Min–min GA Based Task Scheduling In Multiprocessor Systems

Medhat Awadall, Afaq Ahmad, Samir Al-Busaidi

Abstract — An efficient assignment and scheduling of tasks of a multiprocessor system is one of the key elements in the effective utilization of multiprocessor systems. This problem is extremely hard to solve, consequently several methods have been developed to optimally tackle it which is called NP-hard problem. This paper presents two new approaches, Modified List Scheduling Heuristic (MLSH) and enhanced genetic algorithm by constructing promising chromosomes. Furthermore, this paper proposes three different representations for the chromosomes of the genetic algorithm: Min-min task list, processor list and combination of both. Extensive 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 performed with the most related algorithms, LSHs, Bipartite GA (BGA) and Priority based Multi-Chromosome (PMC). The achieved results show that the proposed approaches significantly surpass the other approaches in terms of task makespan and processor efficiency.

Index Terms— Multiprocessors, Task scheduling, Genetic algorithm, Makespan, Parallel and distributed system, List Scheduling Heuristic

I. INTRODUCTION

The proliferation in the use of multiprocessor systems these days in a great variety of applications is the result of many breakthroughs over the last two decades. These developments of multiprocessor systems are being used for several applications, including fluid flow, weather modeling, database systems, real-time, and image processing. The data for these applications can be distributed evenly on the processors of multiprocessor systems and maximum benefits from these systems can be obtained by employing an efficient task assignment and scheduling strategy [1]. 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 [2]. The goal of the scheduler is to assign tasks to available processors such that precedence requirements among tasks are satisfied and the overall length of time required to execute the entire program, the schedule length or makespan, is minimized.

Manuscript received December, 2013.

Medhat Awadalla, Department of Electrical and Computer Engineering, Sultan Qaboos University, PO Box 33, Zip Code 123, Oman.

Afaq Ahmad, Department of Electrical and Computer Engineering, Sultan Qaboos University, PO Box 33, Zip Code 123, Oman

Samir Al-Busaidi, Department of Electrical and Computer Engineering, Sultan Qaboos University, PO Box 33, Zip Code 123, Oman

Many heuristic approaches for task scheduling have been proposed [3]. 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 [4]. Another merit of a genetic search is that their inherent parallelisms that can be exploited so as to further reduce its running time. Thus, several methods have presented to solve this problem based on GAs [5-8].

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: Min-min task list, processor list, and a combination of both.

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.

II. MULTIPROCESSOR TASK SCHEDULING PROBLEM

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

1. The number of tasks and their precedence.
2. 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).
3. Number of processors and processors uniformity (A homogeneous multiprocessor system is composed of a set P = {P1… Pm} of 'm' identical processors.
4. 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 eij for each task Ti ∙ V that task Tj ∙ 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 eij, Ti is called the immediate predecessor of Tj and Tj is called the immediate successor of Ti. An immediate successor Tj depends on its immediate predecessors such that Tj 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 models, Gauss Jordan elimination, LU decomposition [9] and Laplace equation solver [10] task graphs are considered in this paper. Some of these task graphs are illustrated in Fig.1.

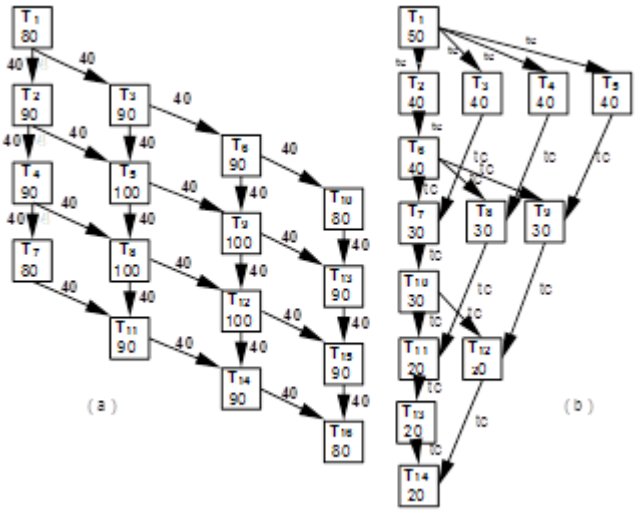

Fig.1. Description of task dependencies for (a) Laplace equation solver and (b) LU decomposition

