Hybrid Algorithm for Multiprocessor Task Scheduling

Mostafa R. Mohamed[1], Medhat H. A. AWADALLA [2]

[1] Information Systems Department, Faculty of Computers and Information, Fayoum University, El Fayoum, Egypt

[2] Communication, Electronics and Computers Department, Faculty of Engineering, Helwan University, Helwan, Egypt

Abstract

Multiprocessors have become powerful computing means for running real-time applications and their high performance depends greatly on parallel and distributed network environment system. Consequently, several methods have been developed to optimally tackle the multiprocessor task scheduling problem which is called NP-hard problem. To address this issue, this paper presents two approaches, Modified List Scheduling Heuristic (MLSH) and hybrid approach composed of Genetic Algorithm (GA) and MLSH for task scheduling in multiprocessor system. Furthermore, this paper proposes three different representations for the chromosomes of genetic algorithm: task list (TL), processor list (PL) and combination of both (TLPLC). Intensive simulation experiments have been conducted on different random and real-world application graphs such as Gauss-Jordan, LU decomposition, Gaussian elimination and Laplace equation solver problems. Comparisons have been done with the most related algorithms like: list scheduling heuristics algorithm LSHs, Bipartite GA (BGA) [1] and Priority based Multi-Chromosome (PMC) [2]. The achieved results show that the proposed approaches significantly surpass the other approaches in terms of task execution time (makespan) and processor efficiency.

Keywords: *Multiprocessors, task scheduling, Genetic algorithm, makespan, parallel and distributed system, Modified List Scheduling Heuristic (MLSH).*

1. Introduction

The problem of scheduling a task graph of a parallel program onto a parallel and distributed computing system is a well-defined NP-hard problem that has received much attention, and it is considered one of the most challenging problems in parallel computing [3]. The scheduling problem has been addressed in several applications such as information processing, database systems, weather forecasting, image processing, fluid flow, process control, economics, operation research and real time high-speed stimulations of dynamical systems. The multiprocessor task scheduling problem considered in this paper is based on the deterministic model, which is the execution time of tasks and the data communication time between tasks that are assigned; and the directed acyclic task graph (DAG) that represents the precedence relations of the tasks of a parallel processing system [4]. The goal of such a scheduler is to assign tasks to available processors such that precedence requirements between tasks are satisfied and the overall length of time required to execute the entire program, the schedule length or makespan, is minimized.

Many heuristic approaches for task scheduling have been proposed [5–10]. The reason for such proposals is because the precedence constraints between tasks can be non-uniform therefore rendering the need for a uniformity solution. We assume that the parallel processor system is uniform and non-preemptive.

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 [11, 12]. Another merit of a genetic search is that its inherent parallelism can be exploited so as to further reduce its running time. Thus, several methods have presented to solve this problem based on GAs [13-16].

To tackle the multiprocessor task scheduling problem (MTSP), this paper presents two approaches: a modified list scheduling heuristic and hybrid approach composed of GA and MLSH. GA used three new different types of chromosomes: task list, processor list, and a combination of both types.

This paper is organized as follows: The multiprocessor task scheduling problem on the general models of a DAG is presented in section 2. Section 3 outlines the most related work to the theme of this paper. Section 4 proposes MLSH. Hybrid approach composed of genetic algorithm and MLSH comprising three different new types of chromosomes is presented in section 5. Genetic operators are presented in section 6. Simulated experiments and discussions are presented in section 7. Section 8 concludes the paper.

2. Multiprocessor task scheduling problem

Multiprocessor scheduling problems can be classified into many different classes based on the following characteristics:

- The number of tasks and their precedence.
- Execution time of the tasks and the communication cost which is the cost to transmit messages from a task on one processor to a succeeding task on a different processor (Communication cost between two tasks on the same processor is assumed to be zero).
- Number of processors and processors uniformity (A homogeneous multiprocessor system is composed of a set P = {P1… Pm} of 'm' identical processors.
- Topology of the representative task graph.

Directed Acyclic Graph (DAG) can represent applications executed within each multiprocessor system. A DAG G = (V,

E) consists of a set of vertices V representing the tasks to be executed and a set of directed edges E representing communication dependencies among tasks. The edge set E contains directed edges e_{ij} for each task $T_i \in V$ that task $T_j \in V$ depends on. The computation weight of a task is represented by the number of CPU clock cycles to execute the task. Given an edge e_{ij}, T_i is called the immediate predecessor of T_j and T_j is called the immediate successor of T_i. An immediate successor T_j depends on its immediate predecessors such that T_j cannot start execution before it receives results from all of its immediate predecessors. A task without immediate predecessors is called an entry-task and a task without immediate successors is called an exit-task. A DAG may have multiple entry tasks and one exit-task. Randomly generated task model, Gauss Jordan elimination, LU decomposition [17] and Laplace equation solver [6] task graphs are considered in this paper. Some of these task graphs are illustrated in Fig.1.

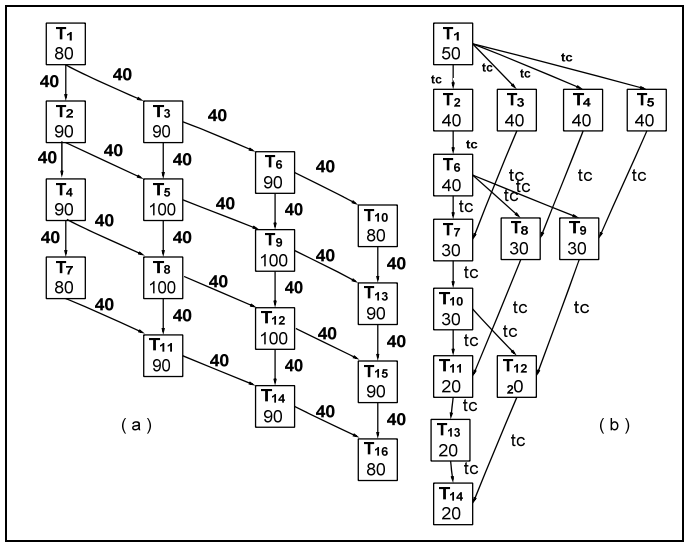

Fig.1. Description of task dependencies for (a) Laplace equation solver and (b) LU decomposition

3. RELATED WORK

Several approaches have been adopted to solve the multiprocessor task scheduling such as heuristic approaches [18-20], evolutionary approaches [2, 11, 13, 14, 15, 17, 21, 22] and hybrid methods [23, 24]. Kwok and Ahmad [25] presented a comprehensive review and classification of deterministic scheduling algorithms. Among the most common methods is a class of methods called list scheduling techniques. List scheduling techniques are widely used in task scheduling problems [26]. Insertion Scheduling Heuristic (ISH) and Duplication Scheduling Heuristic (DSH) are well-known list scheduling heuristic methods [27, 28]. ISH [27] is a list scheduling heuristic that was developed to optimize scheduling DAGs with communication delays. ISH extends a basic list scheduling heuristic from Hu [29] by attempting to insert ready tasks into existing communication delay slots. DSH [27] improved ISH by using task duplication to reduce the starting time of tasks within a schedule. DSH reduces inter-processor communication time by scheduling tasks redundantly to multiple processors.

The genetic-based methods have attracted a lot of researcher attention in solving the MTSP [2, 11, 13, 14, 15, 17, 21]. Genetic operators are the main differences of these genetic approaches, such as crossover and mutation. Using different crossover and mutation methods for reproducing the offspring is strongly dependent upon the chromosome representation which may lead to the production of legal or illegal solutions. Another important point in designing GA is the simplicity of the algorithm and complexity of evolutionary optimization process.

