International Journal of Soft Computing 5 (2): 42-51, 2010
ISSN: 1816-9503

Modified Genetic Algorithm for Task Scheduling in Homogeneous Parallel System Using Heuristics

[1]Kamaljit Kaur, [2]Amit Chhabra and [3]Gurvinder Singh
Department of Computer Science and Engineering,
Guru Nanak Dev University, Amritsar-143001, Punjab, India

Abstract: Multiprocessor task scheduling is an important and computationally difficult problem. Multiprocessors have emerged as a powerful computing means for running real-time applications especially due to limitation of uni-processor system for not having sufficient enough capability to execute all the tasks. Multiprocessor computing environment requires an efficient algorithm to determine when and on which processor a given task should execute. A task can be partitioned into a group of subtasks and represented as a DAG (Directed Acyclic Graph) that problem can be stated as finding a schedule for a DAG to be executed in a parallel multiprocessor system. The problem of mapping meta-tasks to a machine is shown to be NP-complete. The NP-complete problem can be solved only using heuristic approach. The execution time requirements of the applications tasks are assumed to be stochastic. In multiprocessor scheduling problem, a given program is to be scheduled in a given multiprocessor system such that the program's execution time should be minimized. The last job must be completed as early as possible. Genetic Algorithm (GA) is one of the widely used techniques for constrained optimization. Performance of genetic algorithm can be improved with the introduction of some knowledge about the scheduling problem represented by the use of heuristics. In this study the problem of same execution time or completion time and same precedence in the homogeneous parallel system is resolved by using concept of Bottom-level (b-level) or Top-level (t-level). This combined approach named as Modified Genetic Algorithm (MGA) based on MET (Minimum execution time)/Min-Min heuristics and b-level or t-level precedence resolution is finally compared with a pure genetic algorithm, min-min heuristic, MET heuristic and First Come First Serve (FCFS) approach. Results of the experiments show that the modified genetic algorithm produces much better results in terms of quality of solutions.

Key words: DAG, multiprocessor scheduling, genetic algorithm, heuristics, NP-complete, bottom-level, top-level

INTRODUCTION

The problem of scheduling parallel tasks onto multiprocessors is to simply apportion a set of tasks to processors such that the optimal usage of processors and accepted computation time for scheduling algorithm are obtained (Ahmad *et al*., 1996; Kwok and Ahmad, 1999). The assumption of this study is based on the deterministic model that is the number of processors, the execution time of tasks, the relationship among tasks and precedence constraints are known in advance. The precedence constraints between tasks are represented by a Directed Acyclic Graph (DAG). In addition, the communication cost between two tasks is considered to be non-negligible and the multiprocessor system is not diverse and non-preemptive that is the processors are homogeneous and each processor completes the current task before the new one starts its execution. The complexity of the scheduling problem is very depended to the DAG, the number of processors, the execution time of tasks and also the performance criteria which would to be optimized. To date, many heuristic methods have been presented to schedule tasks on multiprocessor systems (Braunt *et al*., 2001; Grajcar, 1999, 2001; Izakian *et al*., 2009; Sutar *et al*., 2006; Ferner and Babb, 1999; McCreary *et al*., 2002). Also, there are many studies have been used for task scheduling based on GA (Hou *et al*., 1994; Rehmani and Vahedi, 2008; Lee and Chen, 1999; Rahmani and Rezvani, 2009; Rinehart *et al*., 2003; Wang and Korfhage, 2005; Page and Naughton, 2004; Carretero *et al*., 2007; Wu *et al*., 2004; Bohler *et al*., 1999; Golub and Kasapovic, 2002; Nikravan and Kashani, 2007; Zhou *et al*., 2006). GA is a problem solving strategy based on Darwinian evolution which has been successfully used for optimization problems (Goldberg, 1990; Mitchell, 1998). The aim of this study is to present a GA to decrease the computation time for finding a suboptimal schedule.

Corresponding Author: Kamaljit Kaur, Department of Computer Science and Engineering, Guru Nanak Dev University, Amritsar-143001, Punjab, India

MATERIALS AND METHODS

Task scheduling problem: Parallel Multiprocessor system scheduling can be classified into many different classes based on the characteristics of the tasks to be scheduled, the multiprocessor system and the availability of information. This study focus on a deterministic scheduling problem. A deterministic scheduling problem (Ahmad *et al*., 1996; Kwok and Ahmad, 1999) is one in which all information about the tasks and the relation to each other such as execution time and precedence relation are known to the scheduling algorithm in advance. The tasks should be non-preemptive i.e., task execution must be completely done before another task takes control of the processor and the processor environment is homogeneous. Homogeneous of processor means that the processors have same speeds or processing capabilities.

The main objective is to minimize the total task completion time (execution time + waiting time or idle time). The multiprocessor computing consists of a set of m homogeneous processor:

$$P = \{p_i: i = 1, 2, 3...m\}$$

They are fully connected with each other via identical links. Figure 1 shows a fully connected three parallel system with identical link.

