Genetic Algorithms for Task Scheduling Problem

Fatma A. Omara and Mona M. Arafa

Abstract. The scheduling and mapping of the precedence-constrained task graph to the processors is considered one of the most crucial NP-complete problems in the parallel and distributed computing systems. Several genetic algorithms have been developed to solve this problem. The primary distinction among most of them is being the used chromosomal representation for a schedule. However, these existing algorithms are monolithic as they attempt to scan the entire solution space without consideration how to reduce the complexity of the optimization. In this chapter, two genetic algorithms have been developed and implemented. Our developed algorithms are genetic algorithms with some heuristic principles have been added to improve the performance. According to the first developed genetic algorithm, two fitness functions have been applied one after another. The first fitness function is concerned with minimizing the total execution time (schedule length) and the second one is concerned with the load balance satisfaction. The second developed genetic algorithm is based on task duplication technique to overcome the communication overhead. Our proposed algorithms have been implemented and evaluated using benchmarks. According to the evolution results, it found that our algorithms always outperform the traditional algorithms.

1 Introduction

The problem of scheduling task graphs of a parallel program onto parallel and distributed computing systems is a well-defined NP-complete problem

Fatma A. Omara
Computer Science Department, Faculty of Computer and Information
Cairo University
e-mail: f.omara@ffci-cu.edu.eg

Mona M. Arafa
Mathematics Dept., Faculty of Science Banha University
e-mail: m-h-banha@yahoo.com

A. Abraham et al. (Eds.): Foundations of Comput. Intel. Vol. 3, SCI 203, pp. 479–507.
springerlink.com

that has received a large amount of attention, and it is considered one of the most challenging problems in parallel computing [1]. This problem involves mapping a Directed Acyclic Graph (DAG), of collection of computational tasks and their data precedence, onto a parallel processing system. The goal of a task scheduler is to assign tasks to available processors such that precedence requirements between tasks are satisfied and in the same time the overall execution length (i.e., make span) is minimized [2]. Generally, the scheduling problem exists in two types: static and dynamic.

According to the static scheduling, the characteristics of a parallel program such as task processing times, communication, data dependencies, and synchronization requirement are known before execution [3] . According to the dynamic scheduling, a few assumptions about the parallel program can be made before execution, then, scheduling decisions have to be made on-the-fly [4]. The work in this chapter concerns static scheduling problem. One the other hand, a general taxonomy for static scheduling algorithms has been reviewed and discussed by Kwong and Ahmad [3]. Many task scheduling techniques have been developed with moderate complexity as a constraint, which is a reasonable assumption for general purpose development platforms [5, 6, 7, 8]. Generally, the task scheduling algorithms may be divided in two main classes; greedy and non-greedy (iterative) algorithms [9]. The greedy algorithms attempt to minimize the start time of the tasks of a parallel program only. This is done by allocating the tasks into the available processors without back tracking. On the other hand, the main principle of the iterative algorithms is that they depart from an initial solution and try to improve it.

The greedy task scheduling algorithms might be classified into two categories: algorithms with duplication and algorithms without duplication. One of the common algorithms in the first category is the duplication scheduling heuristic (DSH) algorithm [1], the main principles of the DSH algorithm are: the nodes are arranged in a descending order according to their static b-level and the start-time of the node on the processor without duplication of any ancestor is determined. After that the ancestors of the node is tried to duplicate into the duplication time slot until the slot is used up or the start-time of the node does not improve. One the other hand, one of the best algorithms in the second category is the Modified Critical Path (MCP) algorithm [10]. The MCP algorithm first computes the ALAPs of all the nodes, then create ready list containing ALAP times of the nodes in an ascending order. The ALAP of a node is computed by first computing the length of the Critical Path (CP) and then subtracting the b-level of a node from it. Ties are broken by considering min ALAP time of the children of a node. If the min ALAP time of the children is equal, ties are broken randomly.

According to MCP algorithm, the highest priority node in the list is picked and assign to a processor that allows the earliest start time using insertion approach. Recently, Genetic Algorithms (GAs) have been widely reckoned as a useful vehicle for obtaining high quality solutions or even optimal solutions for a broad range of combinatorial optimization problems including

task scheduling problem [2, 3]. Another merit of a genetic search is that its inherent parallelism can be exploited so as to further reduce its running time. The basic principles of GAs were firstly laid down by Holland [11], and after that they are well described in many texts. The Gas operate on a population of solutions rather than a single solution. The genetic search begins by initializing a population of individuals. Individual solutions are selected from the population then mate to form new solutions. The mating process implemented by combining or crossing over genetic material from two parents to form the genetic material for one or two new solutions, confers the data from one generation of solutions to the next. Random mutation is applied periodically to promote diversity. The individuals in the population are replaced by the new solutions. A fitness function, which measures the quality of each candidate solution according to the given optimization objective, is used to help determine which individuals are retained in the population as successive generations evolve [12]. There are two important but competing themes exist in a GA search; the need for selective pressure so that the GA is able to focus the search on promising areas of the search space, and the need for population diversity so that important information (particular bit values) is not lost [13, 14].

Recently, several GAs have been developed for solving the task scheduling problem, the primary distinction among them being the chromosomal representation of a schedule [15, 16, 17, 18, 19, 20, 2]. Two hybrid genetic algorithms called Critical Path Genetic Algorithm (CPGA) and Task Duplication Genetic Algorithm (TDGA) have been proposed in this chapter. Our developed algorithms show the effect of the amalgamation of the greedy algorithms with the genetic one. The first algorithm CPGA is based on how to use the ideal time of the processors efficiently, and reschedule the critical path nodes to reduce their start time. Finally, two fitness functions have been applied, one after another. The first fitness function is concerned with how to minimize the total execution time (schedule length), and the second one is concerned with the load balance satisfaction. The second algorithm TDGA is based on task duplication principle to minimize the communication overheads.

The reminder of this chapter is organized as follows: Section 2 gives a description for the model for task scheduling problem. An implementation of the standard GA is presented in Section 3. Our developed CPGA is introduced in section 4. Section 5 produces the details of our TDGA algorithm. A comparative study of our developed algorithms, MCP algorithm, DSH algorithm, and SGA algorithm are presented in Section 6. Conclusion is presented in Section 7.

2 Task Scheduling Problem Model

The model of the underline parallel system to be considered in this research work could be described as follows [3]: The system consists of a limited number

of fully connected homogeneous processors. Let a task graph G be a Directed, Acyclic Graph (DAG) composed of N nodes n_1, n_2,..., n_N, each node termed a task of the graph which in turn is a set of instruction that must be executed sequentially without preemption in the same processor. A node has one or more inputs. When all inputs are available, the node is triggered to execute. A node with no parent is called an entry node and a node with no child is called an exit node. The weight is called the computation cost of a node ni and is denoted by (n_i) weight. The graph also has E directed edges representing a partial order among the tasks. The partial order introduced a precedence-constrained DAG and implies that if $n_i \rightarrow n_j$, then n_j is a child, which cannot start until its parent n_i finishes. The weight on an edge is called communication cost of the edge and is denoted by $c(n_i, n_j)$. This cost is incurred if ni and nj are scheduled on different processors and is considered to be zero if n_i and n_j are scheduled on the same processor. If a node n_i is scheduled to processor P, the start time and finish time of the node are denoted by $ST(n_i, p)$ and $FT(n_i, p)$ respectively. After all nodes have been scheduled, the schedule length is defined as $\max FT(n_i, p)$ across all processors. The objective of the task scheduling problem is that how to find an assignment and the start times of the tasks to processors such that the schedule length is minimized and, in the same time, the precedence constrains are preserved. A Critical Path (CP) of a task graph is defined as the path with the maximum sum of node and edge weights from an entry node to an exit node. A node in CP is denoted by CP Nodes (CPNs). An example of a DAG is represented in Figure1 with CP is drawn in bolt.

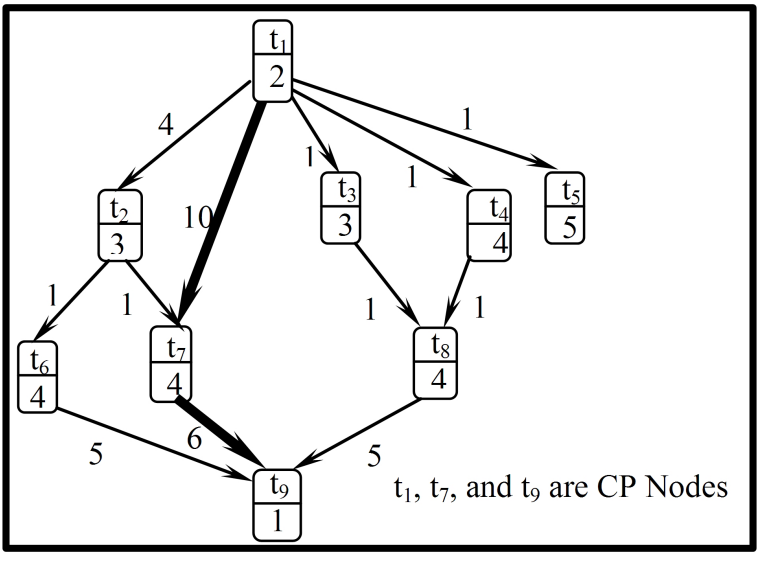


Fig. 1 Example of DAG, where t_1, t_7, and t_9 are CP Nodes

Table 1 Selected Benchmark Programs

Benchmarks programs	No tasks	Source	Note
Pg1	100	[22]	Random Graphs
Pg2	90	[22]	Robot Control program
Pg3	98	[22]	Sparse Matrix Solver

3 The Developed Genetic Algorithms

Before presenting the details of our developed algorithms, some principles which are used in the design are discussed.