Hou et al. [13] reported that the results of GA were within 10% of the optimal schedules. Their results are based on task graphs with dependencies but without communication delays. The method proposed in [30], though it is very efficient, it does not search all the solution space. Due to the strict ordering that only the highest priority ready task can be selected for scheduling, there can be many valid schedules omitted from the search. Correa et al. [7] proposed modifications to the approach in [30] to broaden the search space to include all valid solutions. This modified approach was tested on task graphs that represent well-known parallel programs. Wu et al. [11] proposed a novel GA which allows both valid and invalid individuals in the population. This GA uses an incremental fitness function and gradually increases the difficulty of fitness values until a satisfactory solution is found. This approach is not scalable to large problems since much time is spent evaluating invalid individuals that may never become valid ones. Moore [31] applies parallel GA to the scheduling problem and compares its accuracy with mathematically predicted expected value. More GA approaches are found in [30, 32–35].

Another genetic-based multiprocessor scheduling method has been presented in [7]. The authors of this paper claimed that the task duplication is a useful technique for shortening the length of schedules. In addition, they added new genetic operators to the GA to control the degree of replication of tasks.

Some works tried to change the conventional approach of GA. They combined other problem solving techniques, such as divide and conquer mechanism with GA. A modified genetic approach called partitioned genetic algorithm (PGA) was proposed [15]. In PGA: the input DAG is divided into partial graphs using b-level partitioning algorithm and each of these separate parts is solved individually using GA. After that, a conquer algorithm cascades the subgroups and forms the final solution. In [36], a new GA called task execution order list (TEOL) was presented to solve the scheduling problem in parallel multiprocessor systems. The TEOL guarantees that all feasible search space is reachable with the same probability.

Some researchers proposed a combination of GAs and list heuristics [37-39]. Correa et al. [7] proposed a modified GA by the use of list heuristics in the crossover and mutation in a pure genetic algorithm. This method is said to dramatically improve the quality of the solutions that can be obtained with both pure genetic algorithm and pure list approach. Unfortunately, the running time is larger than the time of running pure genetic

algorithm. Therefore the aim of this paper is to reduce that time however GA is modified by the chromosomes representations.

4. The Proposed Modified List Scheduling Heuristic (MLSH)

List scheduling techniques assign a priority to each task to be scheduled and then sort the list of tasks in decreasing priority. As processors become available, the task with highest priority is processed and removed from the list. If two or more tasks have the same priority, the selection which is performed among the candidate tasks is typically random [26]. The problem with list scheduling algorithms is that the priority assignment may not always order the tasks for scheduling according to their relative importance. In MLSH, Priorities have been determined from DAG and then assigned to the tasks in such way that the important task will be assigned to the processor that eventually leads to a better scheduling. MLSH flowchart is illustrated in Fig.2.

In this paper, real-time tasks are considered. Each task is characterized by the following parameters:

- $t_s(T)$: is the starting time of task T of G.
- $t_s(T, P)$: is the starting time of task T on processor P.
- $t_f(T)$: is the finishing time of task T.
- w (T) : is the processing time of task T.

The algorithm starts by assigning levels for the tasks (the root task has level 0). The level of a task graph is defined as:

$$\text{Level}(Ti) = \begin{cases} 0 & ,\text{if Pred}(Ti) = \Phi \\ 1 + \max_{Tj \in \text{PRED}(Ti)} \text{Level}(Tj) & ,\text{otherwise} \end{cases} \quad (1)$$

where, Pred (Ti) is the set of predecessors of Ti.

Firstly, the Level function indirectly conveys precedence relations between the tasks. If the task T_i is an ancestor of task T_j, then Level (T_i) < Level (T_j). If there is no path between the two tasks, then there is no precedence relation between them and the order of their execution can be arbitrary [13, 40].

Secondly, the sequence of tasks' execution in each level is determined. For the root level (T1, T2 in Fig.3), if there is only one parent task, then it comes first.

If there is more than one parent tasks the number of the children for each parent in the next level is calculated and their parent has got a priority according to that number in a descending order. The parent with the highest number of children comes first (T_1 will be executed before T_2).

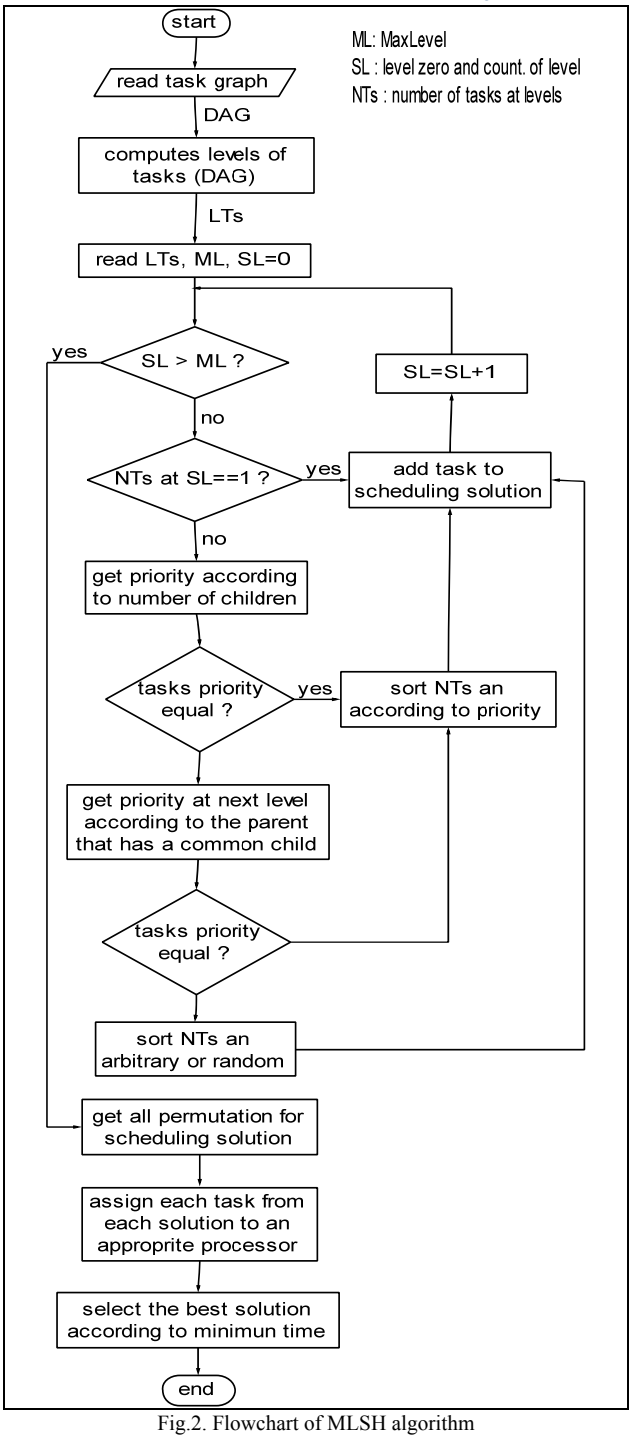

Fig.2. Flowchart of MLSH algorithm

If two or more parents have the same number of children (T3, T_4 and T_5) then the parent that has a common child is to be executed first (T_4 and T_5 will be executed before T_3). When two parents have the same common child, they will be listed in an arbitrary order (T_4 and T_5).