III. RELATED WORK

Several approaches have been adopted to solve the multiprocessor task scheduling such as heuristic approaches, evolutionary approaches and hybrid methods [11]. Authors in [12] 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 [13]. Insertion Scheduling Heuristic (ISH) and Duplication Scheduling Heuristic (DSH) are well-known list scheduling heuristic methods [14]. ISH is a list scheduling heuristic that was developed to optimize scheduling DAGs with communication delays. ISH extends a basic list scheduling heuristic from Hu [15] by attempting to insert ready tasks into existing communication delay slots. DSH [16] 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 [17]. One of the main differences in these genetic approaches is their genetic operators, 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 a GA is the simplicity of the algorithm and complexity of the evolutionary optimization process. Authors in [18] 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. Even though the method proposed in [19] was very efficient, it does not search the entire 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. In [9], it is proposed some modifications to the approach in [20] to broaden the search space to include all the valid solutions. This modified approach was tested on task graphs that represent well-known parallel programs. Authors in [21] 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. In [22], the author applies parallel GA to the scheduling problem and compares its accuracy with mathematically predicted expected value. More GA approaches are found in [23]. Another new genetic-based multiprocessor scheduling method has been presented [24]. In that paper the author 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 have been performed 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 in [25]. In PGA, the input DAG is divided into partial graphs using a 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 [26], 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 [27]. Also it proposed a modified GA by using 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 a pure genetic algorithm and list approach. Unfortunately, the disadvantage is that the running time is larger than running the pure genetic algorithm. Therefore the aim of this paper is to reduce that time however the modification of pure GA is done by the chromosomes' representations. Also, we grasped many ideas for designing algorithms which reduces run time and memory requirement for their implementation [28] – [35].

IV. THE PROPOSED TASK MAPPING AND SCHEDULING ALGORITHM

List scheduling techniques assign a priority for 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 [24]. 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 where each task is characterized by the following parameters:

1. ts (T) : is the starting time of task T.
2. ts (T, P) : is the starting time of task T on processor P.
3. tf (T) : is the finishing time of task T.
4. w (T) : is the processing time of task T.

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

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

where, $\text{Pred}(Ti)$ is the set of predecessors of Ti.

This Level function indirectly conveys precedence relations among the tasks. If the task Ti is an ancestor of task Tj, then Level (Ti) < Level (Tj). 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 [15, 24]. Secondly, the sequence of tasks' execution in each level is determined. For the root level (For example, T1and T2 in Fig.3), if there is only one parent task, then it comes first.

If there are tasks have more than one parent, the number of the children for each parent in the next level is calculated and the parents will get priority according to that number in a descending order. The parent with the highest number of children comes first (for example T1 executed before T2).

If two or more parents have the same number of children (T3, T4 and T5) then the parent that has a common child is to be executed first (T4 and T5 executed before T3). When two parents have the same common child, they will be listed in an arbitrary order (T4 and T5).