Definition 1. *(Data Arrival Time) Any task cannot be start unit all parents have been finished. Let P_j be the processor on which the $k-th$ parent task t_k of task t_i is scheduled. Data Arrival Time (DAT) of t_i on a processor P_i is defined as:*

$$DAT = \max(FT(t_k, P_j) + c(t_i, t_k)), k = 1, 2, ..., No - parent \quad (1)$$

Where, $No - parent$ is the number of parents of t_i,

If $(i = j)$ then $c(t_i, t_k) = 0$

The parent task that maximizes the above expression is called the favorite predecessors of t_i and it is denoted by $favpred(t_i, P_j)$. The benchmark programs which have been used to evaluate our algorithms are listed in Table (1).

3.1 Standard Genetic Algorithm - SGA

The SGA has been implemented first. This algorithm is started with an initial population of feasible solution. Then, by applying some operators, the best solution could be finding through some generations. The selection of the best solution is determined according to the value of fitness function. According to this SGA, the chromosome is divided into two sections; mapping and scheduling. The mapping section contains the processors indices where tasks to be run on it. The schedule section determines the sequence for processing of the

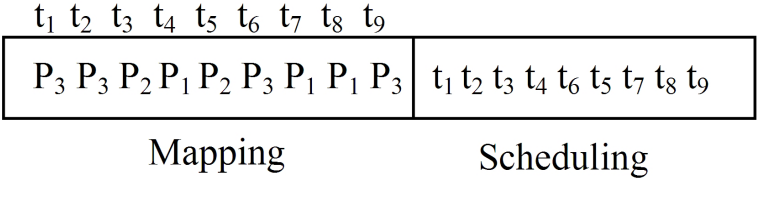


Fig. 2 Representation of a Chromosome

tasks. Figure 2 shows an example of such representation of the chromosome. Where, tasks t_4, t_7, t_8 will be scheduled on processor P_1, tasks t_3, t_5 will be scheduled on processor P_2, and the tasks t_1, t_2, t_6 and t_9 will be scheduled on processor P_3. The length of the chromosome is linear proportional to the number of tasks.

Genetic Formulation of SGA

Initial Population

The initial population is constructed randomly. The first part of the chromosome (i.e. mapping) is chosen randomly from 1 to No-Processors, where the No-Processors is the number of processors in the system. The second part (i.e. schedule) is generated randomly such that the topological order of the graph is preserved. The Pseudo Code of *The Task Schedule* using SGA is as follow:

Fitness Function

The main objective of the scheduling problem is to minimize the schedule length of a schedule.

$$Fitness - Function = (\frac{a}{Slength}) \tag{2}$$

Where a is a constant and $Slength$ is the schedule length which is determined by the following equation:

$$Slength = \max(FT[t,]), i = 1, ..., K_{noTask} \tag{3}$$

Function Schedule length

1. $\forall RT[P_j] = 0$ //RT is the ready time of the processors.

2. Let LT be a list of tasks according to the topological order of DAG.

3. **For** i=1 to NoTasks **Do**

// NoTasks is number of tasks

(a) Remove the first task t_i form list LT.

(b) **For** j = 1 to NoProcessors **Do**
// NoProcessors is number of Processors.

If t_i is scheduled to processor P_j

$$ST[t_i] = \max(RT[P_j], DAT(t_i, P_j))$$

$$FT[t_i] = ST[t_i] + weight[t_i]$$

$$RT[P_j] = FT[t_i]$$

Endif, Endfor, Endfor.

$Slength = \max(FT)$.

Example: By considering the chromosome represented in Figure 2 as a solution of a DAG represented in Figure 1, the Fitness Time function defined by equation 3 has been used to calculate the schedule length (see Figure 3).

Genetic Operators

In order to apply crossover and mutation operators, the selection phase should be applied firstly. This selection phase used to allocates reproductive trials to

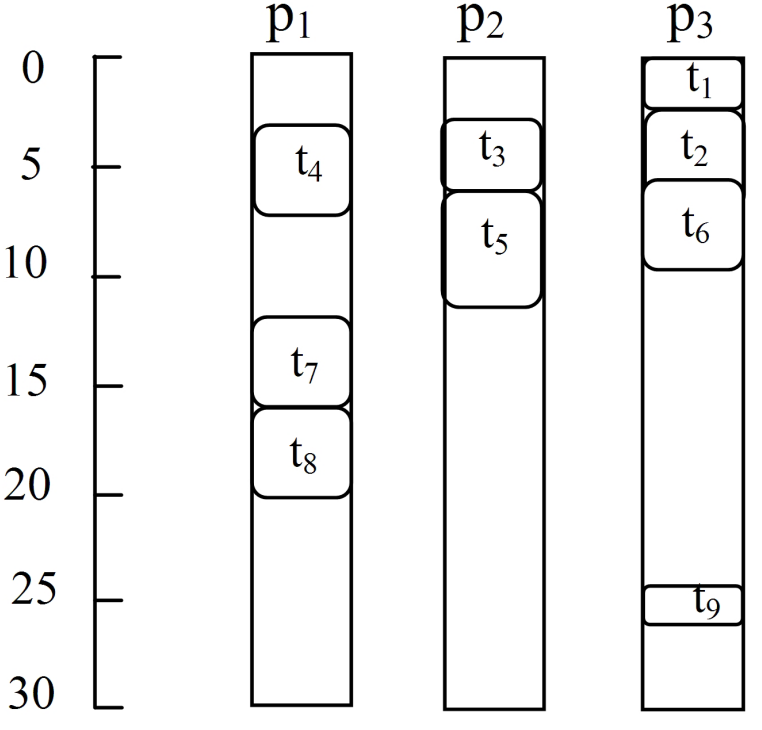


Fig. 3 The Schedule Length

Table 2 A comparison between roulette wheel and tournament selection

Benchmarks programs	Roulette Wheel Selection	Tournament Selection
Pg1	301.6	283.7
Pg2	1331.6	969
Pg3	585.8	521.8

chromosomes according to their fitness. There are different approaches could be applied in the selection phase. According to the work in this chapter, fitness-proportional roulette wheel selection [23] and tournament selection [24] are compared such that the best method is used (i.e., produce the shortest schedule length). In the roulette wheel selection, the probability of selection is proportional to an chromosome's fitness. The analogy with a roulette wheel arises because one can imagine the whole population forming a roulette wheel with the size of any chromosome's slot proportional to its fitness. The wheel is then spun and the figurative **ball** thrown in. the probability of the ball coming to the rest in any particular slot is proportional to the arc of the slot and thus to the fitness of the corresponding chromosome. In binary tournament selection, two chromosomes are picked at random from the population. Whichever has the higher fitness is chosen. This process is repeat number of population size.

Table (2) contains the comparing results between these two selection methods using 4 processors for each benchmark program listed in Table1. According to the results listed in Table 2, the tournament selection method produce schedule length is smaller than the roulette wheel selection. Therefore, the tournament selection method is used in the work of this chapter.

Crossover Operator. Each chromosome in the population is subjected to crossover with probability μ. Two chromosomes are selected from the population, and a random number $RN \in [0, 1]$ is generated for each chromosome. If $RN < \mu$, these chromosomes are applied using one of the two kinds of the crossover operators; single point crossover and order crossover operators. Otherwise, these chromosomes are not changed. The pseudo code of the crossover function is as follows.

Function Crossover

1. Select two chromosomes chrom1 and chrom2

2. Let P a random real number between 0 and 1

3. **If** $P < 0.5$ /* operators probabilty

Crossover-Map(chrom1, chrom2)

Else

Crossover-Order(chrom1, chrom2).

According to the crossover function, one of the crossover operators is used.

Crossover Map. When the single crossover is selected, it is applied to the first part of the chromosome. By given two chromosomes a random integer number called the crossover point is generated from 1 to No-Tasks. The portions of the chromosomes lying to the right of the crossover point are exchanged to produce two offsprings (see Figure 4).