Consider a directed acyclic task graph G = {V, E} of n nodes. Each node V = {T1, T2,....,Tn} in the graph represents a task. Aim is to map every task to a set P = {P1, P2, . . . , Pm} of m processors. Each task Ti has a weight Wi associated with it which is the amount of time the task takes to execute on any one of the m homogeneous processors. Each directed edge eij indicates dependence between the two tasks Ti and Tj that it connects. If there is a path from node Ti to node Tj in the graph G then Ti is the predecessor of Tj and Tj is the successor of Ti. The successor task cannot be executed before all its predecessors have been executed and their results are available at the processor at which the successor is scheduled to execute. A task is ready to execute on a processor if all of its predecessors have completed execution and their results are available at the processor on which the task is scheduled to execute. If the next task to be executed on a processor is not yet ready, the processor remains idle until the task is ready. The elements set C are the weights of the edges as:

$$C = \{C1, C2, C3...Cr\}$$

The value cij ∈ C is the communication cost incurred along the edge eij. It represents the data communication between the two tasks, if they are scheduled to different processors. But if both tasks are scheduled to the same processor then the weight associated to the edge becomes null (Hou *et al*., 1994; Lee and Chen, 1999). A DAG which has eleven tasks according to their height and their execution time (the time needed for a task to execute) is shown in Fig. 2.

T-level (Ti) is defined to be the length of the longest path in the task graph from an entry task to Ti excluding the execution cost of Ti. Symmetrically, b-level (Ti) is the length of the longest path from Ti to an exit task including the execution cost of Ti. Equation 1 and 2 are equation definitions of t-level (Ti) and b-level (Ti). Notice that we consider communication costs while calculating values t-level and b-level (Rahmani and Vahedi, 2008).

$$T-level(Ti) = \max_{Tj \in pred(Ti)} \{tlevel(Tj) + wj + cji\} \quad (1)$$

$$B-level(Ti) = wi + \max_{Ti \in succ(Ti)} \{cij = blevel(Tj)\} \quad (2)$$

Minimum Execution Time (MET): Assigns each task in arbitrary order to the machine with the best expected

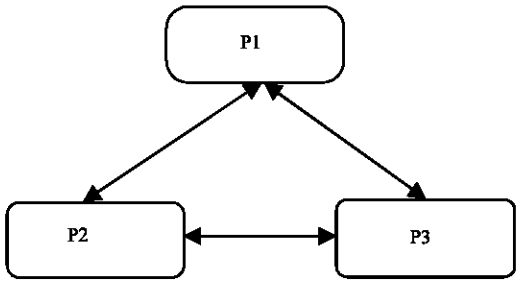

Fig. 1: A fully connected parallel processor

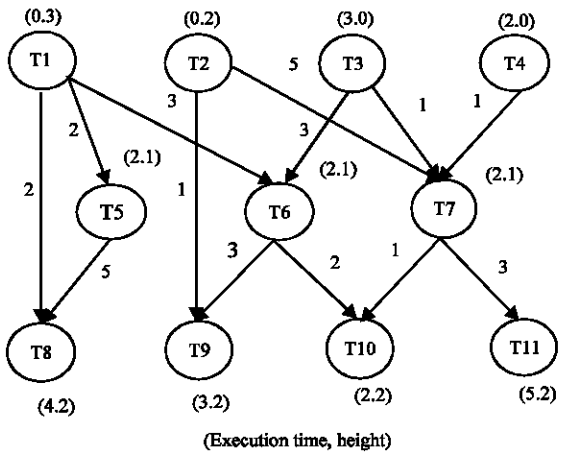

Fig. 2: An example of a DAG

execution time for that task regardless of that machine's availability. The motivation behind MET is to give each task to its best machine (Brunt *et al*., 2001; Izakian *et al*., 2009).

Min-min heuristic: Uses Minimum Completion Time (MCT) as a metric, meaning that the task which can be completed the earliest is given priority. This heuristic begins with the set U of all unmapped tasks. Then the set of minimum completion times, M = {min (completion-time (Ti, Mj)) for (1≤i≤n, 1≤j≤m)} is found. M consists of one entry for each unmapped task. Next, the task with the overall minimum completion time from M is selected and assigned to the corresponding machine and the workload of the selected machine will be updated. And finally the newly mapped task is removed from U and the process repeats until all tasks are mapped (i.e., U is empty) (Braunt *et al*., 2001; Izakian *et al*., 2009).

Genetic algorithms: A genetic algorithm starts with an initial population that evolves through generations and to reproduce depends on its fitness (Goldberg, 1990; Mitchell, 1998). In this case, the fitness of an individual is defined as the difference between its makespan and the one of the individuals having the largest makespan in the population. The best individual corresponds to the one having the smallest makespan and the largest fitness.

Next, the operators that compose a genetic algorithm are reviewed. The selection operator allows the algorithm to take biased decisions favour good individuals when changing generations. For this, some of the good individuals are replicated while some of the bad individuals are removed. As a consequence, after the selection, the population is likely to be dominated by good individuals. Starting from a population P1, this transformation is implemented iteratively by generating a new population P2 of the same size as P1.

Genetic algorithms are based on the principles that crossing two individuals can result an off springs that are better than both parents and slight mutation of an individual can also generate a better individual. The crossover takes two individuals of a population as input and generates two new individuals by crossing the parents characteristics.