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

V. 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.

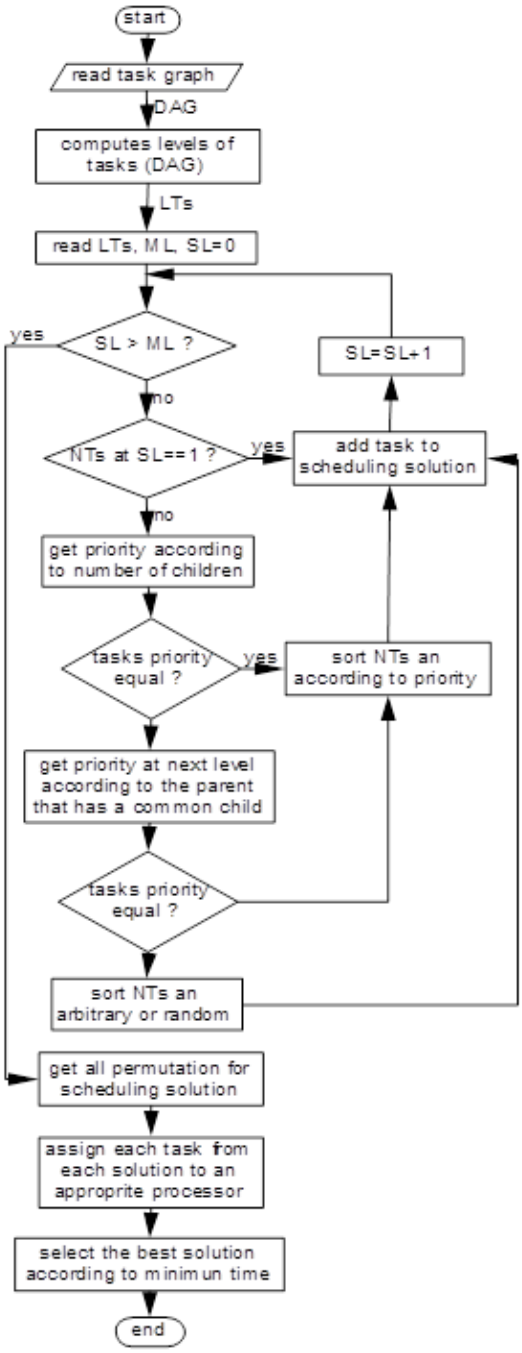

FIG.2. FLOWCHART MLSH ALGORITHM

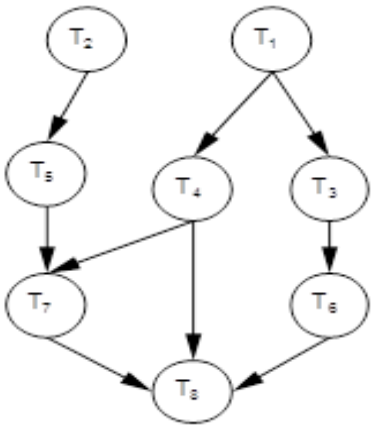

Fig.3. Example of DAG with 8 tasks

A. 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 (Min-min TL), Processor List (PL) and combination of them (TLPLC).

B. Chromosome construction using Min-min 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.

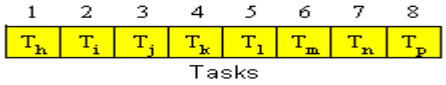

Fig.4. Chromosome that encodes Min-min TL

The Min-min [6] algorithm is originally designed for mapping tasks in heterogeneous computing systems and does not consider real-time tasks. It first finds the minimum completion time of all unmapped tasks, where the completion time of a task on a machine equal task's execution time on that machine plus execution times of all tasks mapped to that machine. Next, the task which has minimum completion time is selected, similar technique called Max-min selects the task with maximum completion time, and mapped to the machine. Finally, the newly mapped task is removed and the process is repeated until all tasks are mapped. The Min-min based GA approach, proposed in this paper, simply modifies the initialization step in GA procedure by incorporating a Min-min solution in the randomly generated population. This approach gives GA algorithm a push to start from a good solution and then GA goes on trying to optimize the solution resulting in the Min-min solution in the worst case.

C. 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 Ti Є 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.

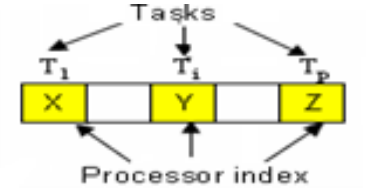

Fig.5. chromosome that encodes processor list

D. Chromosome construction using TLPLC

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

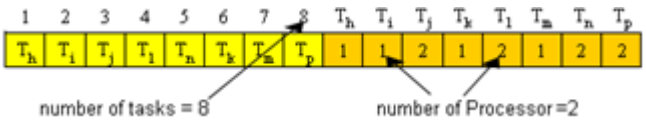

Fig.6. Chromosome that encodes processor list and Min-min task list.

E. 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.

$$t_f(P) = \max_{T \in V} \{ t_f(T) \}. \qquad (1)$$

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:

$$FF = ft_{max} - \max_{T \in V} \{ t_f(T) \} \qquad (2)$$

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

F. Genetic operators

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

G. 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 [18].

The purpose of the selection process 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. Such chromosomes might be lost if it is not selected for reproduction or destroyed by the crossover or mutation processes. This issue significantly improves the performance of GA.

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
Tournament selection (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

H. 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 pc. Two chromosomes are selected from the mating pool, and a random number RN ∈ [0, 1] is generated. If RN < pc, these chromosomes are subjected to the crossover operation using single point crossover operator. Otherwise, these chromosomes are not changed.

I. Crossover of Min-min task list

The chromosome encoding of a task list states that each task T ∈ 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.