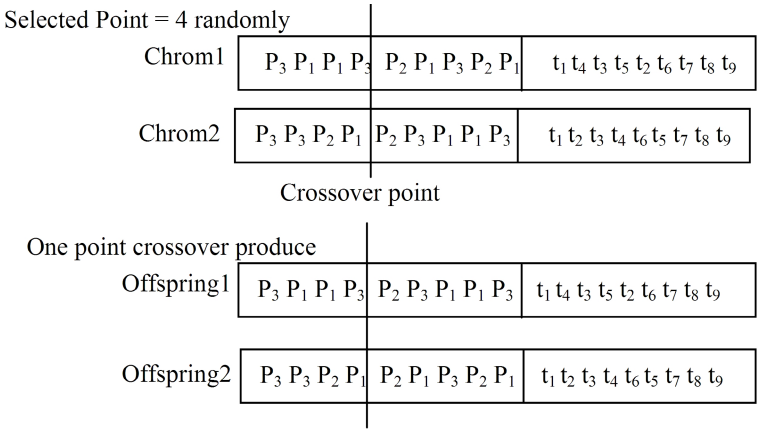


Fig. 4 One point crossover operator

Order Crossover. When the order crossover operator is applied to the second part of the chromosome, a random point is chosen. First pass the left segment from the chrom1 to the offspring, and then construct the right fragment of the offspring according to the order of the right segment of chrom2 crossover operator is given in (see Figure 5 as an example).

Mutation Operator. Each position in the first part of the chromosome is subjected to mutation with probability . Mutation involves changing the

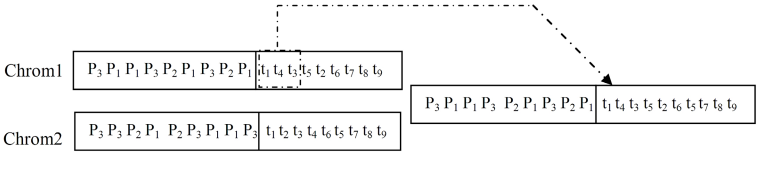


Fig. 5 Order crossover operator

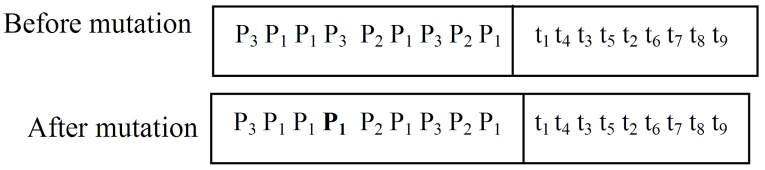


Fig. 6 Mutation Operator

assignment of a task from one processor to another. Figure 6 illustrate the mutation operation on chrom1. After the mutation operator is applied, the assignment of t_4 is changed from processor P_3 to processor P_1.

4 The Critical Path Genetic Algorithm (CPGA)

Our developed CPGA algorithm is considered a hybrid of GA principles and heuristic algorithms principles (e.g., given priority of the nodes according to ALAPlevel). On the other hand, the same principles and operators which are used in SGA algorithm have been used in the CPGA algorithm. The encoding of the chromosome is the same as in SGA, but in the initial population the second part (schedule) of the chromosome can be constructed using one of the following ways:

1. The schedule part is constructed randomly as in SGA.
2. The schedule part is constructed using ALAP.

These two ways have been applied using benchmark programs listed in Table 1 with four processors. According to the comparative results listed in Table (3), it is found that the priority of the nodes by ALAP method outperforms the random one in the most cases.

Table 3 A comparison between Random and Order ALAP Order methods

Benchmarks programs	Random Order	ALAP Order
$Pg1$	183.4	152.3
$Pg2$	848.5	826.4
$Pg3$	301.8	293.8

By using ALAP, the second part of the chromosomes is become static along the population. So, the crossover operators are restricted to the one point crossover operator. Three modifications have been applied in the SGA to improve the scheduling performance. These modifications are: (1) Reuse idle time, (2) Priority of the CPNs, and (3) Load balance.

Function Test-Slots

1. Let LT be a list of ready tasks
2. Initially the deal-time list is empty, S-ideal-time=0, and E-ideal-time=0
3. While the list LT is not empty, get a task t_i from the head of the list
(a) $Min = ST = \inf$
(b) **For** each processor P_j

If t_i is scheduled to P_j.

Let $thisST$ = the start time of t_i on P_j

If $thisST > MinST$ **Then** $MinST = thisST$

If the idealtime list of P_j is not empty

For each timeslot of the idealtime list

If $(Eidealtime - Sidealtime) <= weight[t_i]$ & $DAT(t_i, P_j) > Sidealtime$

Then schedule t_i in the idealtime and update the Sidealtime and Eideal-time

Let sttime be the start time of the task t_i equal to Sidealtime.

End If

(c) **If** $sttime > MinST$ **Then**

MinST =sttime.

Example. Suppose the schedule represented in Figure (3). The processor P_1 has an ideal time slot; the start of this ideal time (S-ideal-time) is equal to 7 while its end time (E-ideal-slot) is equal to 12. On the other hand, the weight (t_S) =4 and $DAT(t_S, P_1) = S - ideal - slot = 7$.By applying the modification, t_S can be rescheduled to start at time 7. The final schedule length according to this modification becomes 23 instead of 26 (see Figure 7).

Priority of CPNs Modification

According to the second modification, another optimization factor is applied to recalculate the schedule length after giving high priorities for the (CPNs) such that they can start as early as possible. This modification is implemented using a function called Reschedule-CPNs Function. The pseudo code of this function is as follows:

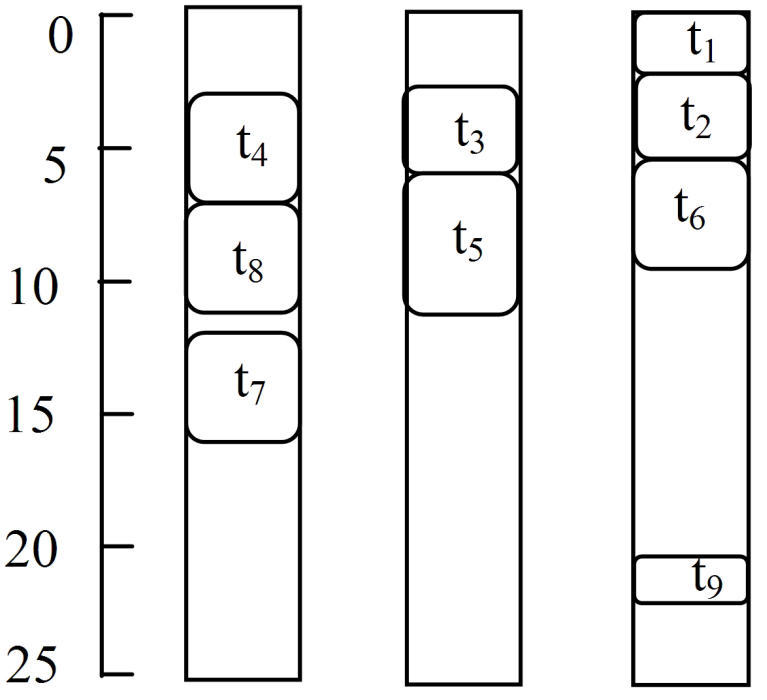


Fig. 7 The schedule after applying the test slots function is reduced from 26 to 23

Function Reschedule-CPNs

1. Determine the CP and make a list of CPNs

2. **While** the list of CPNs is not empty **DO**

- Remove the task t_i from the list

- Let $VIP = favpred(t_i, P_j)$

If VIP is assigned to processor P_j

Then The task t_i is assigned to processor p_j

End If

Example. We apply the Reschedule-CPNs Function in the scheduling presented in Figure 7. According to the DAG in presented in Figure 1, it is found that the CPNs are t1, t7, and t_9. t_1 is the entry node and it has no predecessor and the favpred of the t_7 is the task t_1. The task t_7 is scheduled

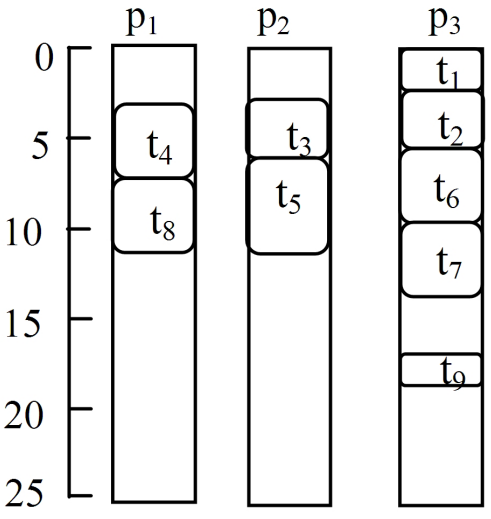


Fig. 8 The schedule after applying the reschedule of the CPNs function is reduced from 23 to 17

to processor P_1. Also the favpred of t_9 is t_8, but in the same time it starts early on the processor P_3, so t_9 is not moved. The final schedule length is reduced to 17 instead of 23(see Figure 8).