The offsprings keep some of the characteristics of the parents. The mutation randomly transforms an individual that was also randomly chosen. It is important to notice that the size of the different populations is same. The structure of the algorithm is a loop composed of a selection followed by a sequence of crossovers and a sequence of mutations. After the crossovers, each individual of the new population is mutated with some (low) probability. This probability is fixed at the beginning of the execution and remains constant. The termination condition may be the number of iterations, execution time, results stability, etc. (Goldberg, 1990; Hou *et al*., 1994); (Rahmani and Vahedi, 2008; Correa *et al*., 1999).

MGA: The suggested algorithm: GAs operates through a simple cycle of stages creation of population strings, evaluation of each string, selection of the best strings and reproduction to create a new population. The number of genes and their values in each chromosome depend on the problem specification.

In this study, the number of genes of each chromosome is equal to the number of the nodes (tasks) in the DAG and the gene values demonstrate the scheduling priority of the related task to the node (each chromosome shows a scheduling) where the higher priority means that task must be executed early. In the basic genetic algorithm the initial population is generated randomly which can cause to generate more bad results.

To avoid the generation of non-optimal results, heuristic approach can be applied to generate the initial population that gives better results in terms of quality of solutions.

Coding of solutions: For multiprocessor scheduling problem, a schedule is one that satisfies following conditions:

- The precedence relations among the tasks are satisfied
- Every task is present and appears only once in the schedule (Hou *et al*., 1994; Rahmani and Vahedi, 2008)

A schedule can be represented as several lists of computational tasks (Fig. 3). Each list corresponds to computational tasks executed on a processor and order of tasks in the list indicates the order of execution.

Population initialization: The next step in the GAs is the creation of the initial population. Number of processors, number of tasks and population size are needed to generate initial population. Each individual of the initial population is generated through a minimum execution time or min-min heuristic along with b-level or t-level (Fig. 4 and 5) precedence resolution to avoid the problem of same execution time or completion time and same precedence Table 1. The problem of same execution

time/completion time and precedence can occur in the homogeneous parallel system as all the processors take same execution time to execute one task. The task to be scheduled for each iteration is determined by the following rules:

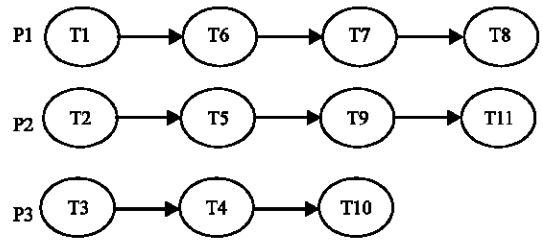

Fig. 3: List representation of a schedule

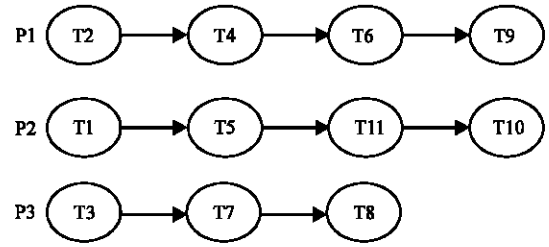

Fig. 4: Initial population of Fig. 2 using b-level resolution

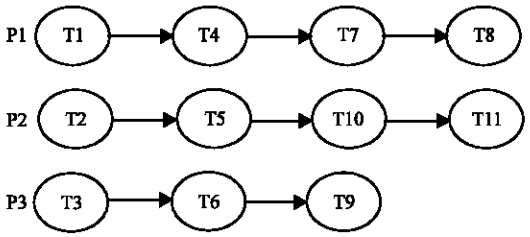

Fig. 5: Initial population of Fig. 2 using t-level resolution

- Sort the tasks according to their execution time/completion time in ascending order according to the Minimum Execution Time (MET)/Min-Min heuristic
- Calculate the bottom-level of each task
- Sort the tasks with the same execution time/ completion time and same precedence according to their bottom-level in descending order
- Assign the tasks to the processors in the order of their bottom-level

Or the task to be scheduled for each iteration is determined by the following rules:

- Sort the tasks according to their execution time/completion time in ascending order according to the Minimum Execution Time (MET)/Min-Min heuristic.
- Calculate the top-level of each task
- Sort the tasks with the same execution time/completion time and same precedence according to their top-level in ascending order
- Assign the tasks to the processors in the order of their top-level

The length of all individuals in an initial population is equal to the number of tasks in the DAG. For example: the initial population of Fig. 2 is generated.

Fitness value: Several optimization criteria can be considered for this problem, certainly the problem is multiobjective in its general formulation (Carretero *et al*., 2007). The elementary criterion is that of minimizing the makespan that is the time when finishes the latest job. A secondary criterion is to minimize the flowtime that is, minimizing the sum of finalization times of all the jobs. These two criteria are defined as follows:

$$\text{makespan: } \min_{S_i \in \text{Sched}\, j \in \text{Jobs}} \{\max F_j\}$$

Table 1: Priority of execution of tasks based on their execution time, completion time, bottom-level and top-level

Task number	Execution time	Complection time	Bottom level	Top level	Order of execution according to execution time	Order of execution according to complection time	Order of execution according to bottom level	Order of execution according to top level
1	3	3	16	0	7	2	2	1
2	2	2	17	0	1	1	1	2
3	3	3	14	0	8	3	3	3
4	2	4	13	0	2	4	4	4
5	2	5	11	5	3	5	5	5
6	2	7	8	6	4	7	7	6
7	2	7	10	7	5	6	6	7
8	4	9	4	12	10	8	9	10
9	3	12	3	11	9	9	10	9
10	2	14	2	10	6	11	11	8
11	5	12	5	12	11	10	8	11