Thirdly, we assign each task to an appropriate processor to reach the minimum finishing time for all tasks according to equations (2-11).

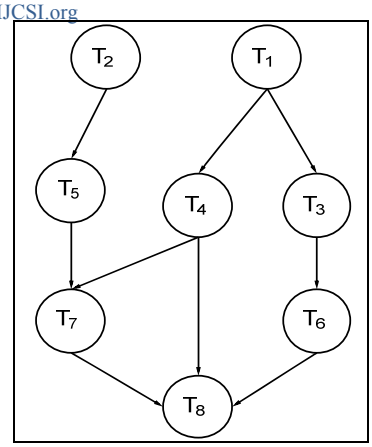

Fig.3. DAG with 8 tasks

5. The Proposed Hybrid approach.

The hybrid approach composed of GA and MLSH. Genetic algorithms try to mimic the natural evolution process and generally start with an initial population of chromosomes, which can either be generated randomly or based on some other algorithms. Here, the initial population has started based on MLSH. Three different types of chromosomes are developed to generate the genetic chromosome. In each generation, the population goes through the processes of fitness evaluation, selection, crossover and mutation. The following subsections will present these processes in full details.

5.1. Chromosomes

For task scheduling, a chromosome represents a solution to the scheduling problem. We present three different types of chromosomes for genetic algorithm: Task List (TL), Processor List (PL) and combination of them (TLPLC).

5.1.1. Chromosome construction using TL

Every chromosome is a permutation of the V tasks of the DAG. Each gene represents a position in the task list and the value of the gene is the index of the task at that position as shown in Fig.4. MLSH is used to form the chromosomes in the initial population.

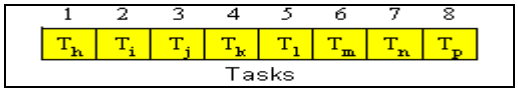

Fig.4. Chromosome that encodes task list

5.1.2. chromosome construction using PL

Every chromosome consists of V genes, each of which represents one task of the DAG. Assuming a consecutive numbering of the tasks and processors, starting with 1, gene i corresponds to task $T_i \in V$. The value of the gene corresponds to the index of the processor, 1 to P, to which the task is allocated as shown in Fig.5. The chromosomes have uniform distribution of the available number of processors.

case (1) : if T_j is a root task.

then, $t_s(T_j) = 0$

$$t_f(T_j) = t_s(T_j) + w(T_j) \quad (2)$$

case (2) : if T_j is not a root task, then :*

case (2.1) : if T_j depends only on one task (i.e. T_j depends on T_i)

then , $t_s(T_j) = t_f(T_i) + c(e_{T_iT_j})$ (3)

form (2) and (3),

$$t_f(T_j) = t_f(T_i) + c(e_{T_iT_j}) + w(T_j) \quad (4)$$

case (2.2) : if T_j depends only on one task (i.e. T_j depends on T_i) and T_j waits until T_k completes.

$$\text{then } t_s(T_j) = \begin{cases} t_f(T_k) & \text{if } c(e_{T_iT_j}) < t_f(T_k) - t_f(T_i) \\ t_f(T_i) + c(e_{T_iT_j}) & \text{othewise} \end{cases} \quad (5)$$

form (2) and (5),

$$t_f(T_j) = t_s(T_j) + w(T_j) \quad (6)$$

case (2.3) : if T_j depends on more than one task (i.e.T_j depends on more than T_i) and no waiting

$$\text{then } t_f(T_i) + c(e_{T_iT_j}) = \max_{i=1}^{n}\left[t_f(T_i) + c(e_{T_iT_j})\right] \quad (7)$$

n : no. of dependenci es

form (7),

$$t_s(T_j) = t_f(T_i) + c(e_{T_iT_j}) \quad (8)$$

form (2) and (8),

$$t_f(T_j) = w(T_j) + t_f(T_i) + c(e_{T_iT_j}) \quad (9)$$

case (2.4) : if T_j depends on more than one task (i.e. T_j depends on more than T_i) and T_j waits until T_k completes

form (7),

$$t_s(T_j) = \begin{cases} t_f(T_k) & \text{if } c(e_{T_iT_j}) < t_f(T_k) - t_f(T_i) \\ t_f(T_i) + c(e_{T_iT_j}) & \text{othewise} \end{cases} \quad (10)$$

form (2) and (10),

$$t_f(T_j) = t_s(T_j) + w(T_j) \quad (11)$$

---

* $c(e_{T_iT_j}) = \begin{cases} c_{ij} & \text{if task } T_i \text{ and } T_k \text{ are assigned to different processor} \\ 0 & \text{otherwise} \end{cases}$

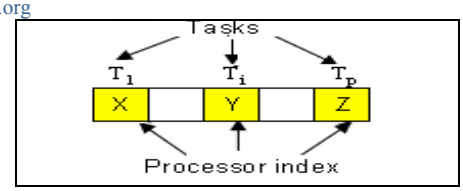

Fig.5. chromosome that encodes processor list

5.1.3. chromosome construction using TLPLC

In this case, the chromosome has a combination of tasks and processors as show in Fig.6.

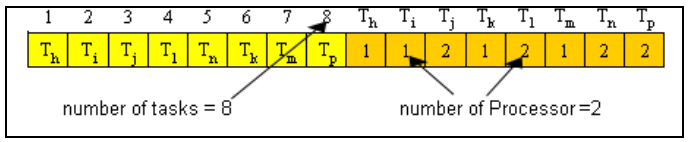

Fig.6. Chromosome that encodes task list and processor list

5.2. Fitness function

The Fitness function (FF) is essentially the objective function for the problem. It is used to evaluate the solution, and it also controls the selection process. For the multiprocessor scheduling problem we can consider factors, such as throughput, finishing time and processor utilization for the FF.

$$t_f(P) = \max_{T \in V} \{t_f(T)\}. \qquad (12)$$

$t_f(p)$: is the finishing time of processor P, the time at which the last task scheduled on P terminates.

Genetic algorithm works naturally on the maximization problem, while the mentioned objective function (finishing time of schedule) has to be minimized. Therefore, it is necessary to convert the objective function into maximization form called fitness function. Here, the calculation of the FF. which is determined by the following equation:

$$FF = ft_{max} - \max_{T \in V} \{t_f(T)\} \qquad (13)$$

where, ft_{max} is the maxmium finishing time observed in the current population

6. Genetic operators

The selection operator should be applied before using the crossover and mutation operators.

6.1. Selection operator

This selection operator allocates the reproductive trials to chromosomes according to their fitness. Different approaches were used in the selection operators such as roulette wheel selection and tournament selection. The tournament selection was found to be the better one [38].

The purpose of the selection is to emphasize fitter individuals in the population in hopes that their offspring's have higher fitness. Chromosomes are selected from the initial population to be parents for reproduction.

In this paper elitism is used to eliminate the chance of any undesired loss of information during the selection process. It selects the best two chromosomes and copies them into the mating pool, meanwhile in the next generation. Because such chromosomes might be lost if not selected for reproduction and also they may be destroyed by the crossover or the mutation process. This issue significantly improves the GA's performance.