Load Balance Modification

Because the main objective of the task scheduling is to minimize the schedule length, it is found that several solutions can give the same schedule length, but load balance between processors might be not satisfied in some of them. The aim of load balance modification is that how to obtain the minimum schedule length and, in the same time, the load balance is satisfied. This has been satisfied by using two fitness functions one after another instead of one fitness function. The first fitness function concerns with minimizing the total execution time, and the second fitness function is used to satisfy load balance between processors. This function is proposed in [25] and it is calculated by the ratio of the maximum execution time (i.e. schedule length) to the average execution time over all processors.

If the execution time of processor P_j is denoted by $Etime[P_j]$, then the average execution time over all processors is:

$$avg = \sum_{j=1}^{NoProcessor} \frac{Etime[P_j]}{NoProcessors} \tag{4}$$

So, the load balance is calculated as:

$$LoadBalance = \frac{Slength}{Avg} \tag{5}$$

Supposing two task scheduling solutions are given in Figure 9 (a,b). The schedule length of both solutions is equal to 23.

Solution a: $Avg = \frac{12+17+23}{3}$,

Loadbalance $= \frac{23}{17.33} \approx 1.326$

Solution b: $Avg = \frac{9+11+23}{3} \approx 14.33$,

$Loadbalance = \frac{23}{14.33} \approx 1.604$.

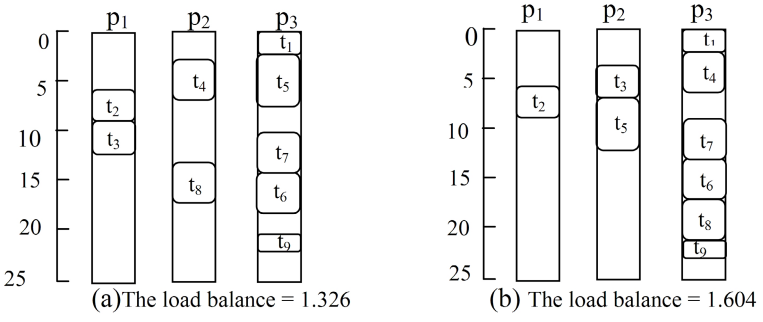


Fig. 9 according to balance fitness function solution (a) is better than solution (b)

According to the balance fitness function as shown in Figure (9), the solution (a) is better than the solution (b).

Adaptive μ_c and μ_m Parameters

Srinivas and patnaik [26] have proposed an adaptive method to tune crossover rate μ_c and mutation rate μ_m on the fly based on the idea of sustaining in diversity in a population without affecting its convergence properties. Therefore; the rate μ_c as:

$$\mu_c = \frac{k_c(f_{max} - f_c)}{(f_{max} - f_{avg})} \tag{6}$$

And the rate μ_m is defined as:

$$\mu_m = \frac{k_m(f_{max} - f_m)}{(f_{max} - f_{avg})} \tag{7}$$

Where, f_{max} is the maximum fitness value, f_{avg} is the average fitness value f_c is the fitness value of the fitter chromosome for the crossover f_m is the fitness value of the chromosome to be mutated k_c and k_m are positive real constant less than 1.

Table 4 A comparison between static and dynamic μ_c, μ_m parameters

Benchmarks programs	Dynamic parameters	Static parameters
$Pg1$	148	152.3
$Pg2$	785.6	826.4
$Pg3$	288.2	293.8

The CPGA algorithm has been implemented into two versions: the first version is done using static parameters ($\mu_c = 0.8$ and $\mu_m = 0.02$) and the second version is done using adaptive parameters. Table 4 represents the comparison results between these two versions. According to the results, it found that using adaptive parameters (μ_c and μ_m) can help preventing a GA from getting stuck at local minima. So the adaptive method is batter than using static values of μ_c and μ_m.

5 The Task Duplication Genetic Algorithm (TDGA)

Even with an efficient scheduling algorithm, some processors might be ideal during the execution of the program because the tasks assigned to them might be waiting to receive some data from the tasks assigned to other processors. If the idle time slots of the waiting processor could be used effectively by identifying some critical tasks and redundantly allocating them in these slots, the execution time of the parallel program could be further reduced [27].

According to our proposed algorithm, a good schedule based on task duplication has been proposed. This proposed algorithm called Task Duplication Genetic Algorithm (TDGA) employs a genetic algorithm for solving the scheduling problem.

Definition 2. *At a particular scheduling step; for any task t_i on a processor P_i, if $STF(favpred(t_i, p_j)) + weight(favpred(t_i, p_j)) \leq EST(t_i, p_j)$ Then $EST(t_i, p_j)$ can be reduced by scheduling $favpred(t_i, p_j)$ to p_j. Therefore, this definition could be applied recursively upward the DAG to reduce the schedule length.*

Example. To clearfy the effect of the task duplication technique, consider a schedule presented in Figure 10(a) for DAG in Figure (1), the schedule length is equal to 21. If t_1 is duplicated to processor p_1 and p_2 the schedule length is reduced to 18 (see Figure 10(b)).

Genetic Formulation of The TDGA

According to our TDGA algorithm, each chromosome in the population consists of a vector of order pairs (t, p) indicates that task t is assigned to processor p. The number of order pairs in a chromosome may vary in length.

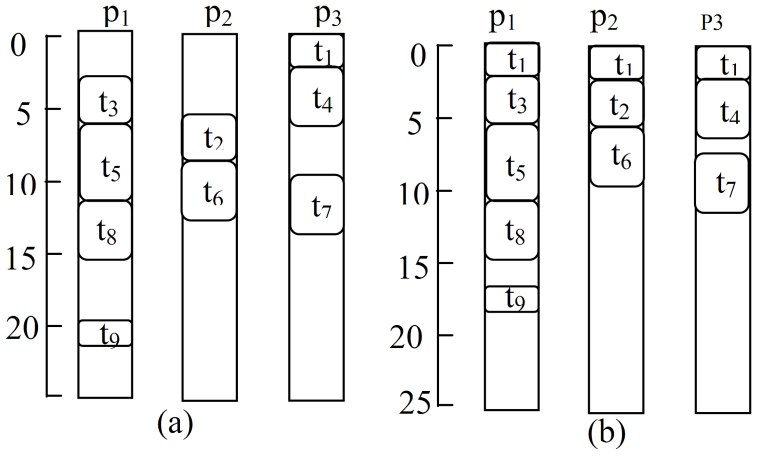


Fig. 10 (a) before duplication (schedule length=21) (b) After duplication (schedule length=18)

Fig. 11 An Example of the Chromosome

$$(t_2,P_1)(t_3,P_2)(t_4,P_1)(t_4,P_2)$$

An example of a chromosome is shown in Figure 11. The first order pair shows that task t_2 is assigned to processor P_1, and the second one indicates that task t_3 is assigned to processor P_2, etc.

According to the duplication principles, the same task may be assigned more than once to different processors without duplicating it in the same processor. If a task processor pair appears more than once on the chromosome, only one of the pairs is considered. According to Figure 11, the task t2 is assigned to processor P1 and P2.

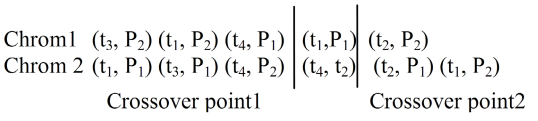
Definition 3. *(Invalid chromosomes) Invalid chromosomes are the chromosomes that not contain all DAG tasks. These invalid chromosomes might be generated.*

Initial Population. According to our TDGA algorithm, two methods to generate the initial population are applied. The first one, called Random Duplication (RD) and the second one called Heuristic Duplication (HD). According to RD, the initial population is generated randomly such that each task can be assigned to more than one processor.

According to HD, the initial population is initialized with randomly generated chromosomes, while each chromosome consists of exactly one copy of each task (i.e. no task duplication). Then, each task is randomly assigned to a processor. After that a duplication technique is applied by a function called *Duplication-Process*. The pseudo code of the *Duplication-Process* function is as follows:

Table 5 A comparison between the methods (HD and RD)

Benchmarks programs	HD	RD
$Pg1$	493.9	494.1
$Pg2$	1221	1269.5
$Pg3$	641.2	616.2



Two points 2 and 4 are generated randomly, two point crossover operator produce

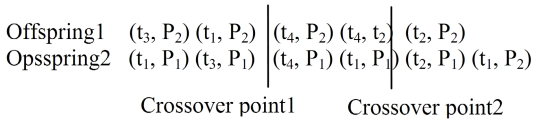


Fig. 12 Example of two point crossover operator

$(t_3, P_2)\ (t_1, \mathbf{P_2})\ (t_4, P_1)\ (t_1, P_1)\ (t_2, P_2)$ Chrom

$(t_3, P_2)\ (t_1, \mathbf{P_1})\ (t_4, P_1)\ (t_1, P_1)\ (t_2, P_2)$ Offspring

Fig. 13 Example of Mutation Operator

Function Duplicatin-Process

1.Compute SL for each task in the DAG
2.Make a list Slist of the tasks according to SL in descending order
3.Take the task t_i from Slist
4. **While** Slist is not empty.

