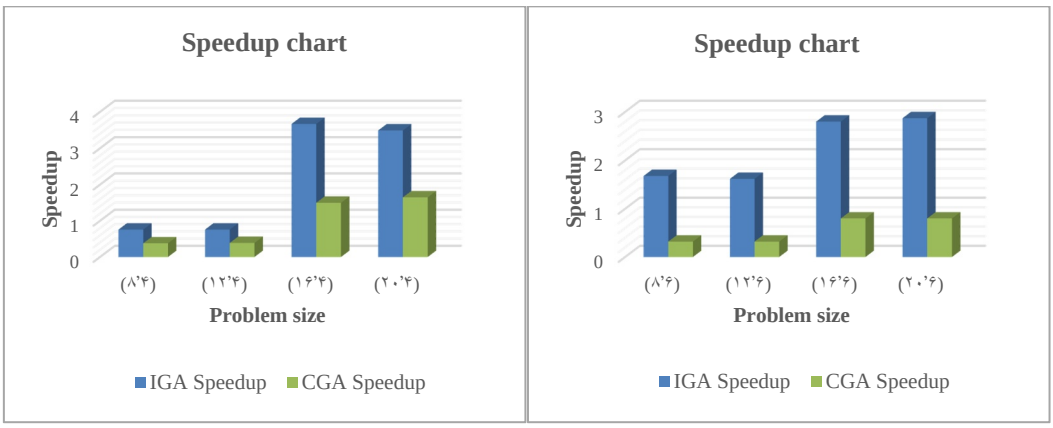



**Figure 6. Speedup rate of PGAs for static task allocation problem by single machine comparison method; (a) In a distributed system with 4 computers; (b) In a distributed system with 6 computers.**

Figure 6 shows that IGA performs better than the CGA when using the single-machine comparison method, and has been able to achieve super-linear speedup. IGA increases diversity in its sub-islands because of migration implementation and helps to find better solutions; in contrast, the CGA in each generation must apply a crossover operator on each of its cells, and due to the high number of cells, each cell has to wait a long time for its genetic operators to apply.

### 6.3. Comparison of the proposed method and PSOBat-Greedy method presented in [22]

In another experiment, to compare the proposed method with the method presented in [22], the number of tasks was increased to 50. The execution time of the methods was compared and the results are presented in Figure 7. The proposed method was much faster than the standard GA and the PSOBat-Greedy methods in solving the mentioned problem.

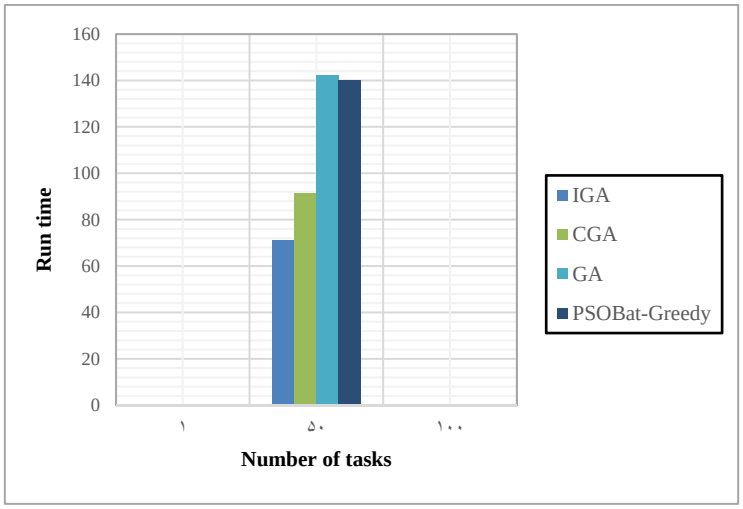



**Figure 7. Comparison of the proposed methods with the PSOBat-Greedy method presented in [22]**

### 6.4 Behavior of PGAs as the number of MATLAB workers increases[1]

By default, the number of workers is set equal to the number of CPU cores when a program is locally runs on one computer by MATLAB parallel computing toolbox. However, the number of workers can also be set to more than the number of processor cores up to 12. Another experiment increased the number of workers but kept the number of processor cores constant at 2, and measured the runtime and speedup of parallel algorithms for different samples. Algorithms efficiency was evaluated based on orthodox comparison method. The interesting thing about this part is that with the increase in the number of MATLAB workers without increasing the number of processor cores, concurrence is actually being implemented, and it can be said that this experiment examines the effect of concurrence on parallel genetic algorithms.

Figure 8 shows IGA and CGA efficiency and speedup graphs for the problem of static task allocation to processors in a distributed system with 12 tasks and 4 computers. The horizontal axes of these two graphs indicate the number of MATLAB workers and the vertical axes show the speedup rate.

[1] In the discussion, the basis of the algorithm evaluation is the formal comparison method.