Tournament selection randomly picks a tournament size (Ts) of chromosomes from the tournament which is a copy of the population (pop). The best chromosome from (Ts) that has the highest fitness (fit) is the winner. It is then inserted into the mating pool (which is for example half of the tournament). The tournament competition is repeated until the mating pool for generating new offspring is filled. After that, crossover and mutation are performed. The developed tournament method is as shown in Fig.7.

```
Tournament Selection Method
tournamentselection (pop, fit, Ts);
BEGIN
1. Compute size of mating pool as size of population/2;
2. Compute the best two individuals from population
3. Add them at mating pool m & at new population
4. for j←1 to Ts
     5. DO compute random point as any point between 1 and
        population size
     6. T[j] ← pop [point];
     7. TF[j] ← fit [point];
8. ENDFOR
9. Compute the best one from T according to fitness
10. Add it to the mating pool
11. Repeat steps 4 to 10 until mating pool is full
END
```
Fig.7. Tournament Selection Method

6.2. Crossover operator

The crossover operator is a reproduction operator which implements the principles of evolution. It creates new chromosomes (children or offspring) by combining two randomly selected parent chromosomes. These newly created chromosomes inherit the genetic material of their ancestors. Chromosomes in the mating pool are subjected to crossover with probability p_c. Two chromosomes are selected from the mating pool, and a random number RN $\in [0, 1]$ is generated. If RN $< p_c$, these chromosomes are subjected to the crossover operation using single point crossover operator. Otherwise, these chromosomes are not changed.

6.2.1. Crossover of task list.

The chromosome encoding of a task list states that each task T $\in$ V, can appear only once in the chromosome. To understand this, see Fig.8, the child (offspring) ch1 and ch2 will have tasks T5 and T4 twice, respectively.

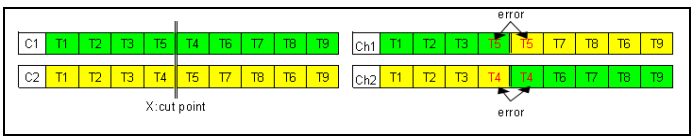

Fig.8. Single-point crossover for task list chromosomes and error

The problem is overcome with the following single-point crossover operator. Given two randomly chosen chromosomes c1 and c2, a cut point x, $1 \le x < \mathbf{V}$, is selected randomly. The genes [1, x] of c1 and c2 are copied to the genes [1, x] of the new children ch1 and ch2, respectively. To fill the remaining genes [x + 1, **V**] of ch1 (ch2), chromosome c2 (c1) is scanned from the first to the last gene and each task that is not yet in ch1 (ch2) is added to the next empty position of ch1 (ch2) in the order that it is discovered. Fig.9, illustrates the procedure of this operator. Under the condition that the task lists of chromosomes c1 and c2 are in precedence order, this operator even guarantees that the task lists of ch1 and ch2 also are. It is easy to see this for the genes [1, x] of both ch1 and ch2 as they are only copied from c1 and c2. The remaining genes of ch1 and ch2 are filled in the same relative order in which they appear in c2 and c1, respectively. Hence, among themselves, these remaining tasks must also be in precedence order. Furthermore, there cannot be a precedence conflict between the tasks on the left side of the crossover point with those on the right side of the crossover point, this separation of the tasks into two groups has not changed from c1 to ch1 neither from c2 to ch2 and it adheres to the precedence constraints in c1 and c2.

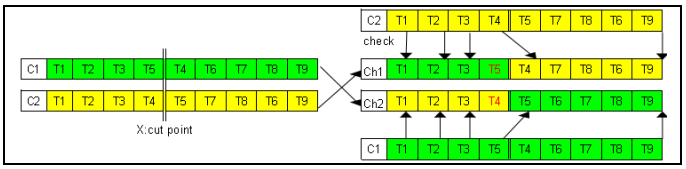

Fig.9. Single-point crossover for task list chromosomes after error

6.2.2. Crossover of Processor List

For the chromosome encoding of the processor list, quite simple crossover operators can be employed. The processor list chromosome in each gene can assume the same range of values (1 to P). Furthermore, the value of one gene has no impact on the possible values of the other genes. Fig.10, illustrates how the single point crossover operator works. Note that the generated new children are always valid chromosomes.

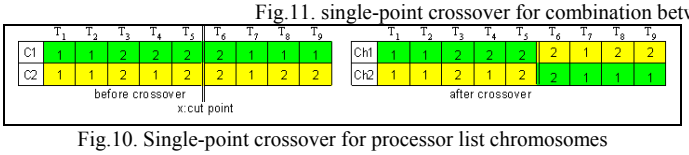

Fig.10. Single-point crossover for processor list chromosomes

6.2.3. Crossover of TLPLC

Crossover of TLPLC combines the task list and the processor list in one chromosome. Since these parts differ strongly, the simple solution for the operator is to apply the two previously described operators separately to each part. If $0.35 \le pc \le 0.55$ we apply on the first part. If $0.55 < pc \le 0.75$ we apply on the second part and if $0.75 < pc \le 0.95$ we apply on the two parts as shown in Fig.11.

6.3. Mutation operator

The mutation operator (p_m) has a much lower probability than the crossover operator. Its main function is to safeguard avoiding the convergence of the state search to a locally best solution. A swapping mutation operator is very suitable for chromosomes in TL as in Fig. 12 and PL as in Fig. 13, where we swap the two genes that are randomly selected. Another alternative for PL is to change the values of the genes that were randomly picked shown in Fig.14.

The mutation in TLPLC is based on the same methods used in TL and PL.

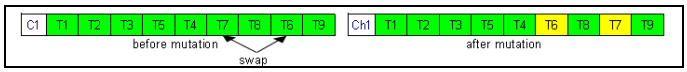

Fig.12. Swapping mutation operator for task list chromosome

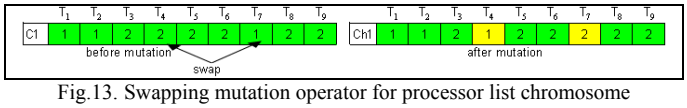

Fig.13. Swapping mutation operator for processor list chromosome

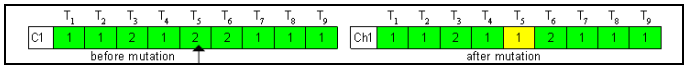

Fig.14. Mutation operator for processor list chromosome

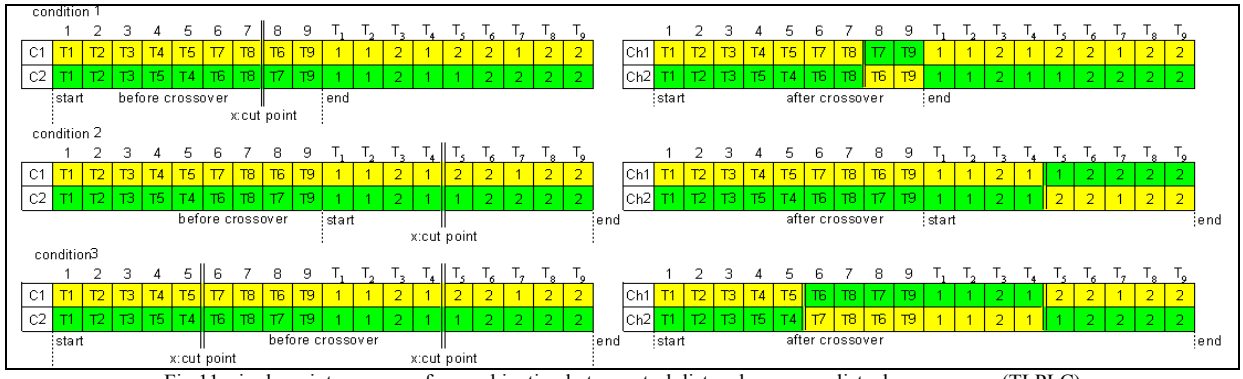

Fig.11. single-point crossover for combination between task list and processor list chromosomes (TLPLC)

7. Experimental results

In this section, intensive simulated experiments on random and real applications have been conducted. The genetic

algorithms used the following parameters throughout the simulations:

• Population size = 20.
• Maximum number of generation = 1000
• Crossover probability (p_c) = 0.7
• Mutation probability (p_m) = 0.3
• Number of generation without improvement (with same fitness) = 200.