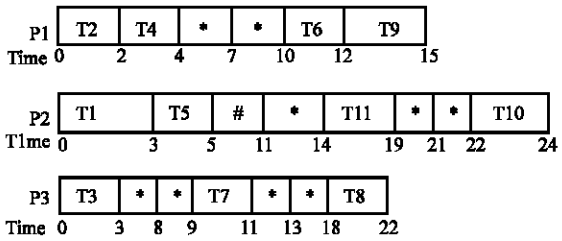

Fig. 6: Assignment of tasks to processors using b-level resolution

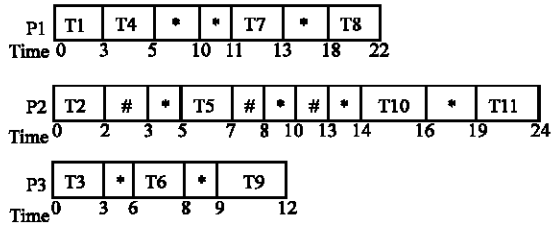

Fig. 7: Assignment of tasks to processors using t-level resolution

and $$\text{flowtime: } \min_{Si \in Sched} \left\{ \sum_{j \in Jobs} F_j \right\}$$

F_j denotes the time when job j finalizes, Schd is the set of all possible schedules and jobs is the set of all jobs to be scheduled. For example fitness value of the initial population is as show in Fig. 6.

The fitness value is calculated in terms of Makespan and Flowtime as discussed before as:

Makespan = 24 time units
Flowtime = 3+2+3+4+5+12+11+22+15+24+19 = 120 time units
Makespan = 24 time units
Flowtime = 3+2+3+5+7+8+13+22+12+16+24 = 115 time units (Fig. 7)

The * denotes the communication time and # denotes the waiting time.

Selection operator: The design of the fitness function is the basic of selection operation, the design of the fitness function will directly affect the performance of genetic algorithm. GAs uses selection operator to select the superior and eliminate the inferior. The individuals are selected according to their fitness value. Once fitness values have been evaluated for all chromosomes, good chromosomes can be selected through rotating roulette wheel strategy. This operator generate next generation by selecting best chromosomes from parents and offspring.

Crossover operator: Crossover operator randomly selects two parent chromosomes (chromosomes with higher values have more chance to be selected) and randomly chooses their crossover points and mates them to produce two child (offspring) chromosomes. In this study one and two point crossover operators are examined. In one point crossover, the segments to the right of the crossover points are exchanged to form two offspring as shown in Fig. 8a and in two point crossover (Goldberg, 1990; Rahmani and Vahedi, 2008), the middle portions of the crossover points are exchanged to form two offspring as shown in Fig. 8b.

Mutation: Ensures that the probability of finding the optimal solution is never zero. It also acts as a safety net to recover good genetic material that may be lost through selection and crossover. Implementation of two mutation operators is there in MGA. The first one selects two tasks randomly and swaps their allocation parts. The second one selects a task and alters its allocation part at random. These operators can always generate feasible off spring, too. Figure 9a-d demonstrate the mutation operation.

Makespan = 24 time units
Flowtime = 3+2+3+4+5+12+11+22+15+24+19 = 120 time units
Makespan = 20 time units
Flowtime = 3+2+3+4+5+8+7+19+14+20+15 = 100 time units

The mutation operation swaps task t6 on processor p1 to task t7 on processor p3

Makespan = 20 time units
Flowtime = 3+2+3+4+5+8+7+18+12+20+15 = 97 time units

The mutation operation swaps task t9 on processor p1 to task t8 on processor p3.

Makespan = 14 time units
Flowtime = 3+2+3+4+5+8+7+9+12+14+12 = 79 time unit

The mutation operation swaps task t8 on processor p1 to task t11on processor p2 that takes 14 time units to complete. The procedure of the Suggested Modified Genetic Algorithm is:

Step 1: Setting the parameter: Read DAG (number of tasks n, number of processors m and comm. cost), population size pop-size, crossover probability pc, mutation probability pm and maximum generation maxgen. Let generation gen = 0.