If is assigned to processor ρ_i
if $favpred(t_i, \rho_i)$ is not assigned to ρ_i
if $(timeslot \geq weight(favpred(t_i, \rho_i))$

assigned $favpred(t_i, \rho_i)$ to ρ_i

According to the implementation results using two methods, it is found that the methods give nearly results. Therefore, the first method (HD) has been considered in our TDGA algorithm.

Fitness Function. Our fitness function is defined as 1/Slength, where Slength is defined as the maximum finishing time of all tasks of the DAG. The proposed GA assigns zero to an invalid chromosome as its fitness value.

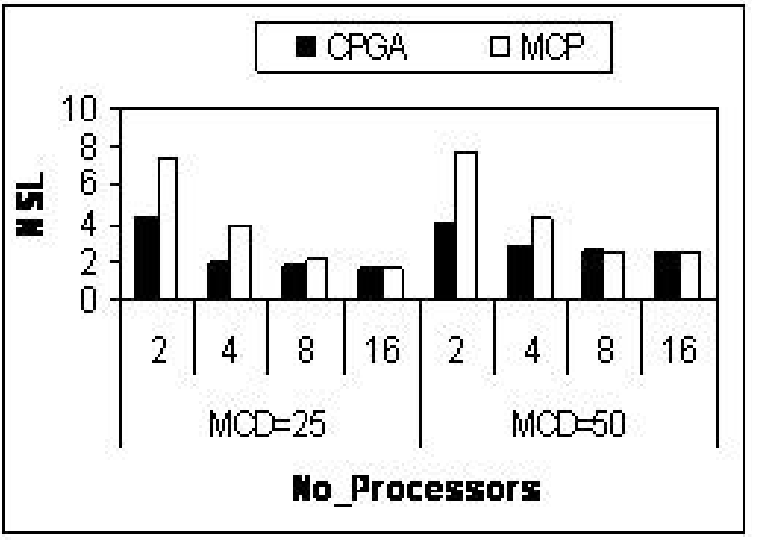


Fig. 14 NSL for $Pg1$ and MCD 25, 50

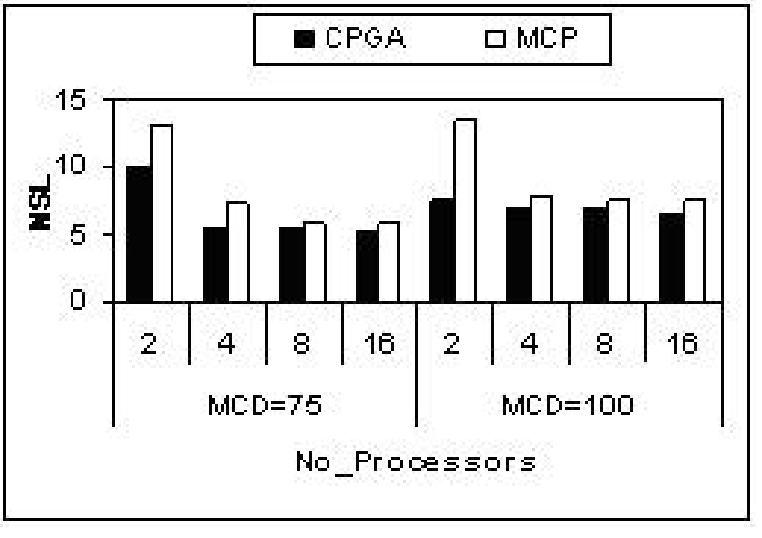


Fig. 15 NSL for $Pg1$ and MCD 75, 100

Genetic Operators: Crossover Operator. Two point crossover operator is used. Since each chromosome consists of a vector of task processor pair, crossover exchange substrings of pairs between two chromosomes. Two points are randomly chosen and the partitions between the points are exchanged between two chromosomes to form two offsprings. The crossover probability

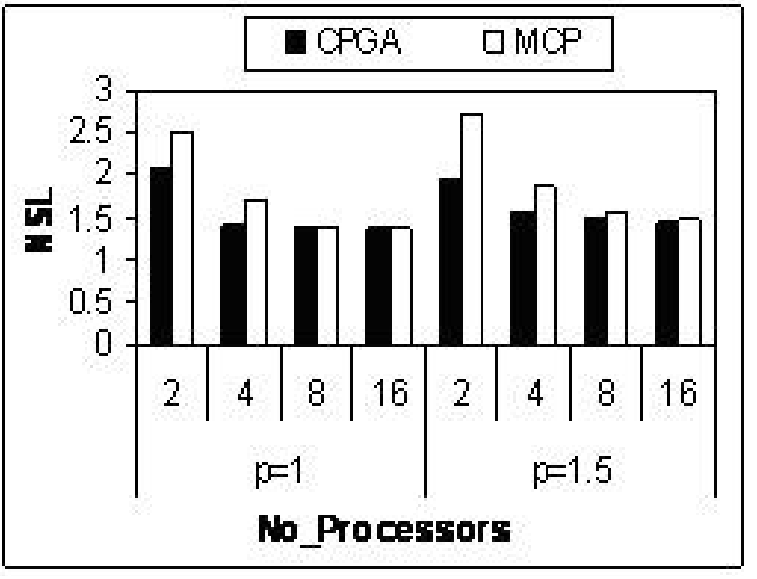


Fig. 16 NSL for $Pg2$ and two values of ρ

gives the probability that a pair of chromosome will undergo crossover. An example of two point crossover is shown in Figure 12.

Mutation Operator

The mutation probability indicates the probability that an order pair will be changed. If a pair is selected to be mutated, the processor number of that pair will be randomly changed. An example of mutation operator is shown in Figure 13.

6 Comparative Study and Performance Evaluation

To evaluate our proposed algorithms, we have implemented them using an Intel processor (2.6 GH) using c++ language and it is applied using different task graphs of specific benchmark applications programs as well as, a random one without communication delays which are listed in Table (1). All benchmark programs are taken from a Standard Task Graph (STG) archive [22]. The first two programs of This STG set consists of task graphs generated randomly $Pg1$, the second program is the robot control ($Pg2$) as an actual application programs and the last program is the sparse matrix solver ($Pg3$). Also, we consider the task graphs with random communication costs. These communication costs are distributed uniformly between 1 and a specified maximum communication delay (MCD). Also, the population size is considered 200 and the number of generations is considered 500 generation.