7.1. simulated experiments based on MLSH

In this section, MLSH is compared with the most related heuristics, such as modified critical path (MCP) [6], dominant sequence clustering (DSC) [4], mobility directed (MD) [6] and dynamic critical path (DCP) [20]. Table 1 demonstrates the makespan of randomly generated task graph, problem 1 shown in Fig. 15. The obtained performance shown in Fig. 16 shows that processor efficiency with MLSH outperforms all other algorithms.

In experimental problem 1:

- Best solution: the best result from the 15 times iterations.
- Processor efficiency (%) = Sequential Time /total processing time
- Sequential Time: is actually task execution time on uniprocessor.
- Processing time: is (number of used processors × makespan).

For example: Efficiency for MLSH = (30 / 2*23) =65.2%

Efficiency for DCP and MD = (30 / 2*32) =46.9%

Efficiency for DSC = (30 / 4*27) =27.8%

Efficiency for MCP = (30 / 3*29) =34.48%

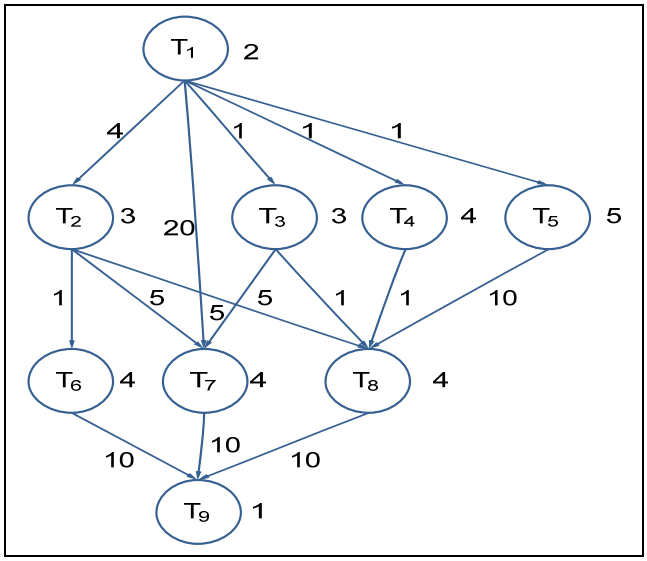

Fig.15. Randomly generated task graph

Table 2: Comparative results based on problem 1

Algorithms	TL	PL	TLPLC
No.of processors	2	2	2
Best solution	22	21	21

Table 1: Comparative results based on problem1

Algorithms	MCP	DSC	MD	DCP	MLSH
No.of processors	3	4	2	2	2
Best solution	29	27	32	32	23

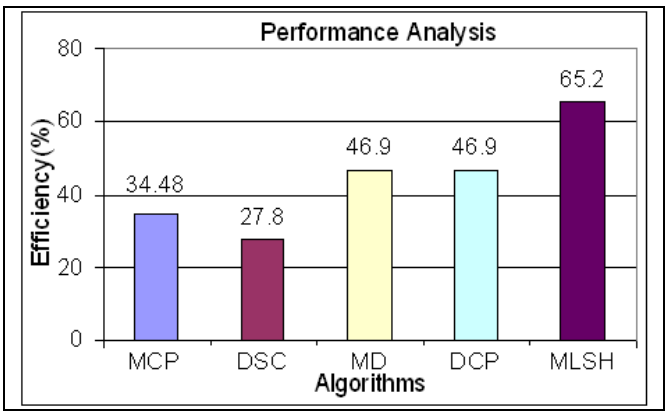

Fig.16. performance analysis of MCP, DSC, MD, DCP, and MLSH

7.2. Comparison between TL, PL, and TLPLC

In this section, the TLPLC is compared with the TL and the PL for two problems. Problem 1 mentioned above and problem 2, Gaussian elimination method graphs shown in Fig. 17. Tables 2 and 3 demonstrate the best solution (the makespan) for 15 iterations of problems 1 and problem 2 respectivly. The achieved results shown in Fig.18 and Fig.19 prove that the efficiency of TLPLC is better than TL and PL.

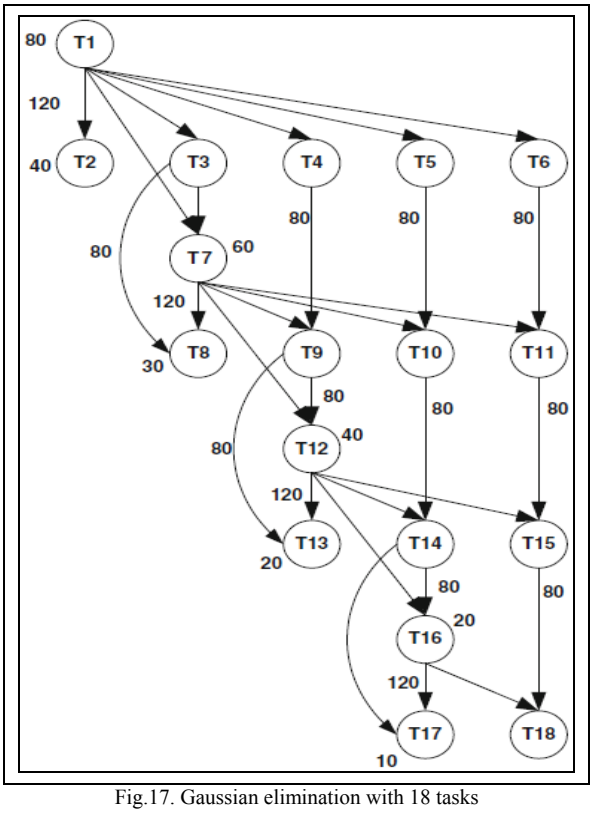

Fig.17. Gaussian elimination with 18 tasks

Table 3: Comparative results based on problem 2

Algorithms	TL			PL			TLPLC		
No. of processors	2	3	4	2	3	4	2	3	4
Best solution	460	440	450	510	490	490	440	440	440