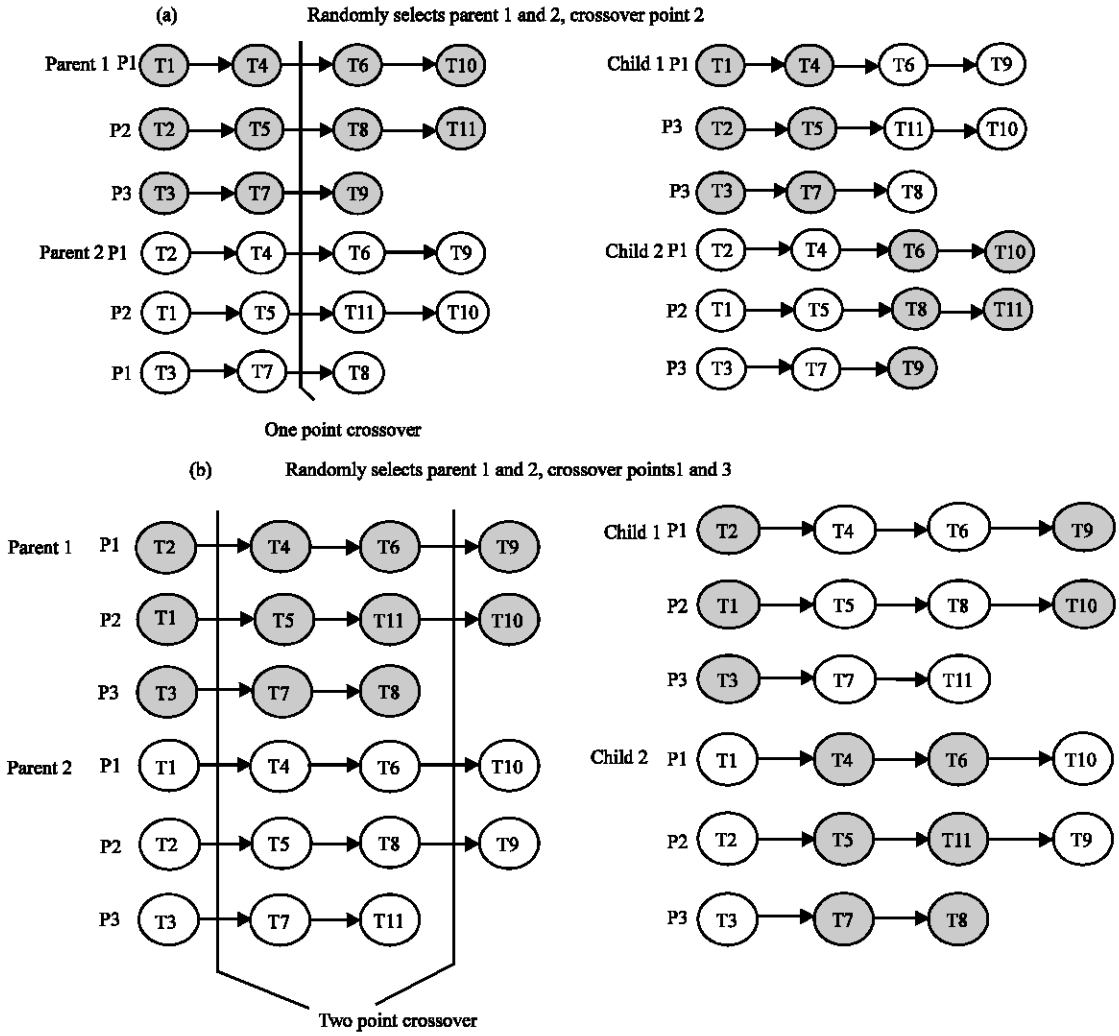

Fig. 8: (a) One point crossover and (b) two point crossover

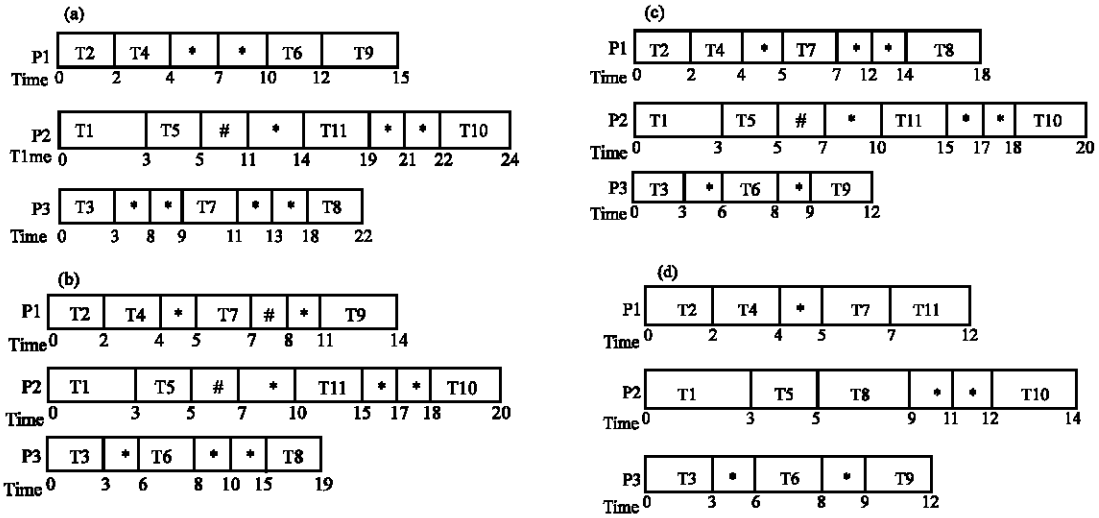

Fig. 9: (a) A Gantt chart before mutation operation, (b) A Gantt chart after swap mutation operation, (c) A Gantt chart after swap mutation and (d) A Gantt chart after swap mutation operation

Step 2: Initialization: Generate pop_size chromosomes using Mnimum Execution Time (MET)/Min-Min heuristic and b-level/t-level precedence resolutions.

