Heuristics Based Genetic Algorithm for Scheduling Static Tasks in Homogeneous Parallel System

Kamaljit Kaur *kamal.aujla86@gmail.com*
Department of Computer Science & Engineering,
Guru Nanak Dev University,
Amritsar- 143001, Punjab, India

Amit Chhabra *chhabra_amit78@yahoo.com*
Department of Computer Science & Engineering,
Guru Nanak Dev University,
Amritsar- 143001, Punjab, India

Gurvinder Singh *gsbawa71@yahoo.com*
Department of Computer Science & 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 that a uni-processor system would not be sufficient enough to execute all the tasks. That 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 paper 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 heuristics based genetic algorithm (HGA) based on MET (Minimum execution time)/Min-Min heuristics and b-level or t-level precedence resolution and is 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 heuristics based genetic algorithm produces much

better results in terms of quality of solutions.

Keywords: DAG, multiprocessor scheduling, genetic algorithm, heuristics.

1. 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 [1,2]. The assumption of this paper 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 [5, 9, 10, 11, 16, 18, 19]. Also, there are many studies have been used for task scheduling based on GA [7, 8, 12, 13, 14, 15, 16, 17, 20, 21, 22, 23]. GA is a problem solving strategy, based on Darwinian evolution, which has been successfully used for optimization problems [3, 4].

The aim of this paper is to present a GA to decrease the computation time for finding a suboptimal schedule.

This paper is divided as follows: In section 2 an overview of the problem is given along with brief description of the solution methodology. Section 3 provides a more detailed heuristics based genetic algorithm. Experimental results and performance analysis are provided in section 4 and conclusion follow in section 5.

2. PROBLEM STATEMENT

In this section, more prescribed multiprocessor scheduling problem and the principles of genetic algorithms are discussed.

2.1 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 paper focus on a deterministic scheduling problem. A deterministic scheduling problem [1, 2] 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

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

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

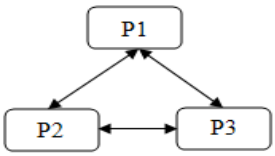


FIGURE 1: A fully connected parallel processor

Consider a directed acyclic task graph *G* = {*V,E*} of *n* nodes. Each node V = {*T*1*, T2,...., Tn*} in the graph represents a task. Aim is to map every task to a set *P* = {*P*1*,P*2*, . . . , 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 $C = \{c_k : k = 1, 2, 3 \ldots r\}$ 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 [7, 12].

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 Figure 2.

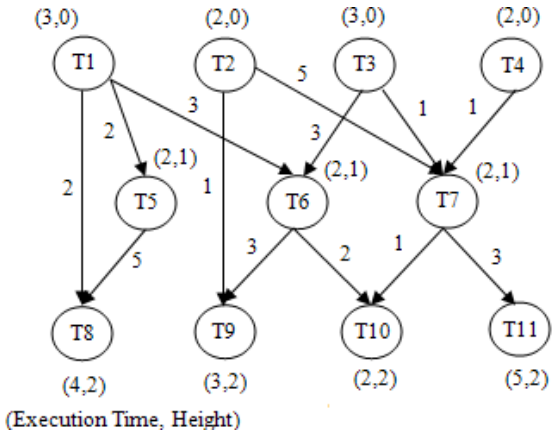


FIGURE 2: An example of a DAG

Tlevel(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, blevel(Ti) is the length of the longest path from Ti to an exit task, including the execution cost of Ti. Formula (2.1) and (2.2) are formal definitions of tlevel(Ti) and blevel(Ti). Notice that we consider communication costs while calculating values tlevel and blevel [8].

$$\text{tlevel}(T_i) = \max_{T_j \in \text{pred}(T_i)} \{\text{tlevel}(T_j) + W_j + c_{ji}\} \quad (2.1)$$

$$\text{blevel}(T_i) = W_i + \max_{T_j \in \text{succ}(T_i)} \{c_{ij} + \text{blevel}(T_j)\} \quad (2.2)$$

2.2 Minimum Execution Time (MET)

Minimum Execution Time (MET) assigns each task, in arbitrary order, to the machine with the best expected execution time for that task, regardless of that machine's availability. The motivation behind MET is to give each task to its best machine [5, 11].

2.3 Min-min Heuristic

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, $M = \{\min(completion\_time(T_i, M_j)) \text{ for } (1 \le i \le n, 1 \le j \le m)\}$, 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) [5, 11].

2.4 Genetic Algorithms

A genetic algorithm starts with an initial population that evolves through generations and to reproduce depends on its fitness [3, 4]. 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 favor 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 offsprings 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 [3, 7, 8, 6].

3. HGA: 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 paper,

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 along with precedence resolution can be applied to generate the initial population that gives better results in terms of quality of solutions.

3.1 Coding of Solutions

For multiprocessor scheduling problem, a schedule is one that satisfies following conditions.

1. The precedence relations among the tasks are satisfied
2. Every task is present and appears only once in the schedule [7, 8].

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.