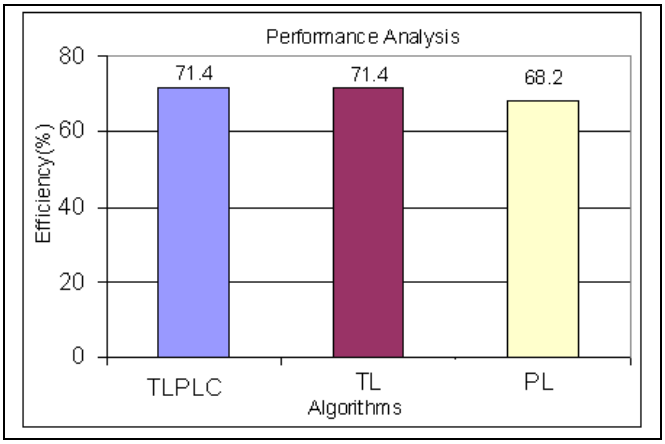

Fig.18. Performance analysis of TLPLC, TL, and PL based on problem 1

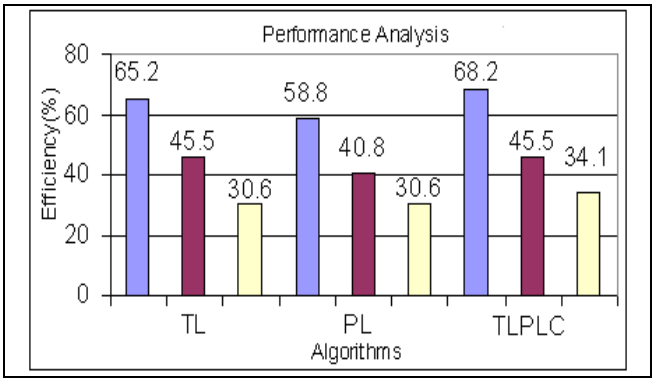

Fig.19. Performance analysis of TLPLC, TL, and PL based on problem 2

Based on the achieved results:-

1- **Optimum solution**

- TL method reached the optimum solution several times with the same chromosome, and sometimes, it did not reach it.
- PL method did not reach the optimum solution for these two problems, but it did in some other problems.
- TLPLC method did reach the optimum solution many times with different chromosomes (TLPLC has many optimum solutions).

2- Number of Iterations

- TL required small number of iterations to reach the optimum solution however, each iteration consumes high computational time.
- PL required much iteration.
- TLPLC required small number of iterations with less computational time.

3- **Crossover and Mutation operation**

- TL applies the crossover operation on tasks that might cause task duplication in the same chromosome. Mutation operation when applied on tasks, it might cause conflict with their precedence that requires more processing time to get rid of it.
- PL applies the crossover operation on processors which is easy to be implemented. The mutation is also simple and can be done using two different methods:
  1. Swapping two tasks between any two processors
  2. Migration, assigning a processor's task to any other processor.
- TLPLC applies the crossover operation on the chromosome using the methods used in PL and TL. The mutation is applied in the same way as in PL and TL.

So we can conclude that TLPLC based GA (TLPLC-GA) algorithm is better than PL and TL based GA and hence it will be used in the rest of the paper when comparing some heuristics and genetic algorithms.

7.3. Comparison between TLPLC-GA and some well known heuristics

In this section, the proposed algorithm, TLPLC-GA, is compared with MCP, DSC, MD, and DCP. Table 4 and 5 demonstrate the makespan of problem 1 and problem 2, respectively. The results of TLPLC-GA in all cases are better than the compared algorithms. TLPLC-GA was run 15 times in each case (using 2, 3 and 4 processors) and the best makespan of each case has been reported in Table 4 and 5. The obtained results shown in Fig.20 for problem 1 and Fig.21 for problem 2 prove that the performance of TLPLC-GA surpasses the other algorithms.

Table 4: Comparative results based on problem 1

Algorithms	MCP	DSC	MD	DCP	TLPLC-GA		
No. of processors	3	4	2	2	2	3	4
Best solution	29	27	32	32	21	21	21

Table 5: Comparative results based on problem 2

Algorithms	MCP	DSC	MD	DCP	TLPLC-GA		
No. of processors	4	6	3	3	2	3	4
Best solution	520	460	460	440	440	440	440

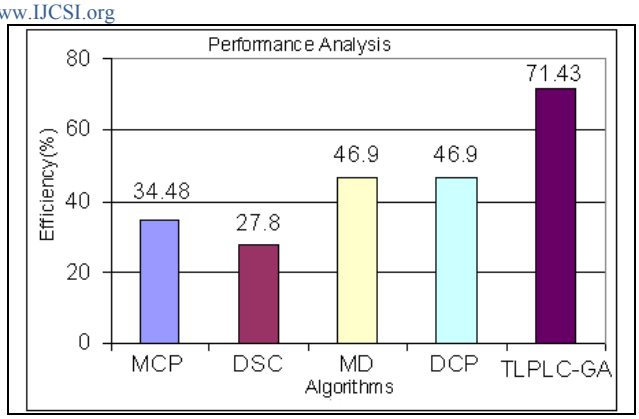

Fig.20 Performance analysis of MCP, DSC, MD, DCP, and TLPLC-GA for problem 1

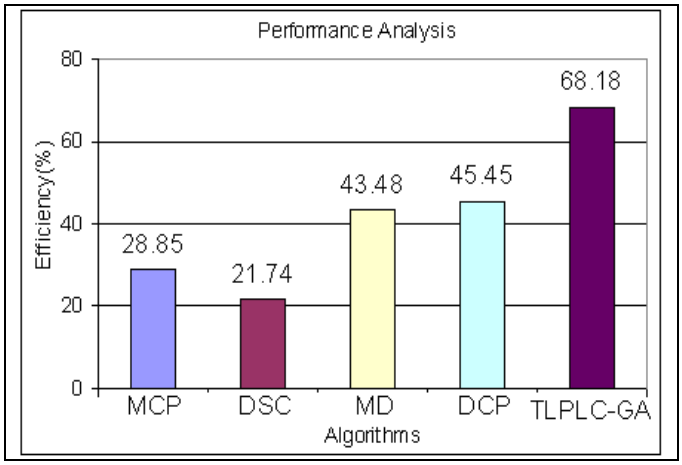

Fig.21 Performance analysis of MCP, DSC, MD, DCP, and TLPLC-GA for problem 2

7.4. Comparisons with GA Based Algorithms

Here we compare the proposed algorithm, TLPLC-GA, with other GA-based methods, BGA and PMC and two test benchmarks were employed.

Test bench 1:

First, TLPLC-GA, BGA and PMC are applied on problem 1. Table 6 and Fig. 22 show that the obtained results in terms of average makespam and processor efficiency for TLPLC-GA and BGA in all cases are the same, and they are both better than PMC.

Table 6: Comparative results based on problem 1

Algorithms	TLPLC-GA			BGA			PMC		
No. of processors	2	3	4	2	3	4	2	3	4
Average makespam	21	21	21	21	21	21	21.9	22.4	22.3

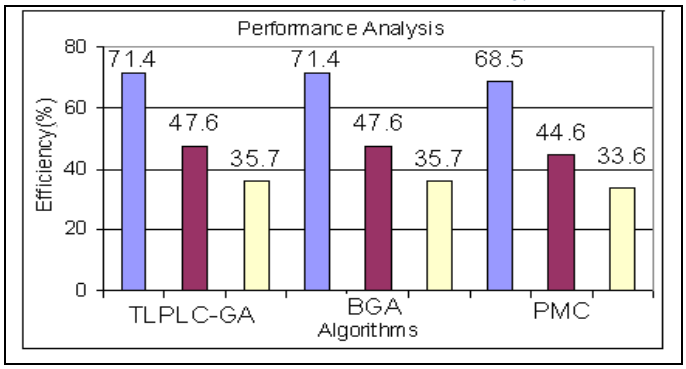

Fig.22 Performance analysis of TLPLC-GA, BGA and PMC for problem 1

Second, the three algorithms are applied on the problem 2. Table 7 shows that TLPLC-GA has better makespan compared to BGA and PMC in terms of average solutions in all cases.Fig.23 shows that Processor efficiency for the proposed algorithm, TLPLC-GA is quite better than that of BGA and PMC.