Step 3: Evaluate: Calculate the fitness value of each chromosome.

Step 4: Crossover: Perform the crossover operation on the chromosomes selected with probability pc.

Step 5: Mutation: Perform the swap/move mutation on chromosomes selected with probability pm.

Step 6: Selection: Select pop-size chromosomes from the parents and offspring for the next generation.

Step 7: Stop testing: If gen = maxgen, then output best solution and stop.

Else gen = gen + 1 and return to step 3

RESULTS AND DISCUSSION

Performance analysis: The final best schedule obtained by applying the suggested algorithm to the DAG of Fig. 2 onto the parallel multiprocessor system in Fig. 1 is shown in Fig. 10 and 11.

The completion time obtained by modified method using b-level resolution is 14 time units and with t-level resolution is 16 time units. We also compare the results with FCFS scheduling method, min-min scheduling method, MET scheduling method and also with the Basic Genetic Algorithm (BGA) (Hou *et al.*, 1994) on parallel systems and execution of the schedule are shown in Fig. 12-15.

After applying the suggested modified GA, the best schedule found using b-level resolution is:

P1: T2→T4→T7→T11
P2: T1→T5→T8→T10
P3: T3→T6→T9

Makespan = 14 time units
Flowtime = 3+2+3+4+5+8+7+9+12+14+12 = 79 time units

After applying the suggested modified GA, the best schedule found using t-level precedence resolution is:

P1: T2→T4→T7→T11
P2: T1→T5→T10→T8
P3: T3→T6→T9
Makespan = 16 time units
Flowtime = 3+2+3+4+5+8+7+16+12+12+12 = 84 time units

Min-min scheduling policy assigns the tasks to processors p1, p2 and p3 as:

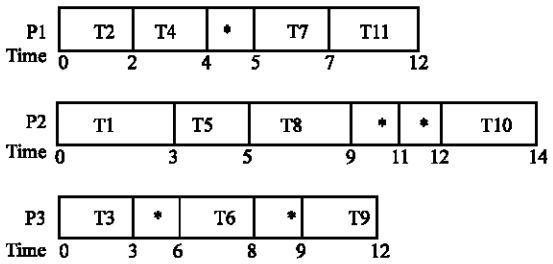

Fig. 10: A gantt chart of suggested modified genetic algorithm using b-level resolution

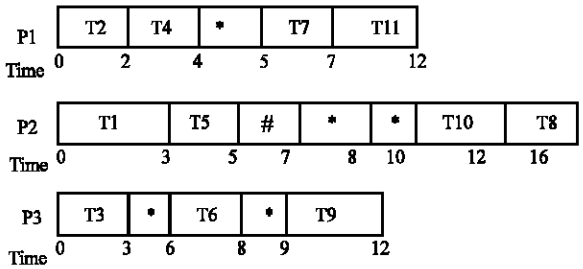

Fig. 11: A gantt chart of suggested modified genetic algorithm using t-level resolution

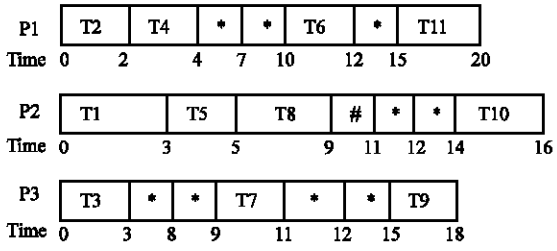

Fig. 12: A gantt chart of Min-min scheduler

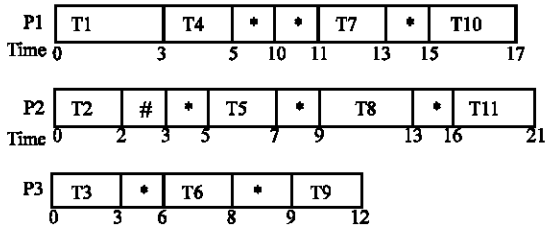

Fig. 13: A gantt chart of FCFS scheduler

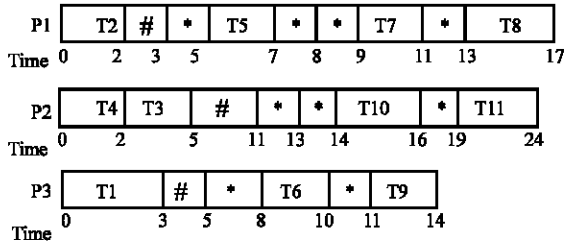

Fig. 14: A gantt chart of MET scheduler

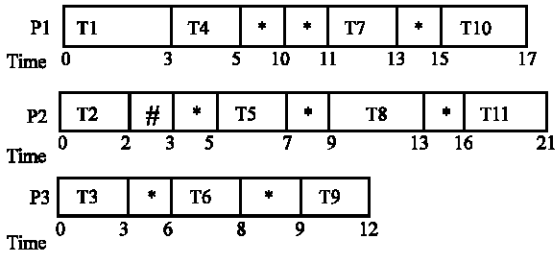

Fig. 15: A gantt chart of BGA scheduler

P1: T2→T4→T6→T11
P2: T1→T5→T8→T10
P3: T3→T7→T9

Makespan = 20 time units
Flowtime = 3+2+3+4+5+12+11+9+18+16+20 = 103 time units

FCFS scheduling Policy assign the tasks to processors p1, p2 and p3 as:

P1: T1→T4→T7→T10
P2: T2→T5→T8→T11
P3: T3→T6→T9

Makespan = 21 time units
Flowtime = 3+2+3+5+7+8+13+13+12+17+21 = 104 time units

Minimum Execution Time (MET) Scheduling Policy assigns the tasks to processors p1, p2 and p3 as:

P1: T2→T5→T7→T8
P2: T4→T3→T10→T11
P3: T1→T6→T9

Makespan = 24 time units
Flowtime = 3+2+5+2+7+10+11+17+14+16+24 = 111 time units

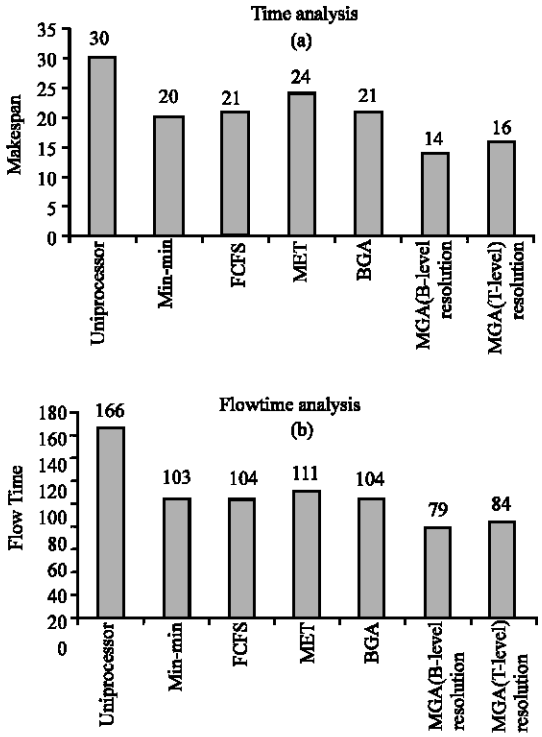

Fig. 16: Experimental results for (a) Makespan (b) Flowtime

After applying the Basic GA, the best schedule found is:

P1: T1→T4→T7→T10
P2: T2→T5→T8→T11
P3: T3→T6→T9

Makespan = 21 time units
Flowtime = 3+2+3+5+7+8+13+13+12+17+21 = 104 time units.

In Fig. 16 a,b it is clear that MGA can considerably decreases the scheduling time.

Performance analysis

Suggested modified GA using b-level resolution: Speed up (S): speed up is defined as the completion time on a uniprocessor divided by completion time on a multiprocessor system:

S = 30/14= 2.142

Efficiency (E): (S * 100)/m where m is the number of processors.

E = (2.142 * 100) / 3 = 71.42%

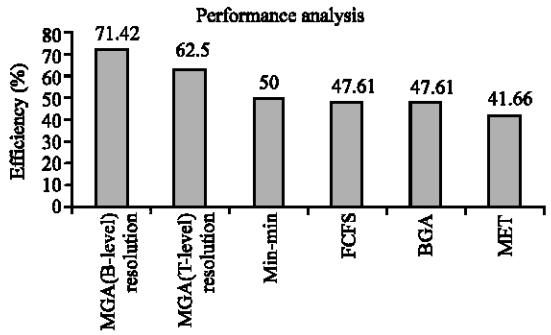

Fig. 17: Performance analysis of min-min, FCFS, BGA, MET, MGA algorithms

Suggested Heuristics based GA using t-level resolution:
S = 30/16 = 1.875
E = (18.75 *100)/3 = 62.5%

Min-min scheduler:
S = 30/20 = 1.5
E = (1.5 * 100)/3 = 50%

FCFS scheduler:
S = 30/21 = 1.428
E = (1.428 * 100)/3 = 47.61%

MET scheduler:
S = 30/24 = 1.25
E = (1.25 * 100)/3 = 41.66%

BGA scheduler:
S = 30/21 = 1.428
E = (1.428 * 100)/3= 47.61%

The performance analysis of various scheduling schemes is shown in Fig. 17.

CONCLUSION

In this study, a Modified Genetic Algorithm for task scheduling in homogeneous parallel multiprocessor system is suggested that tends to minimize the completion time and increase the throughput of the system. The heuristics based method found a best solution for assigning the tasks to the homogeneous parallel multiprocessor system.

Experimental results and performance of the heuristics based GA with different precedence resolution is compared with Min-min, MET, FCFS and BGA Scheduling method and shows the efficiency of 71.42%. The performance study is based on the best randomly generated schedule of the proposed GA.

REFERENCES

Ahmad, I., Y.K. Kwok and M.Y. Wu, 1996. Analysis, evaluation and comparison of algorithms for scheduling task graphs on parallel processors. Proceedings of the 1996 International Symposium on Parallel Architectures, Algorithms and Networks, June 12-14, IEEE Computer Society Washington, DC, USA., pp: 207-207.

Bohler, M., F. Moore and Y. Pan, 1999. Improved multiprocessor task scheduling using genetic algorithms. Proceedings of the 12th International Florida Artificial Intelligence Research Society Conference, May 1-5, AAAI Press, OH., pp: 140-146.