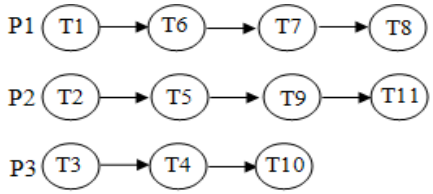


FIGURE 3: List Representation of a schedule

3.2 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 precedence resolution to avoid the problem of same execution time or completion time and same precedence. 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:

i. Sort the tasks according to their execution time/completion time in ascending order according to the minimum execution time (MET)/Min-Min heuristic.
ii. Calculate the bottom-level of each task.
iii. Sort the tasks with the same execution time/completion time and same precedence according to their bottom-level in descending order.
iv. 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:

i. Sort the tasks according to their execution time/completion time in ascending order according to the minimum execution time (MET)/Min-Min heuristic.
ii. Calculate the top-level of each task.
iii. Sort the tasks with the same execution time/completion time and same precedence according to their top-level in ascending order.
iv. 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 as:

Task Number	Execution Time	Completion Time	Bottom Level	Top Level	Order of execution According to Execution Time	Order of execution According to Completion 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

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

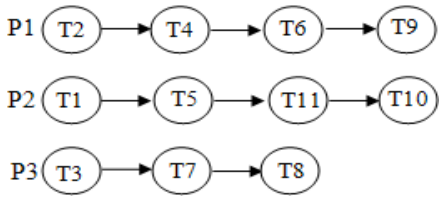


FIGURE 4: Initial Population of figure 1 using b-level resolution

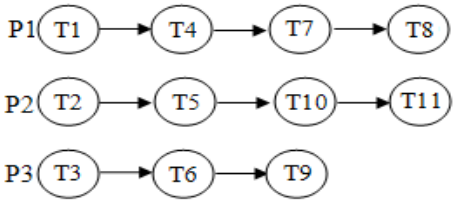


FIGURE 5: Initial Population of figure 1 using t-level resolution

3.3 Fitness Value

Several optimization criteria can be considered for this problem, certainly the problem is multiobjective in its general formulation [20]. 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:

$$\text{makespan} : \min_{S_i \in Sched} \{ \max_{j \in Jobs} F_j \} \quad \text{and}$$

$$\text{flowtime} : \min_{S_i \in Sched} \{ \sum_{j \in Jobs} F_j \}$$

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 follows:

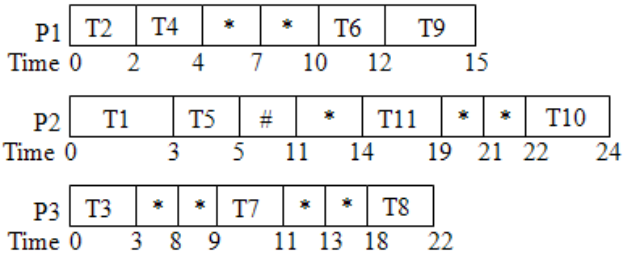


FIGURE 6: Assignment of tasks to processors using b-level resolution

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

Makespan = 24 time units
Flowtime = 3+2+3+4+5+12+11+22+15+24+19 = 120 time units.

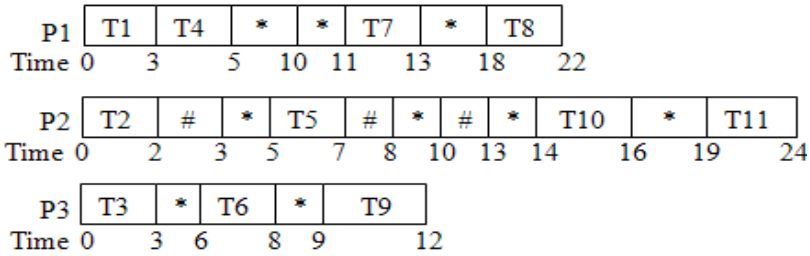


FIGURE 7: Assignment of tasks to processors using t-level resolution

Makespan = 24 time units
Flowtime = 3+2+3+5+7+8+13+22+12+16+24 = 115 time units.

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

3.4 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.

3.5 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 paper 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 figure 8 (a) and in two point crossover [3] [8], the middle portions of the crossover points are exchanged to form two offspring as shown in figure 8 (b).