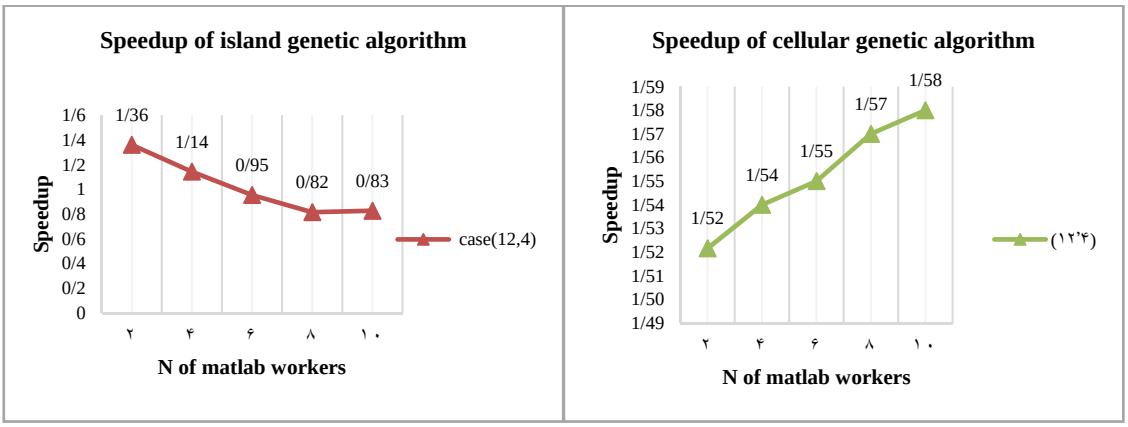



**Figure 8. (A) Measuring speedup rate changes in IGA by changing the number of MATLAB workers. (B) Measuring speedup rate changes in CGA by changing the number of MATLAB workers.**

According to Figure 8 (A), IGA efficiency decreases with increasing the number of MATLAB workers, and concurrence has a negative effect on this method. It appears that the overload caused by data communication between workers and clients slows speedup and ultimately reduces system efficiency due to implementing migration and requiring communication between workers and clients for sending and receiving migrants between islands by IGA.

Figure 8 (b) shows CGA speedup increases by the number of MATLAB workers in task allocation problems and can achieve super-linear speedup. This experiment shows that the concurrency has a positive effect on CGA and well improves its performance.

### 6.5 Other observations

The two PGAs with the traditional genetic algorithm were run sequentially on a single processor core and the results were recorded to compare their performance. Comparisons between these methods suggested that the traditional genetic algorithm requires two to three times the number of repetitions of more generations than PGAs to achieve the answer, requiring more runtime. In addition, traditional genetic algorithms are more likely to be trapped in local optimizations due to their lower diversity than PGAs. In this case, in particular, IGAs share genetic material due to migration between sub-populations and increase diversity in them.

## 7. Conclusion

This study discussed using PGAs for static task allocation problem in distributed systems. PGAs were successfully applied to this problem and were able to find optimal solutions to the problem. The performance of parallel algorithms was evaluated by orthodox and single machine comparison methods. These algorithms performed well on large-scale problems and have been shown not cost-effective in small-scale problems. In addition, the results showed that using PGAs, even when executing on one core of a processor, both decreases the runtime and yields better numerical results. The interesting point is that division of the population into islands or grids is responsible for such numerical benefits.

Further studies are recommended to implement the proposed method on a cluster of computers as well as and develop it for solving other performance criteria or even multi-objective allocation problems. It is recommended to use this method to increase system reliability and load balancing on processors.

## References

[1] P. Neelakantan and S. Sreekanth, "Task allocation in distributed systems," Indian Journal of Science and Technology, vol. 9, 2016.

[2] M. Akbari and H. Rashidi, "A multi-objectives scheduling algorithm based on cuckoo optimization for task allocation problem at compile time in heterogeneous systems," Expert Systems with Applications, vol. 60, pp. 234-248, 2016.

[3] Y. Jiang, "A survey of task allocation and load balancing in distributed systems," IEEE Transactions on Parallel and Distributed Systems, vol. 27, pp. 585-599, 2016.

[4] J. Wu, Distributed system design: CRC press, 2017.

[5] R. Kaluri and P. R. CH, "Optimized feature extraction for precise sign gesture recognition using self-improved genetic algorithm," International Journal of Engineering and Technology Innovation, vol. 8, pp. 25-37, 2018.

[6] J. Liu, B. Ma, and H. Zhao, "Combustion parameters optimization of a diesel/natural gas dual fuel engine using genetic algorithm," Fuel, vol. 260, p. 116365, 2020.

[7] X. Lü, Y. Wu, J. Lian, Y. Zhang, C. Chen, P. Wang, et al., "Energy management of hybrid electric vehicles: A review of energy optimization of fuel cell hybrid power system based on genetic algorithm," Energy Conversion and Management, vol. 205, p. 112474, 2020.