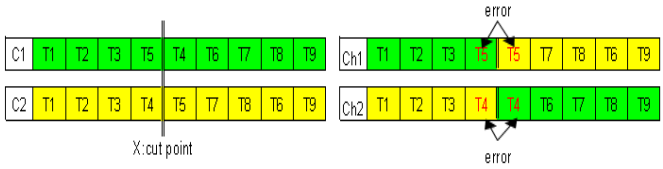

Fig.8. Single-point crossover for Min-min task list chromosomes with error

The problem is overcome with the following single-point crossover operator. Given two randomly chosen chromosomes c1 and c2, a cut point x, $1 \le x < V$, 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.

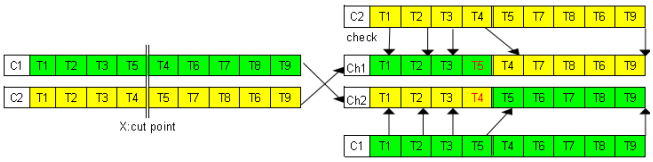

Fig.9. Right Single-point crossover for Min-min task list chromosomes

J. 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

K. 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 will apply on the first part.
If $0.55 < pc \le 0.75$, we will apply on the second part and
If $0.75 < pc \le 0.95$ we apply on the two parts.

L. Mutation operator

The mutation operator (pm) 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 Min-min TL as in Fig. 11 and PL as in Fig. 12, 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.13.
The mutation in TLPLC is based on the same methods used in Min-min TL and PL.

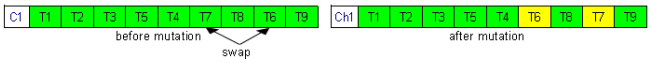

Fig.11. Swapping mutation operator for Min-min task list chromosome

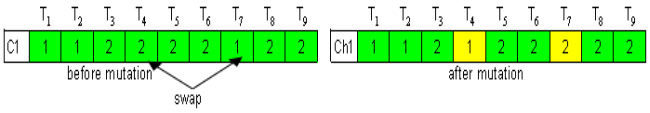

Fig.12. Swapping mutation operator for processor list chromosome

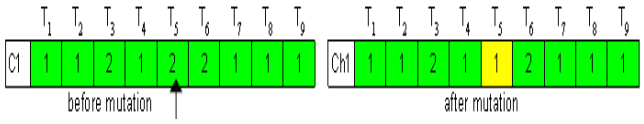

Fig.13. Mutation operator for processor list chromosome

VI. 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 (pc) = 0.7
- Mutation probability (pm) = 0.3
- Number of generation without improvement (with same fitness) = 200.

A. Simulated experiments based on MLSH

In this section, The MLSH is compared with some well-known heuristics, such as modified critical path (MCP) [6], dominant sequence clustering (DSC) [4], mobility directed (MD) [10] and dynamic critical path (DCP) [22]. Table 1 demonstrates the makespan of problem 1 that shown in Fig.14. It is a randomly generated task graph. The obtained performance shown in Fig. 15 shows that processor efficiency with MLSH outperforms all other algorithms.

In experimental problem 1:

1. Best solution: the best result from the 15 times iterations.
2. Processor efficiency (%) = Sequential Time /total processing time.
3. Sequential Time: is actually task execution time on uniprocessor.
4. 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%