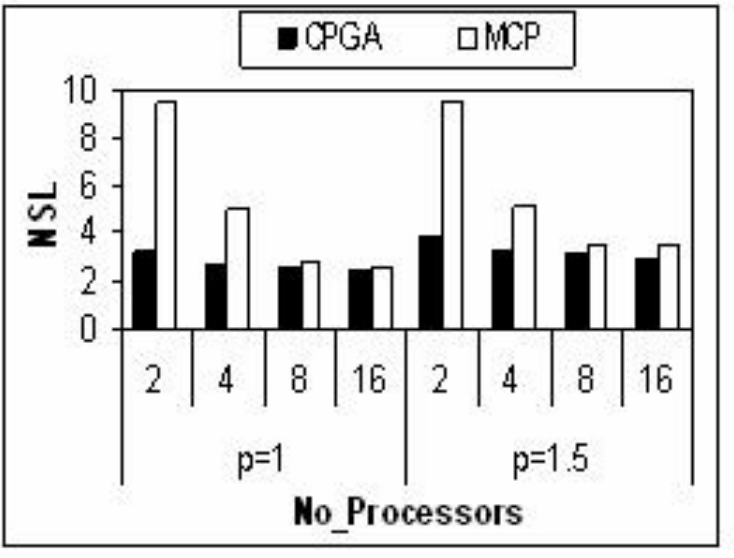


Fig. 17 NSL for $Pg3$ and two values of ρ

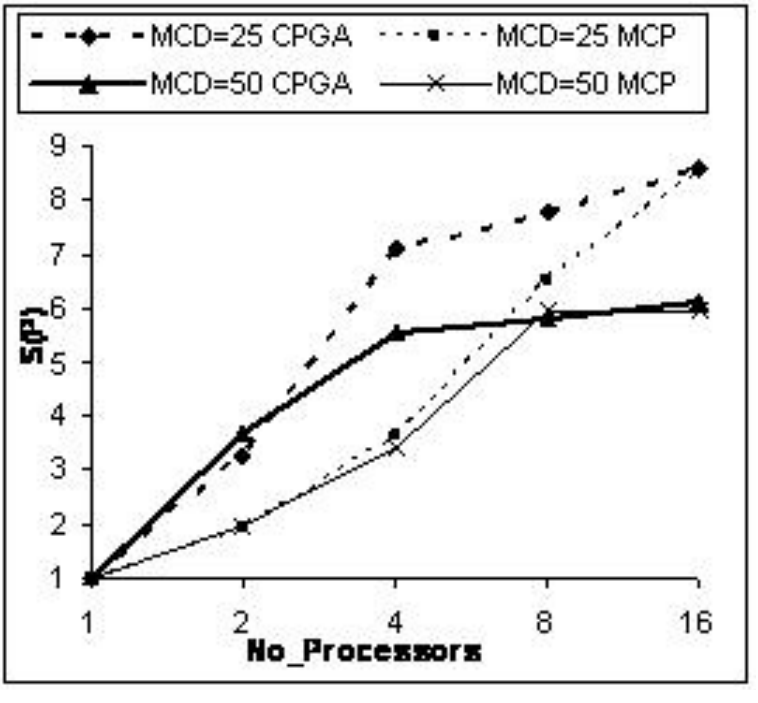


Fig. 18 Speedup for $Pg1$ and MCD 25 and 50

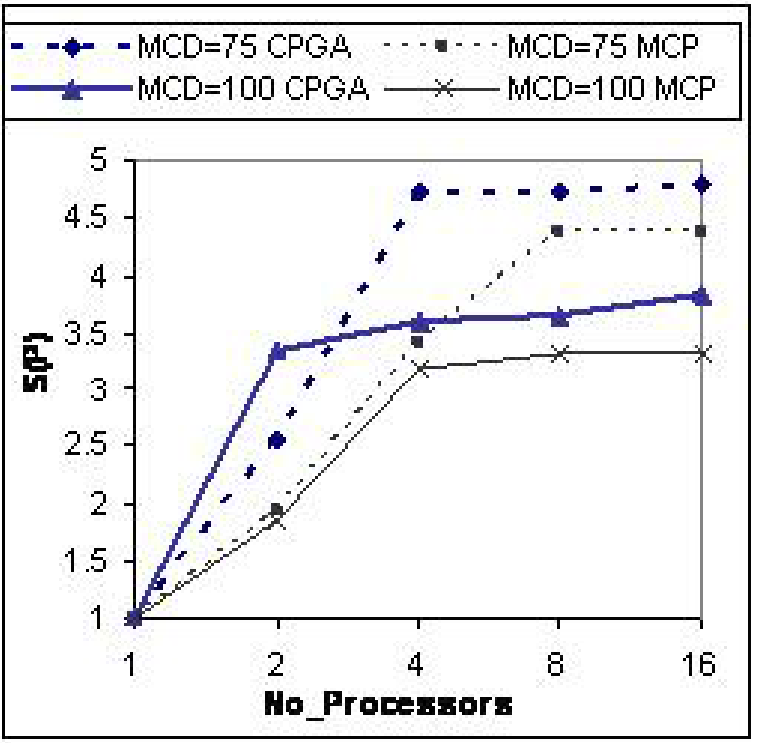


Fig. 19 NSpeedup for $Pg1$ and MCD 75and 100

6.1 The Developed CPGA Evaluation

The comparison has been done among our algorithm CPGA, SGA and one of the best greedy algorithms is called MCP algorithm. Firstly, a comparison among the CPGA, SGA and MCP algorithms with respect to the Normalized Schedule Length (NSL) with different number of processors has been done. The NSL is defined as [28]:

$$NSL = \frac{Slength}{\sum_{x \in CP}(Weight(n_i))} \tag{8}$$

Where SLength is the schedule length and weight (n_i) is the weight of the node n_i. The sum of computation costs on the CP represents a lower bound on the schedule length. Such lower bound may not always be possible achieve, and the optimal schedule length may be larger than this bound. Secondly, the performance of the CPGA, SGA and MCP are measured with respect to speedup [29]. The speedup is can be estimated as:

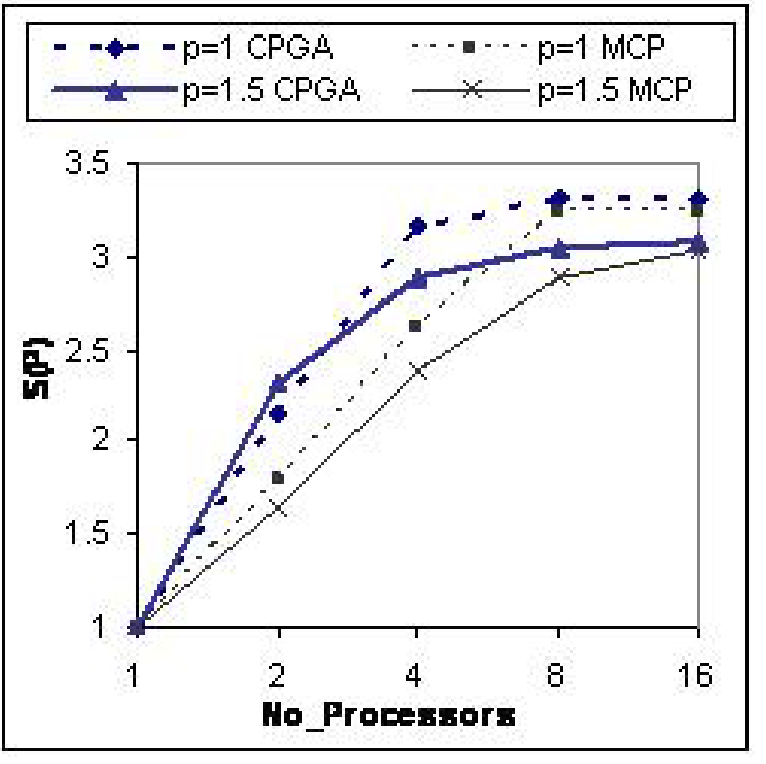


Fig. 20 Speedup for $Pg2$ and two values of ρ

$$S(p) = \frac{T1}{Tp} \tag{9}$$

where, $T(1)$ is the time required for executing a program on a uniprocessor computer and $T(P)$ is the time required for executing the same program on a parallel computer containing P processors. The NSL for CPGA and MCP algorithms using 2, 4, 8, and 16 processors for $Pg1$ and different MCD (25, 50, 75, and 100) are given in Figures (14 and 15). Also the NSL for Pg2 and Pg3 graphs with two different number of μ are given in Figures 16 and 17 respectively.

Figures (14, 15, 16, and 17) show that the performance of our proposed CPGA algorithm is always outperformed SGA and MCP algorithms. According to the obtained result, it is found that the NSL of all algorithms is increased when processor number is increased. Although, our CPGA is always the best, and it achieves lower bound when the communication delay is small.

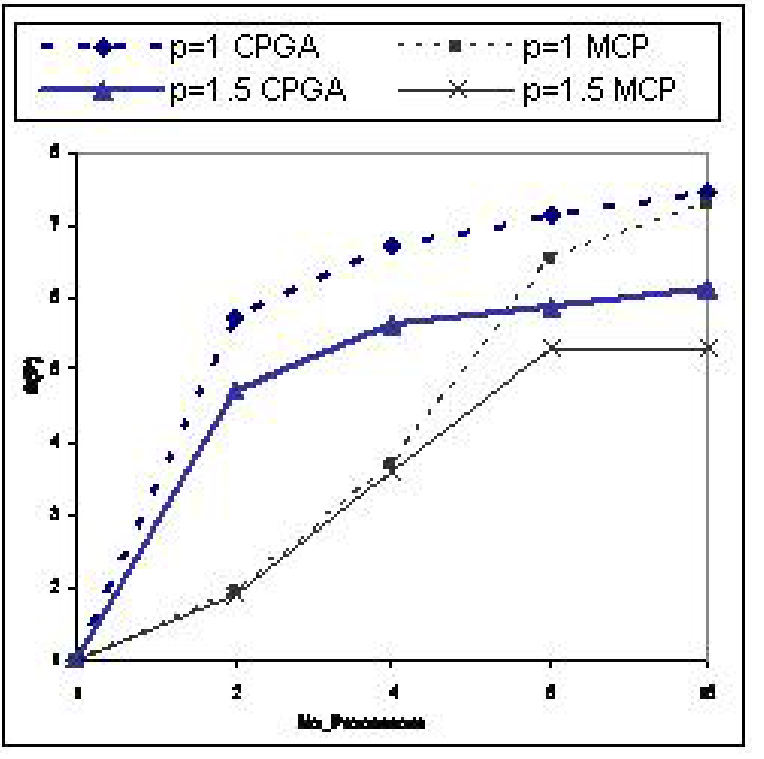


Fig. 21 Speedup for $Pg3$ and two values of ρ

6.2 The Developed TDGA Evaluation

To measure the performance of the TDGA, a comparison among the TDGA algorithm, SGA, and one of well known heuristic algorithm based on task duplication called DSH algorithm has been done with respect to NSL and speedup. To clarify the effect of task duplication in our TDGA algorithm, the same benchmark random and application programs $Pg1$, $Pg2$, and $Pg3$ listed in Table (1) have been used with high communication delay.