Table 7: Comparative results for Gaussian elimination on problem 2

Algorithms	TLPLC-GA			BGA			PMC		
No. of processors	2	3	4	2	3	4	2	3	4
Average makespam	446	443	451	463	461	461	491	522	544

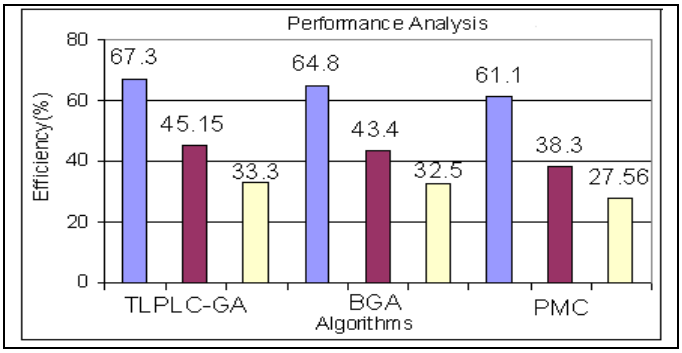

Fig.23 Performance analysis of TLPLC-GA, BGA and PMC for problem 2

Test bench2:

Table 8 summarizes different problems that have been addressed in this paper for the sake of comparison among TLPLC-GA, BGA and PMC.

Table 8. Selected Test Problems

Problem	# tasks	Comm. costs	Description
P1[17]	15	25 (fixed)	Gauss-Jordan algorithm
P2[17]	15	100 (fixed)	Gauss-Jordan algorithm
P3[17]	14	20 (fixed)	LU decomposition
P4[17]	14	80 (fixed)	LU decomposition
P5[41]	17	Variable for each graph edge	Random
P6[6]	18	Variable for each graph edge	Gaussian elimination
P7[6]	16	40 (fixed)	Laplace equation solver
P8[6]	16	160 (fixed)	Laplace equation solver

The three algorithms applied on the test benches are presented in Table 8.

According to results in Table 9, TLPLC-GA and BGA showed the same average makespan for problems 1, 2, 3, and 6. In some cases, the TLPLC-GA achieved better average makespan than BGA as in problems 4, 5, 7, and 8. The TLPLC-GA showed better makespan than PMC in all problems. Fig. 24 shows that the efficiency of TLPLC-GA is better than BGA and PMC.

Table 9 The average makespan for problems in Table 8

Problem	TLPLC-GA Average Makespan	BGA Average Makespan	PMC Average Makespan
P1	***300***	***300***	***300***
P 2	***440***	***440***	472
P3	***270***	***270***	290
P 4	***360***	365	418
P 5	***37***	37.2	38.4
P 6	***390***	***390***	424
P 7	***760***	790	810
P8	***1070***	1088	1232

Note: The best results in each row are shown by Bold-Italic-Underline font.

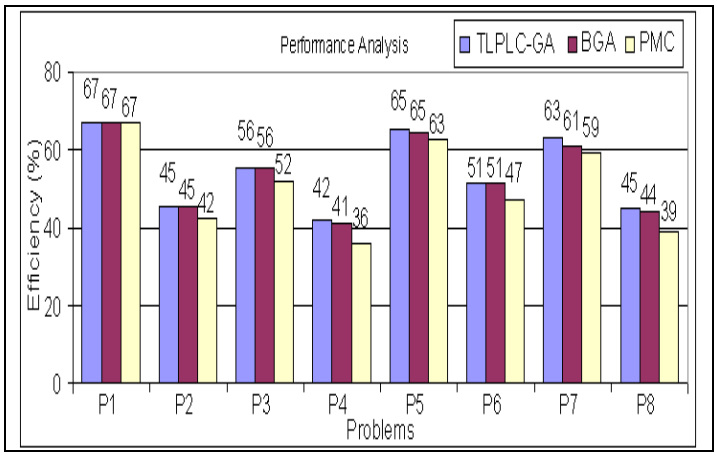

Fig.24 Performance analysis of TLPLC-GA, BGA and PMC for table 8

8. Conclusions.

This paper presents two new approaches for task scheduling in multiprocessor systems: Modified list scheduling heuristic (MLSH) and hybrid approach composed of Genetic Algorithm and MLSH. Furthermore, three different types of chromosomes For Genetic algorithm: task list (TL), processor list (PL) and combination of both (TLPLC) have been presented. Simulated results show that TLPLC representation for GA is better than TL and PL for GA. Comparisons of the proposed algorithm, TLPLC-GA, with the most related algorithms based on GA and heuristic algorithms in terms of best makespan, average makespan, and processor efficiency have been conducted. The experimental results showed that the hybrid approach (TLPLC-GA) outperforms the other algorithms.

References

[1]. M. R. Bonyadi and M. E. Moghaddam, "A bipartite genetic algorithm for multi-processor task scheduling", International Journal of Parallel Programming, Vol. 37, No. 5, 2009, pp. 462-487.

[2]. R. Hwang, M. Gen and H. Katayama, "A comparison of multiprocessor task scheduling algorithms with communication costs", Computers and Operations Research, Vol. 35, No. 3, 2008, pp. 976–993.

[3]. H. El-Rewini, T. G. Lewis and H. H. Ali, "Task Scheduling in Parallel and Distributed Systems", Prentice-Hall International Editions, 1994.

[4]. T. Yang and A. Gerasoulis , "DSC: scheduling parallel tasks on an unbounded number of processors", IEEE Transactions on Parallel and Distributed Systems, Vol. 5, No. 9, 1994, pp. 951–967.

[5]. T.L. Adam, K.M. Chandy and J.R. Dicksoni, "A comparison of list schedules for parallel processing systems", Communications of the ACM, Vol. 17, No. 12, 1974, pp. 685–690.

[6]. M.Y. Wu and D.D. Gajski, "Hypertool: A Programming Aid for Message-Passing Systems," IEEE Trans. Parallel and Distributed Systems, vol. 1, No. 3, 1990, pp. 330-343.

[7]. R.C. Correa, A. Ferreira and P. Rebreyend, "Scheduling multiprocessor tasks with genetic algorithms", IEEE Transactions on Parallel and Distributed Systems, Vol. 10, No. 8, 1999, pp. 825–837.

[8]. T. Thanalapati and S. Dandamudi, "An efficient adaptive scheduling scheme for distributed memory multicomputer", IEEE Transactions on Parallel and Distributed Systems, Vol. 12, No. 7, 2001, pp. 758–768.

[9]. N. Nissanke, A. Leulseged and S. Chillara, "Probabilistic performance analysis in multiprocessor scheduling", Journal of Computing and Control Engineering, Vol. 13, No. 4, 2002, pp. 171–179.

[10]. J. Corbalan, X. Martorell and J. Labarta, "Performance-driven processor allocation", IEEE Transactions on Parallel and Distributed Systems, Vol. 16, No. 7, 2005, pp. 599–611.

[11]. A.S. Wu, H. Yu, S. Jin, K.-C. Lin, and G. Schiavone, "An incremental genetic algorithm approach to multiprocessor scheduling", IEEE Transactions on Parallel and Distributed Systems, Vol. 15, No. 9, 2004, pp. 824–834.

[12]. Y. Kwok and I. Ahmad, "Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors," ACM Computing Surveys, vol. 31, no. 4, 1999, pp. 406-471.