Table 1: Comparative results in context of problem 1

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

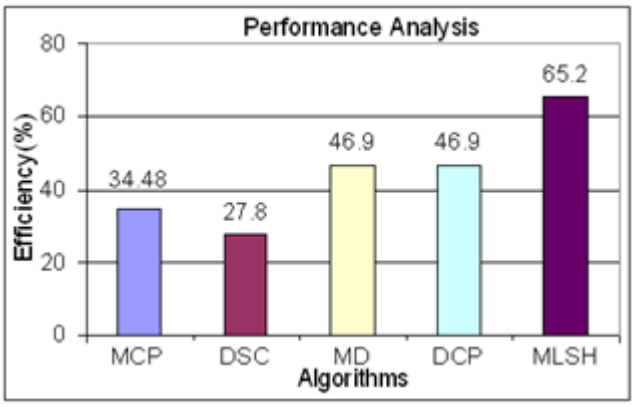

Fig.14. Performance analysis of MCP, DSC, MD, DCP, and MLSH

Table 2: Comparative results in the context of problem 1

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

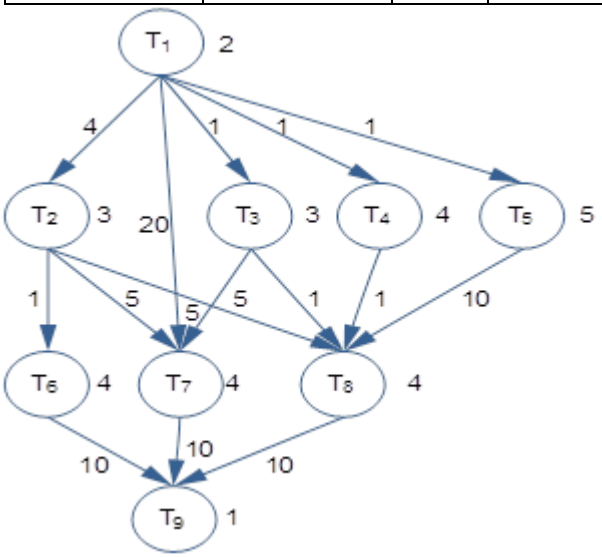

Fig.15. Example DAG with nine tasks

A. Comparison among Min-min TL, PL, and TLPLC

In this section, the TLPLC is compared with the Min-min TL and the PL in the context of two problems. Problem1 mentioned before and problem 2 shown in fig. 16 which is the Gaussian elimination method graphs. Tables 2 and 3 demonstrate the best solution (the makespan) for running of 15 iterations. The achieved results shown in Fig. 17 and Fig. 18 verified that the efficiency of TLPLC is better than min-min TL and PL.

Table 3: Comparative results in the context of Gaussian elimination with 18 tasks

Algorithms	Min-min 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

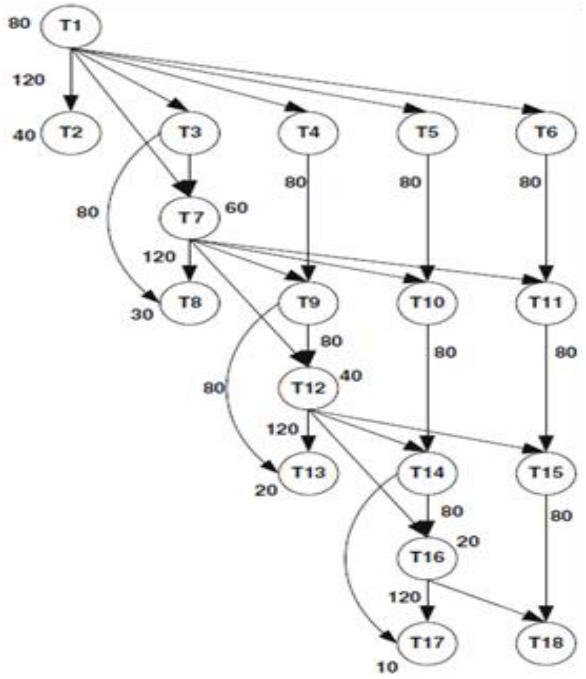

Fig.16. Performance analysis of TLPLC, TL, and PL for Fig.15

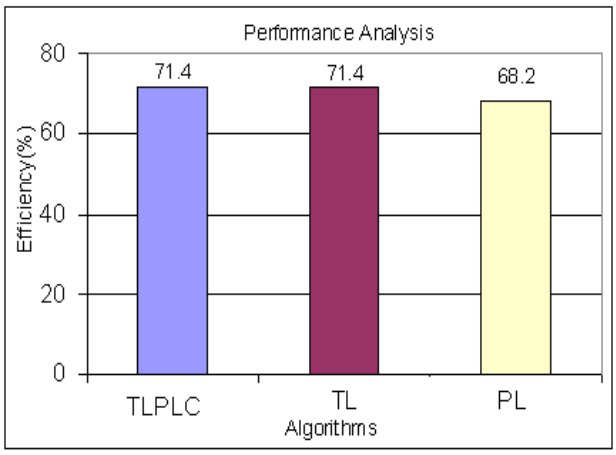

Fig.17. Gaussian elimination with 18 tasks

Based on the achieved results:-

1- Optimum solution

Min-min 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

Min-min TL required small number of iterations to reach the optimum solution however each iteration consumes high computational time.

PL required large number of iterations.