The NSL for TDGA, SGA, and DSH algorithms using 2, 4, 8, and 16 processors for $Pg1$ with two value of Communication Delay (CD) 100 and 200 is given in Figure 22. Also the NSL for bench mark application programs $Pg2$, and $Pg3$ is given in Figures 23, and 24.

According to the results in Figures (22, 23, and 24), it is found that our TDGA algorithm outperforms SGA and DSH algorithms especially when the number of communication, as well as, the number of processor increases.

The speedup of TDGA algorithm and DSH algorithm is given in Figures (25, 26, and 27) for $Pg1$, $Pg2$, and $Pg3$ programs respectively.

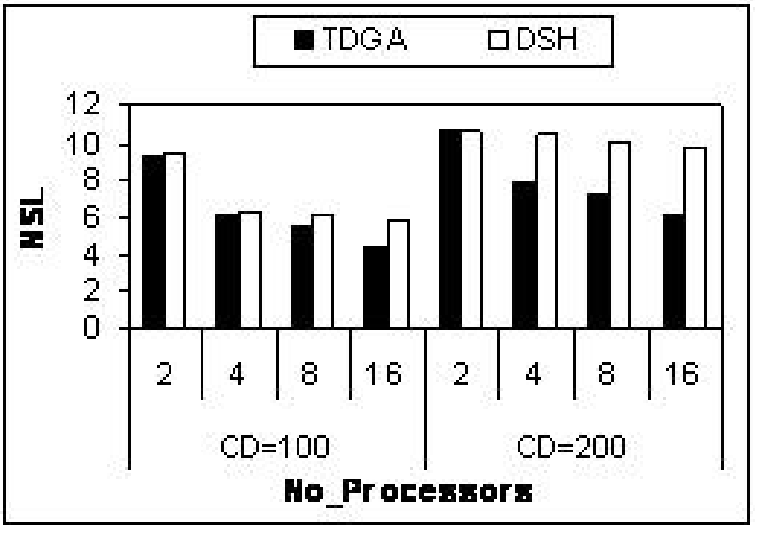


Fig. 22 NSL for Pg1 and CD =100 and 200

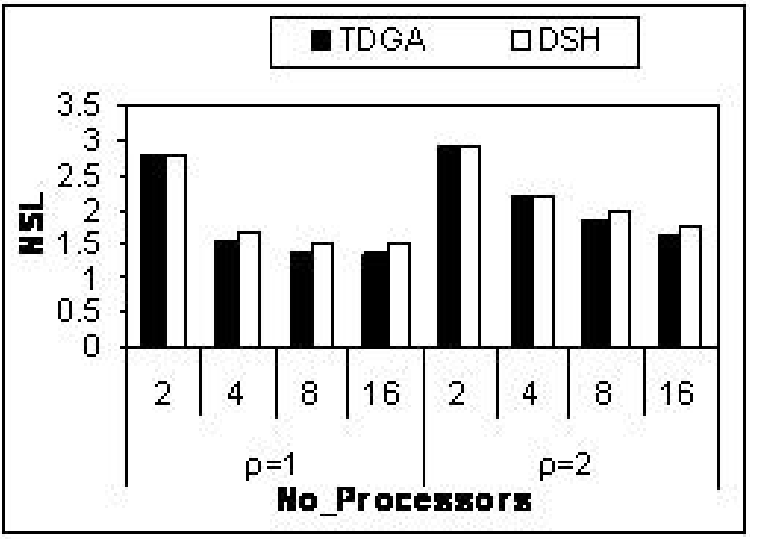


Fig. 23 NSL for Pg2 and ρ=1 and 2

The results reveal that the performance of the TDGA algorithm is always outperformed the DSH algorithm. Also, the TDGA speedup is nearly linear especially for random graphs