[8] Z. Jalali, E. Noorzai, and S. Heidari, "Design and optimization of form and façade of an office building using the genetic algorithm," Science and Technology for the Built Environment, vol. 26, pp. 128-140, 2020.

[9] D. P. Augustine and P. Raj, "Performance Evaluation of Parallel Genetic Algorithm for Brain MRI Segmentation in Hadoop and Spark," Indian Journal of Science and Technology, vol. 9, 2016.

[10] P. Cai, Y. Cai, I. Chandrasekaran, and J. Zheng, "Parallel genetic algorithm based automatic path planning for crane lifting in complex environments," Automation in Construction, vol. 62, pp. 133-147, 2016.

[11] Z.-K. Feng, W.-J. Niu, J.-Z. Zhou, C.-T. Cheng, H. Qin, and Z.-Q. Jiang, "Parallel multi-objective genetic algorithm for short-term economic environmental hydrothermal scheduling," energies, vol. 10, p. 163, 2017.

[12] X.-Y. Zhang, J. Zhang, Y.-J. Gong, Z.-H. Zhan, W.-N. Chen, and Y. Li, "Kuhn–Munkres parallel genetic algorithm for the set cover problem and its application to large-scale wireless sensor networks," IEEE Transactions on Evolutionary Computation, vol. 20, pp. 695-710, 2016.

[13] E. Alba and J. M. Troya, "A survey of parallel distributed genetic algorithms," Complexity, vol. 4, pp. 31-52, 1999.

[14] J. Kacprzyk and W. Pedrycz, Springer handbook of computational intelligence: Springer, 2015.

[15] H. S. Stone, "Multiprocessor scheduling with the aid of network flow algorithms," IEEE transactions on Software Engineering, pp. 85-93, 1977.

[16] Y. Jiang, "A survey of task allocation and load balancing in distributed systems," IEEE Transactions on Parallel and Distributed Systems, vol. 27, pp. 585-599, 2015.

[17] P. K. Yadav, M. Singh, and K. Sharma, "Task Allocation Model for Reliability and Cost optimization in Distributed Computing System," International Journal Of Modeling, Simulation, and Scientific Computing, vol. 2, pp. 131-149, 2011.

[18] A. Y. Hamed, "Task allocation for minimizing cost of distributed computing systems using genetic algorithms," International Journal of Advanced Research in Computer Science and Software Engineering, vol. 2, 2012.

[19] M. Chaubey and M. Gupta, "DESIGNING A TASK ALLOCATOR FRAMEWORK FOR DISTRIBUTED COMPUTING," International Journal of Advanced Research in Computer Science, vol. 10, 2019.

[20] M. Akbari, "An efficient genetic algorithm for task scheduling on heterogeneous computing systems based on TRIZ," Journal of Advances in Computer Research, vol. 9, pp. 103-132, 2018.

[21] M. Hosseini, "A new Shuffled Genetic-based Task Scheduling Algorithm in Heterogeneous Distributed Systems," Journal of Advances in Computer Research, vol. 9, pp. 19-36, 2018.

[22] B. Barzegar, S. Habibian, and M. Fazlollah Nejad, "Heuristic algorithms for task scheduling in Cloud Computing using Combined Particle Swarm Optimization and Bat Algorithms," Journal of Advances in Computer Research, vol. 10, pp. 83-95, 2019.

[23] P. Yadav, M. Singh, and K. Sharma, "An optimal task allocation model for system cost analysis in heterogeneous distributed computing systems: A heuristic approach," International Journal of computer applications, vol. 28, pp. 30-37, 2011.

[24] D. Sudholt, "Parallel evolutionary algorithms," in Springer Handbook of Computational Intelligence, ed: Springer, 2015, pp. 929-959.

[25] E. Alba, "Parallel evolutionary algorithms can achieve super-linear performance," Information Processing Letters, vol. 82, pp. 7-13, 2002.

[26] G. Sharma and J. Martin, "MATLAB®: a language for parallel computing," International Journal of Parallel Programming, vol. 37, pp. 3-36, 2009.

[27] G. Attiya and Y. Hamam, "Hybrid Algorithm for Mapping Parallel Applications in Distributed Systems," in Fifth International Conference on PRAM, Poland, 2003, pp. 7-10.

[28] G. Attiya and Y. Hamam, "Task allocation for maximizing reliability of distributed systems: A simulated annealing approach," Journal of parallel and Distributed Computing, vol. 66, pp. 1259-1266, 2006.

[29] Q.-M. Kang, H. He, H.-M. Song, and R. Deng, "Task allocation for maximizing reliability of distributed computing systems using honeybee mating optimization," Journal of Systems and Software, vol. 83, pp. 2165-2174, 2010.