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

B. Comparison between TLPLC-GA and related 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 running 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. 19 for problem 1 and Fig. 20 for problem 2 verified that the performance of TLPLC-GA surpasses the other algorithms.

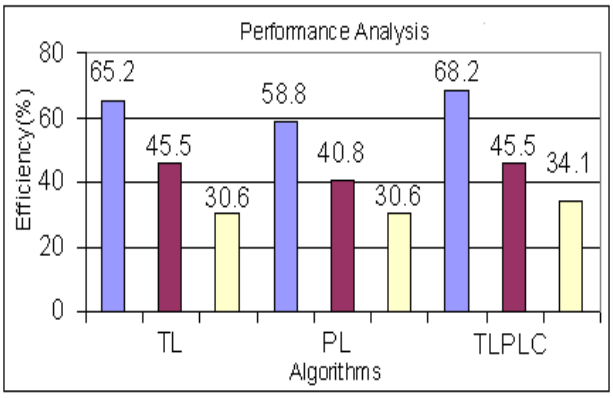

Fig.18. Performance analysis of TLPLC, Min-minTL, and PL

Table 5: Comparative results for 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	430	430	430

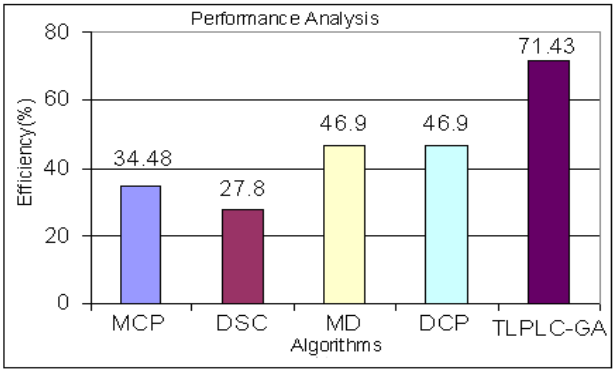

Fig.19 Performance analysis of MCP, DSC, MD, DCP, and TLPLC-GA for problem 1

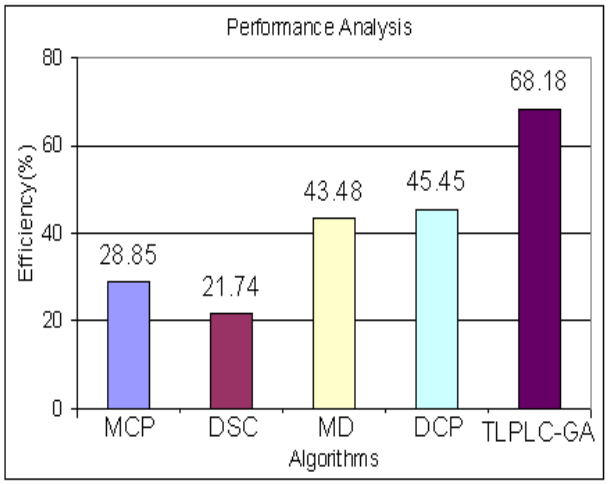

Fig.20 Performance analysis of MCP, DSC, MD, DCP, and TLPLC-GA for problem 2

C. 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:

At first, TLPLC-GA, BGA and PMC are applied on problem 1. Table 6 and Fig. 21 show that the obtained results in terms of average make spam and processor efficiency. 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 make spam	21	21	21	21	21	21	21.9	22.4	22.3

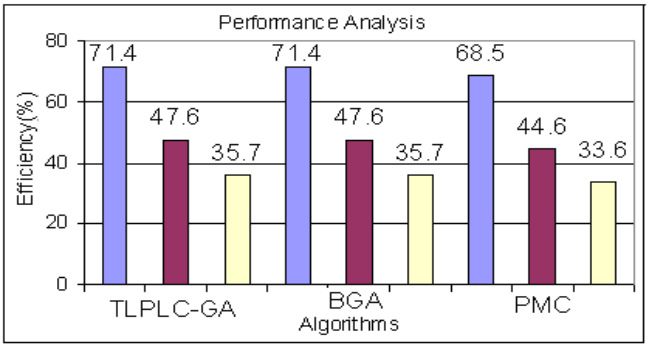

Fig.21 Performance analysis of TLPLC-GA, BGA and PMC for problem 1

Second, the three algorithms are applied on 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.22 shows that Processor efficiency for the proposed algorithm, TLPLC-GA is quite better than that of BGA and PMC.