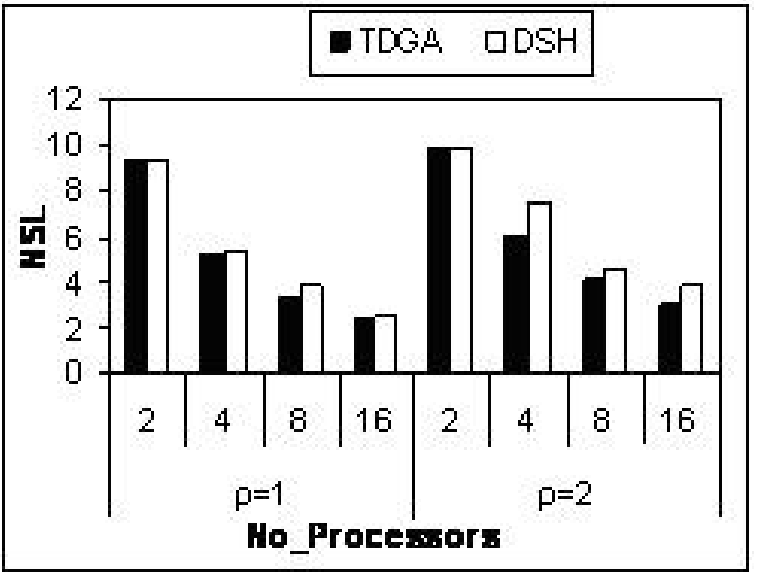


Fig. 24 NSL for Pg3 and ρ=1 and 2

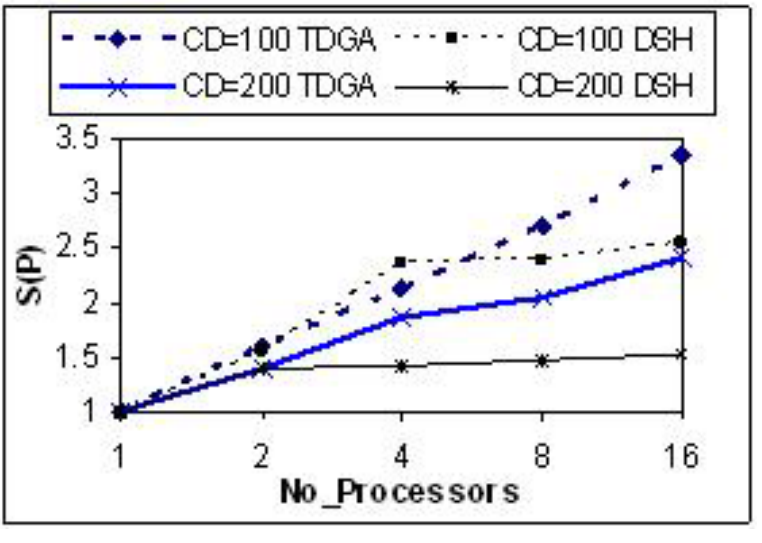


Fig. 25 Speedup for Pg1 and ρ=1 and 2

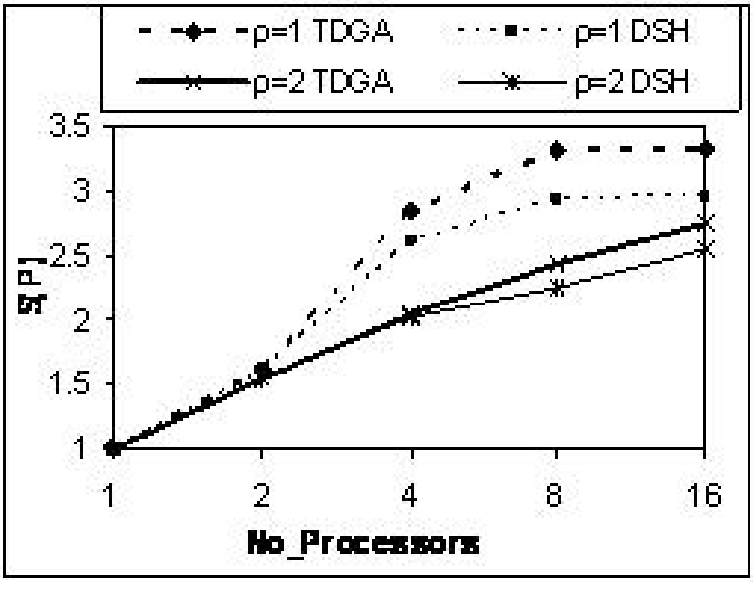


Fig. 26 Speedup for Pg2 and ρ=1 and 2

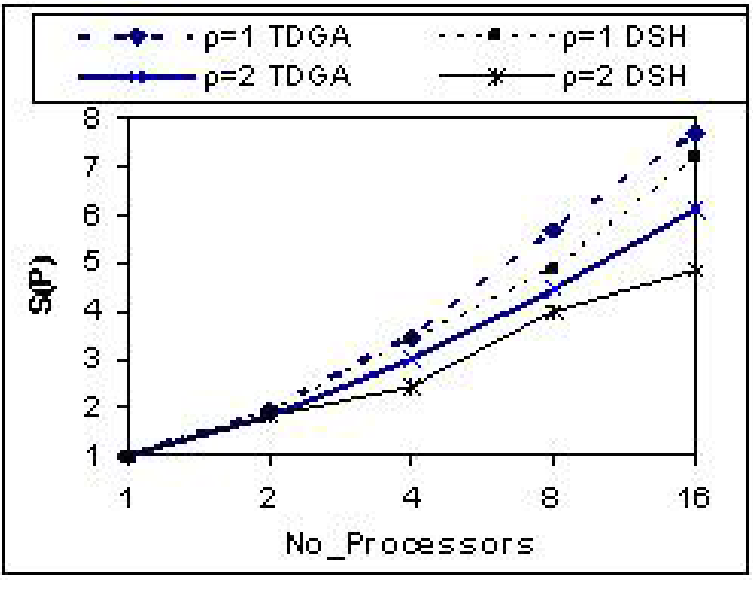


Fig. 27 Speedup for Pg3 and ρ =1 and 2

7 Conclusion

In this chapter, an implementation of a standard GA (SGA) to solve the task scheduling problem has been presented. Some modifications have been added to this SGA to improve the scheduling performance. These modifications are based on amalgamating heuristic principles with the GA principles. The new developed algorithm called Critical Path Genetic Algorithm (CPGA) is based on rescheduling the critical path nodes (CPNs) in the chromosome and then through different generations. Also, two modifications have been added. The first one concerns with how to use the idle time of the processors efficiently, and the second one concerns about to satisfy load balance among processors. The last modification is applied only when there are two or more scheduling solutions with the same schedule length are produced.

A comparative study among our CPGA, SGA algorithms and one of the standard heuristic algorithm called MCP algorithm have been presented using standard task graphs with considering random communication costs. The experimental studies show that the CPGA always outperform the SGA as well as the MCP algorithm in most cases. Generally, the performance of our CPGA is better than the SGA and MCP algorithms. According to task duplication technique, the communication delays are reduced and then minimizing the overall execution time, in the same time, the performance of the genetic algorithm is increased. The performance of the TDGA is compared with a traditional heuristic scheduling technique: DSH and SGA. The TDGA outperforms the DSH algorithm and SGA in most cases.

Acknowledgements. This research has been partially supported by funds from Cairo University, Egypt (Young Researcher Annual Grants).

References

1. El-Rewini, H., Lewis, T.G., Ali, H.H.: Task Scheduling in Parallel and Distributed Systems. Prentice-Hall International Editions (1994)
2. Wu, A.S., Yu, H., Jin, S., Lin, K.-C., Schiavone, G.: An Incremental Genetic Algorithm Approach to Multiprocessor Scheduling. IEEE Trans. Parallel and Distributed Systems 15, 824–834 (2004)
3. Kwok, Y., Ahmad, I.: Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors. ACM Computing Survey 31, 406–471 (1999)
4. Palis, M.A., Liou, J.C., Rajasekaran, S., Shende, S., Wei, S.S.L.: Online Scheduling of Dynamic Trees. Parallel Processing Letter 5, 635–646 (1995)
5. Sih, G.C., Lee, E.A.: A Compile-Time Scheduling Heuristic for Interconnection-Constrained Heterogeneous Processor Architectures. IEEE Trans. Parallel and Distributed Systems. 4, 75–87 (1993)
6. Kwok, Y., Ahmad, I.: Dynamic Critical Path Scheduling: An Effective Technique for Allocating Task Graphs to Multi-processors. IEEE Trans. Parallel and Distributed Systems. 7, 506–521 (1996)

7. Omara, F.A., Allam, A.: An Efficient Tasks Scheduling Algorithm for Distributed Memory Machines With Communication Delays. Information Technology Journal (ITJ) 4, 326–334 (2005)
8. Radulescu, A., van Gemund, A.J.C.: Low Cost Task scheduling for Distributed Memory Machines. IEEE Trans. Parallel and Distributed Systems 13, 648–658 (2002)
9. Bouvry, P., Chassin, J., Trystram, D.: Efficient Solutions for Mapping Parallel Programs. CWI-Center for Mathematics and computer science, Amsterdam, The Netherlands (1995) (published in Euro-Par)
10. Wu, M., Gajski, D.D.: Hypertool: A Programming aid for message-passing systems. IEEE Trans. Parallel Distributed Systems 1, 381–422 (1990)
11. Corman, T.H., Leiserson, C.E., Rivests, R.L.: Introduction to Algorithms. MIT Press, Cambridge (1990)
12. Holland, J.H.: Adaptation in Natural and Artificial Systems. Univ. Of Michigan Press, Ann Arbor (1975)
13. Levine, D.: A Parallel Genetic Algorithm for The Set Partitioning Problem, Ph.D. thesis in computer science, Department of Mathematics and computer science, IIIinois Institute of Technology, Chicago, USA (1994)
14. Back, T., Hammel, U., Schwefel, H.-P.: Evolutionary Computation: Comments on the History and Current State. IEEE Trans. Evolutionary Computation 1, 3–17 (1997)
15. Talbi, E.G., Muntean, T.: A new Approach for The Mapping Problem: A Parallel Genetic Algorithm (1993), `citessr.ist.psu.edu`
16. Ali, S., Sait, S.M., Benten, M.S.T.: GSA: Scheduling And Allocation Using Genetic Algorithm. In: Proceedings of the Conference on EURO-DAC with EURO WDHL 1994, Grenoble, pp. 84–89 (1994)
17. Hou, E.H., Ansari, N., Ren, H.: A Genetic Algorithm for Multiprocessor Scheduling. IEEE Trans. Parallel Distributed Systems. 5, 113–120 (1994)
18. Ahmed, I., Dhodhi, M.K.: Task Assignment using a Problem-Space Genetic Algorithm. Concurrency. Pract. Exper. 7, 411–428 (1995)
19. Kwok, Y.: High performance Algorithms for Compile-time Scheduling of Parallel Processors, Ph.D. Thesis, Hong Kong University (1997)
20. Tsuchiya, T., Osada, T., Kikuno, T.: Genetic-Based Multiprocessor Scheduling Using Task Duplication. Microprocessors and Microsystems 22, 197–207 (1998)
21. Alaoui, S.M., Frieder, O., EL-Ghazawi, T.A.: Parallel Genetic Algorithm for Task Mapping On Parallel Machine. In: Proc. of the 13th International Parallel Processing Symposium & 10th Symp. Parallel and Distributed Processing (IPPS/SPDP) Workshops, San Juan, Puerto Rico (April 1999)
22. Haghighat, A.T., Nikravan, M.: A Hybrid Genetic Algorithm for Process Scheduling in Distributed Operating Systems Considering Load Balancing. In: The IASTED Conference on Parallel and Distributed Computing and Networks (PDCN), Innsbruck, Austria (2005)
23. Blickle, T., Thiele, L.: A Mathematical Analysis of Tournament Selection. In: Proc. of the 6th International Conf. on Genetic Algorithms (ICGA 1995). Morgan Kaufmann, San Francisco (1995)
24. Kumar, S., Maulik, U., Bandyopadhyay, S., Das, S.K.: Efficient Task Mapping on Distributed Heterogeneous Systems for Mesh Applications. In: Proceedings of the International Workshop on Distributed Computing, Kolkata, India (2001)

25. Ahmad, I., Kwok, Y.: A New Approach to Scheduling Parallel Programs Using Task Duplication. In: Proc. of the 23rd International Conf. on Parallel Processing, North Carolina State University, NC, USA (August 1994)
26. http://www.Kasahara.Elec.Waseda.ac.jp/schedule/
27. Ahmad, I., Kwok, Y.: Benchmarking and Comparison of the Task Graph Scheduling Algorithms. Journal of Parallel and Distributed Computing 95, 381–422 (1999)
28. Akl, S.G.: Parallel Computation: Models and Methods. Prentice-Hall, Inc., Englewood Cliffs (1997)
29. Wilkinson, B., Allen, M.: Parallel Programming: Techniques and applications using Networked Workstations and Parallel Computers. Pearson Prentic Hall, London (2005)