Braunt, T.D., H.J. Siegel, N. Beck, B. Yao and R.F. Freund, 2001. A comparison study of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. J. Parallel Distrib. Comput., 61: 810-837.

Carretero, J., F. Xhafa and A. Abraham, 2007. Genetic algorithm based schedulers for grid computing systems. Int. J. Innovative Comput. Inform. Control, 3: 1053-1071.

Correa, R.C., A. Ferreira and P. Rebreyend, 1999. Scheduling multiprocessor tasks with genetic algorithms. IEEE Trans. Parallel Distrib. Syst., 10: 825-837.

Ferner, C.S. and R.G. Babb, 1999. Automatic choice of scheduling heuristics for parallel/distributed computing. Scientific Programm., 7: 47-65.

Goldberg, D.E., 1990. Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley Publishing Co. Inc., Boston, MA.

Golub, M. and S. Kasapovic, 2002. Scheduling multiprocessor tasks with genetic algorithms. Proceedings of the International Conference on Applied Informatics, Feb. 18-21, OACTA Press, Austria, pp: 273-278.

Grajcar, M., 1999. Genetic list scheduling algorithm for scheduling and allocating on a loosely coupled heterogeneous multiprocessor system. Proceedings of the 36th annual ACM/IEEE Design Automation Conference, June 21-25, New Orleans, Louisiana, USA., pp: 280-285.

Grajcar, M., 2001. Strengths and weaknesses of genetic list scheduling for heterogeneous systems. Proceedings of the Second International Conference on Application of Concurrency to System Design, June 25-29, IEEE Computer Society, Washington, DC, USA., pp: 123-123.

Hou, E.S.H., N. Ansari and H. Ren, 1994. A genetic algorithm for multiprocessor scheduling. IEEE Trans. Parallel Distributed Syst., 5: 113-120.

Izakian, H., A. Abraham and V. Snasel, 2009. Comparison of heuristics for scheduling independent tasks on heterogeneous distributed environments. Proc. Int. Joint Conf. Computat. Sci. Optimizat., 1: 8-12.

Kwok, Y.K. and I. Ahmad, 1999. Static scheduling algorithms for allocating directed task graphs to multiprocessors. ACM Comput. Surveys, 31: 406-471.

Lee, Y.H. and C. Chen, 1999. A modified genetic algorithm for task scheduling in multiprocessor systems. Proceedings of the 6th International Conference Systems and Applications, (ICSA'99), IEEE Computer Society Washington, DC, USA., pp: 382-387.

McCreary, C.L., A.A. Khan, J. Thompson and M.E. McArdle, 2002. A comparison of heuristics for scheduling DAGs on multiprocessors. Proceedings of 8th International Symposium on Parallel Processing, (ISPP'02), Cancun, Qr, Mexico, pp: 446-451.

Mitchell, M., 1998. An Introduction to Genetic Algorithms. The MIT Press, USA., ISBN-10: 0262631857, pp: 221.

Nikravan, M. and M.H. Kashani, 2007. A genetic algorithm for process scheduling in distributed operating systems considering load balancing. Proceedings 21st European Conference on Modelling and Simulation Ivan Zelinka, (ECMS'07), Zuzana Oplatková, Alessandra Orsoni, pp: 1-6.

Page, A.J. and T.J. Naughton, 2004. Framework for Task scheduling in heterogeneous distributed computing using genetic algorithms. Proceedings of the 15th Artificial Intelligence and Cognitive Science Conference (AICS'04), Castlebar Co., Mayo, Ireland, pp: 137-146.

Rahmani, A.M. and M. Rezvani, 2009. A novel genetic algorithm for static task scheduling in distributed systems. Int. J. Comput. Theor. Eng., 1: 1793-8201.

Rahmani, A.M. and M.A. Vahedi, 2008. A Novel Task Scheduling in Multiprocessor Systems with Genetic Algorithm by Using Elitism Stepping Method. Science and Research Branch, Tehran, Iran.

Rinehart, M., V. Kianzad and S.S. Bhattacharyya, 2003. A Modular Genetic Algorithm for Scheduling Task Graphs. Institute for Advanced Computer Studies, University of Maryland at College Park, Maryland.

Sutar, P.S.R., J.P. Sawant and J.R. Jadhav, 2006. Task scheduling for multiprocessor systems using memetic algorithms. Proceedings of the 4th International Working Conference, May 7-10, IEEE Computer Society Washington, DC, USA., pp: 27/1-27/9.

Wang, P.C. and W. Korfhage, 2005. Process scheduling with genetic algorithms. Proceedings of the 7th IEEE Symposium on Parallel and Distributed Processing, Oct. 25-28, IEEE Computer Society Washington, DC, USA., pp: 638-638.

Wu, A.S., H. Yu, S. Jin, K.C. Lin and G. Schiavone, 2004. An incremental genetic algorithm approach to multiprocessor scheduling. IEEE Trans. Parallel Distrib. Syst., 15: 824-834.

Zhou, S.E., Y. Liu and D. Jiang, 2006. A genetic-annealing algorithm for task scheduling based on precedence task duplication. Proceedings of the 6th IEEE International Conference on Computer and Information Technology, Sept. 20-22, IEEE Computer Society Washington, DC, USA., pp: 117-117.