Table 7: 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 make spam	446	443	451	463	461	461	491	522	544

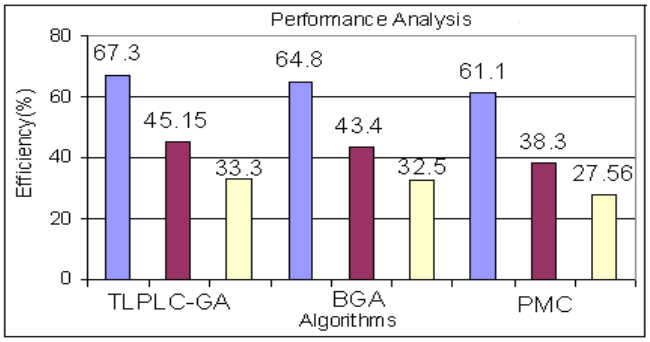

Fig.22 Performance analysis of TLPLC-GA, BGA and PMC Test bench2

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

Table8. 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 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. 23 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	BGA	PMC
	Average Makespan	Average Makespan	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.

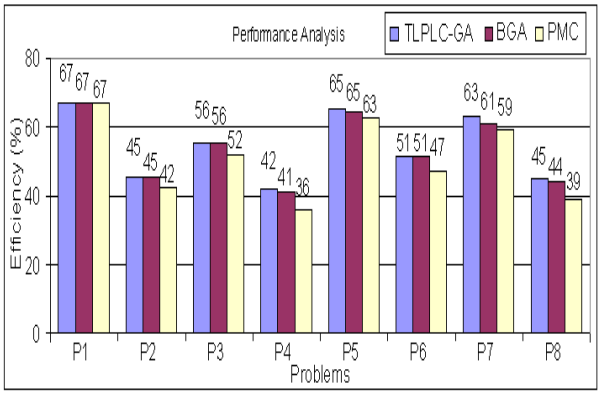

Fig.23 Performance analysis of TLPLC-GA, BGA and PMC for table 8

VII. CONCLUSION

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: Min-min task list (Min-min 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] Garshasbi, M. S. and Mehdi, E., "Tasks Scheduling on Parallel Heterogeneous Multi-processor Systems using Genetic Algorithm", International Journal of Computer Applications, Volume 61 - Number 9, 2013, pp. 23-27

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

[3] Ye Xu, Ling Wang, Shengyao Wang and Min Liu. "An effective immune algorithm based on novel dispatching rules for the flexible flow-shop scheduling problem with multiprocessor tasks". Int J Adv. Manufacturing , 2012

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

[5] Orhan Engin, Gülşad Ceran, Mustafa K. Yilmaz. "An efficient genetic algorithm for hybrid flow shop scheduling with multiprocessor task problems". Applied Soft Computing Volume 11, Issue 3, 2011, PP. 3056–3065.

[6] 4. Jasbir, Gurvinder, "Improved Task Scheduling on Parallel System using Genetic Algorithm", International Journal of Computer Applications (0975 – 8887) Volume 39– No. 17, February 2012

[7] Oncan T., " A Genetic Algorithm for the Order Batching Problem in Low-Level Picker-to-Part Warehouse Systems", Proceedings of The International MultiConference of Engineers and Computer Scientists 2013, pp. 19-24.

[8] Katanosaka, T., Sato H., Oyama S. and Kurihara M., "A Preference Search System Using an Interactive Genetic Algorithm", Proceedings of The International MultiConference of Engineers and Computer Scientists 2013 , pp. 66-69.

[9] Dai, M., Tang C., and Chuang C. (2013). "An Energy-Aware Workload Dispatching Simulator for Heterogeneous Clusters", Proceedings of the International MultiConference of Engineers and Computer Scientists 2013 Vol I, IMECS 2013, March 13 - 15, 2013, Hong Kong.

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

[11] R., Rahimi Azghadi,M., Hashemi, S., Ebrahimi Moghadam, M., "A hybrid multiprocessor task scheduling method based on immune genetic algorithm" Qshine 2008 Workshop on Artificial Intelligence in Grid Computing 2008.