[13]. E.S.H. Hou, N. Ansari and R. Hong, "A Genetic Algorithm for Multiprocessor Scheduling", IEEE Transactions on Parallel and Distributed Systems. Vol. 5, No. 2, 1994, pp. 113 – 120.

[14]. R.K. Hwang and M. Gen, "Multiprocessor scheduling using genetic algorithm with priority-based coding", Proceedings of IEEJ conference on electronics, information and systems, 2004.

[15]. Y.H. Lee and C. Chen, "A Modified genetic algorithm for task scheduling in multi processor systems", The Ninth Workshop on Compiler Techniques for High Performance Computing 2003.

[16]. F. Montazeri, M.S. Jelodar, S.N. Fakhraie and S.M. Fakhraie, "Evolutionary multiprocessor task scheduling", Proceedings of the International Symposium on Parallel Computing in Electrical Engineering (PARELEC'06) 2006.

[17]. T. Tsuchiya, T. Osada and T. Kikuno, "Genetic-Based Multiprocessor Scheduling Using Task Duplication," Microprocessors and Microsystems, vol. 22, No. 3-4, 1998, pp. 197-207.

[18]. J.J. Hwang, Y.C. Chow, F.D. Anger and C.Y. Lee, "Scheduling precedence graphs in systems with inter-processor communication times", SIAM Journal on Computing vol. 8, No. 2, 1989, pp. 244-258.

[19]. H. Kasahara and S. Narita, "Practical multiprocessing scheduling algorithms for efficient parallel processing", IEEE Transactions on Computers, Vol. 33, No. 11, 1984, pp. 1023–1029.

[20]. Y.K. Kwok and I. Ahmad, "dynamic critical path scheduling: an effective technique for allocating task graphs to multiprocessors", IEEE Transactions on Parallel and Distributed Systems, Vol. 7, No. 5, 1996, pp. 506–521.
[21]. R.C. Corrga, A. Ferreira and P. Rebreyend, "scheduling multiprocessor tasks with genetic algorithm", IEEE Transactions on Parallel and Distributed Systems, Vol. 10, No. 8, 1999, pp. 825–837.
[22]. K. Kaur, A. Chhabra and G. Singh, "Modified Genetic Algorithm for Task Scheduling in Homogeneous Parallel System Using Heuristics", International Journal of Soft Computing, Vol. 5, No. 2, 2010, pp. 42-51.
[23]. I. Ahmad and M.K. Dhodhi, "Multiprocessor scheduling in a genetic paradigm", Parallel Computing, Vol. 22, No. 3, pp. 395 – 406, 1996.
[24]. M.R. Bonyadi and M. R. Azghadi, S. Hashemi and M. E. Moghadam, "A hybrid multiprocessor task scheduling method based on immune genetic algorithm" Qshine Workshop on Artificial Intelligence in Grid Computing 2008.
[25]. Y.K. Kwok and I. Ahmad, "Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors," ACM Computing Surveys, vol. 31, no. 4, 1999, pp. 406-471.
[26]. Dr.G.Padmavathi, Mrs.S.R.Vijayalakshmi, "A Performance Study of GA and LSH in Multiprocessor Job Scheduling", International Journal of Computer Science Issues, IJCSI, Vol. 7, No. 1, 2010, pp. 37-42.
[27]. B. Kruatrachue and T.G. Lewis, "Duplication Scheduling Heuristic, a New Precedence Task Scheduler for Parallel Systems", Technical Report 87-60-3, Oregon State University, 1987.
[28]. B.S. Macey and A.Y. Zomaya, "A Performance Evaluation of CP List Scheduling Heuristics for Communication Intensive Task Graphs," Proc. Joint 12th Int'l Parallel Processing Symposium and Ninth Symposium. Parallel and Distributed Processing. , 1998, pp. 538-541.
[29]. T.C. Hu, "Parallel Sequencing and Assembly Line Problems,"Operations Research, Vol. 19, No. 6, 1961, pp. 841-848.
[30]. A. Zomaya, C. Ward and B. Macey, "Genetic scheduling for parallel processor systems comparative studies and performance issues", IEEE Transactions on Parallel and Distributed Systems, Vol. 10, No. 8, 1999, pp. 795 – 812.
[31]. M. Moore, "An accurate parallel genetic algorithm to schedule tasks on a cluster", Parallel and Distributed Systems, Vol. 30, No. 5-6, 2004, pp. 567–583.
[32]. W. Yao, J. You and B. Li, "Main sequences genetic scheduling for multiprocessor systems using task duplication", Microprocessors and Microsystems, Vol. 28, No 2, 2004, pp. 85–94.
[33]. Y.K. Kwok and I. Ahmad, "Efficient scheduling of arbitrary task graphs to multiprocessors using a parallel genetic algorithm", Parallel Distributed Computing, Vol. 47, No.1, 1997, pp. 58–77.
[34]. O. Ceyda and M. Ercan, "A genetic algorithm for multilayer multiprocessor task scheduling. In:TENCON 2004. IEEE region 10 conference, Vol. 2, 2004, pp. 68-170.
[35]. S. Cheng and Y. Huang, "Scheduling multi-processor tasks with resource and timing constraintsusing genetic algorithm", IEEE international symposium on computational intelligence inrobotics and automation, Vol. 2, 2003, pp 624–629.
[36]. Y.W. Zhong and J.G. Yang, "A genetic algorithm for tasks scheduling in parallel multiprocessor systems". In: Proceedings of the Second International Conference on Machine Learning and Cybernetics, 2003, pp. 1785–1790
[37]. K. Kaur, A. Chhabra and G. Singh, "Heuristics Based Genetic Algorithm for Scheduling Static Tasks in Homogeneous Parallel System", International Journal of Computer Science and Security, Vol. 4, No.2, 2010, pp. 149-264
[38]. F.A. Omara and M.M. Arafa, "Genetic algorithms for task scheduling problem", Journal of Parallel and Distributed Computing, Vol. 70, No.1, 2010, pp. 13–22.
[39]. S.N. Sivanandam and S.N. Deepa, "Introduction to Genetic Algorithms", Springer-Verlag Berlin Heidelberg, 2008.
[40]. E. S. H. Hou, R. Hong and N. Ansari, "Efficient Multiprocessor Scheduling Based On Genetic Algorithms", Industrial Electronics Society, IECON '90., 16th Annual Conference of IEEE , Vol. 2, 1990, pp. 1239 – 1243.
[41]. M.A. Al-Mouhamed, "Lower Bound on the Number of Processors and Time for Scheduling Precedence Graphs with Communication Costs," IEEE Transactions on Software Engineering., Vol. 16, No. 12, pp. 1390- 1401, 1990.

Mostafa R. Mohamed obtained his B.Sc. degree of Electronics and Communications engineering from Faculty of Engineering, Fayoum University in 2006. Eng. Mostafa interests include parallel and distributed systems, parallel processing, grid computing, distributed operating systems, multi processor scheduling and genetic algorithms.

Medhat H. A. Awadalla obtained his B.Sc. degree from Helwan University in 1991 in the Electronics and Communications Department and his M.Sc in the field of reconfigurable computer architecture in 1996 from Helwan University. He received his PhD from Cardiff University, UK in the field of mobile robots in 2005. He was a postdoctoral fellow at Cardiff University in 2006 and currently he is working as an Assistant Professor in Helwan University. His research interests include real time systems, multi processor scheduling, parallel and distributed systems, grid computing and sensor networks.