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

[13] ReaKook H., Mitsuo G. and Hiroshi K., "A Performance Evaluation of Multiprocessor Scheduling with Genetic Algorithm". ReaKook Hwang et al./Asia Pacific Management Review (2006) 11(67-72).

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

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

[16] Kamaljit Kaur, Amit Chhabra and Gurvinder 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.

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

[18] G.Padmavathi and S.R.Vijayalakshmi, "Multiprocessor scheduling for tasks with priority using GA", International Journal of Computer Science Issues, IJCSI, Vol. 7, No. 1, 2010, pp. 37-42.

[19] Moore M., "An accurate parallel genetic algorithm to schedule tasks on a cluster", Parallel and Distributed Systems, Vol. 30, No. 5-6, 2004, pp. 567–583.

[20] Yao W, You J, Li B., "Main sequences genetic scheduling for multiprocessor systems using task duplication", Microprocessors and Microsystems, Vol. 28, No 2, 2004, pp. 85–94.

[21] Ceyda O, Ercan M "A genetic algorithm for multiprocessor task scheduling. In: TENCON 2004. IEEE region 10 conference, Vol. 2, 2004, pp. 68-170

[22] Cheng S, Huang Y "Scheduling multi-processor tasks with resource and timing constraints using genetic algorithm", IEEE international symposium on computational intelligence in robotics and automation, Vol. 2, 2003, pp 624–629.

[23] Zhong, Y.W., Yang, J.G., "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

[24] Kamaljit Kaur, Amit Chhabra, Gurvinder 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

[25] Fatma A. Omara, Mona M. Arafa, "Genetic algorithms for task scheduling problem", Journal of Parallel and Distributed Computing, Vol. 70, No.1, 2010, pp. 13–22.

[26] Probir Roy, Md. Mejbah Ul Alam and Nishita Das, "Heuristic based task scheduling in multiprocessor systems with genetic algorithm by choosing the eligible processor", International Journal of Distributed and Parallel Systems (IJDPS) Vol. 3, No. 4, July 2012.

[27] Amit Bansal and Ravreet Kaur, "Task Graph Scheduling on Multiprocessor System using Genetic Algorithm", International Journal of Engineering Research & Technology (IJERT), ISSN: 2278-0181, Vol. 1 Issue 5, July – 2012.

[28] Afaq Ahmad, Sayyid Samir Al-Busaidi, Mufeed Juma Al-Musharafi "On Properties of PN Sequences generated by LFSR – a Generalized Study and Simulation Modeling", Indian Journal of Science and Technology (IJST), vol. 6, no. 10, pp. 5351-5358, 2013.

[29] Ahmad, A. Al-Busaidi, S. S., Al Maashri, A., Awadalla, M., Rizvi, M. A. K., Mohanan, N., (2013) "Computing and Listing of Number of Possible m-Sequence Generators of Order n", Indian Journal of Science and Technology (IJST), vol. 6, no. 10, pp. 5359-5369, 2013

[30] Ahmad, A., and Bait-Shiginah, F., "A Nonconventional Approach to Generating Efficient Binary Gray Code Sequences", IEEE Potentials, vol. 31, no. 3, pp. 16-19, 2012

[31] Ahmad, A., Dawood Al-Abri, "Design of an Optimal Test Simulator for Built-In Self Test Environment", Journal of Engineering Research, vol. 7, no. 2, pp. 69 – 79, 2010

[32] Ahmad, A., and Mohammed M. Bait Suwailam, "A Less Complex Algorithmic Procedure for Computing Gray Codes", The Journal of Engineering Research, vol. 6, no. 2, , pp. 12 -19, 2009

[33] Ahmad, A., Al-Musharafi, M.J., and Al-Busaidi S., "A new algorithmic procedure to test m-sequences generating feedback connections of stream cipher's LFSRs", Proceedings IEEE conference on electrical and

electronic technology (TENCON'01), Singapore, August 19 – 22, 2001, vol. 1, pp. 366 - 369

[34] Ahmad A., "Achievement of higher testability goals through the modification of shift register in LFSR based testing," International Journal of Electronics (UK), vol. 82, no. 3, pp. 249-260, 1997

[35] Ahmad A. and Elabdalla A. M., "An efficient method to determine linear feedback connections in shift registers that generate maximal length pseudo-random up and down binary sequences", Computer & Electrical Engineering - An Int'l Journal (USA), vol. 23, no. 1, pp. 33-39, 1997