Randomly selects Parent 1 & 2, crossover point 2

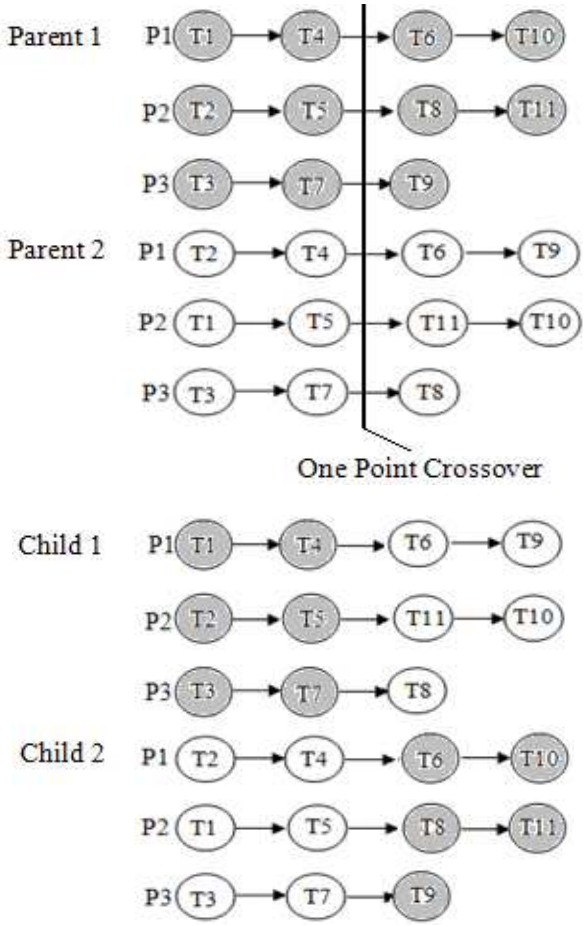
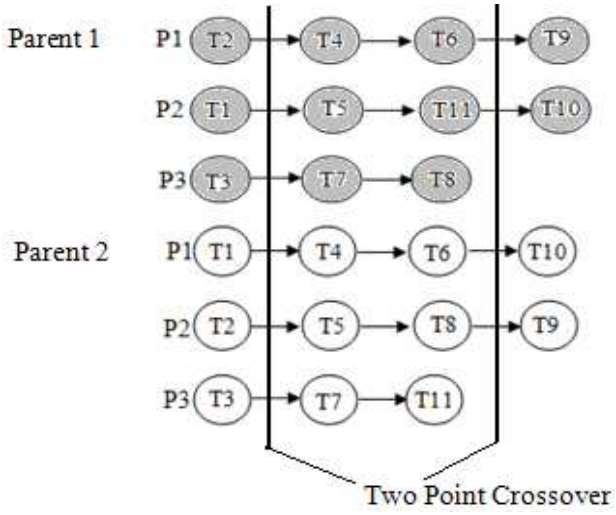


FIGURE 8(a): One Point Crossover

Randomly selects parent 1 & 2, crossover points 1 & 3



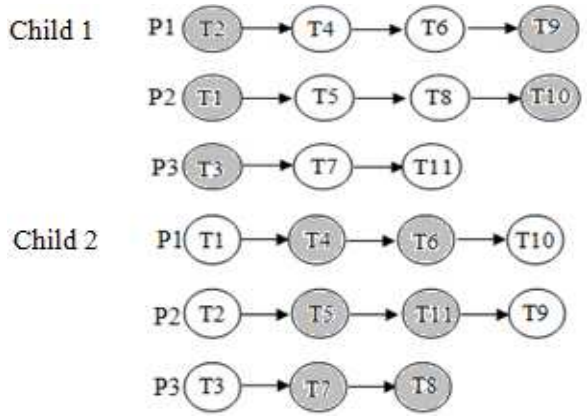


FIGURE 8(b): Two Point Crossover

3.6 Mutation

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 HGA. 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 offspring, too. Figure 9(a), 9(b), 9(c) & 9(d) demonstrate the mutation operation.

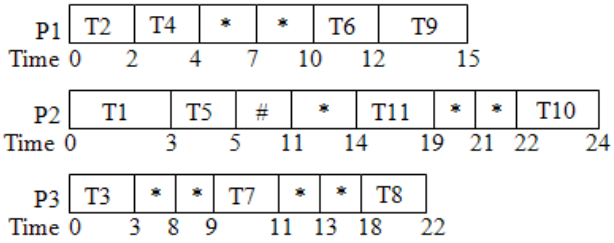


FIGURE 9(a): A Gantt chart before mutation operation

Makespan = 24 time units
Flowtime = 3+2+3+4+5+12+11+22+15+24+19 = 120 time units.

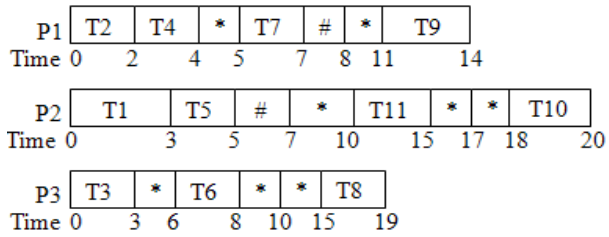


FIGURE 9(b): A Gantt chart after swap mutation operation.

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.

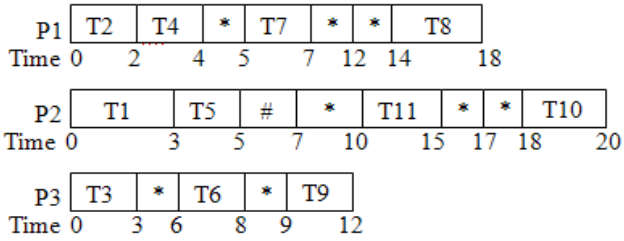


FIGURE 9(c): A Gantt chart after swap mutation operation.

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.

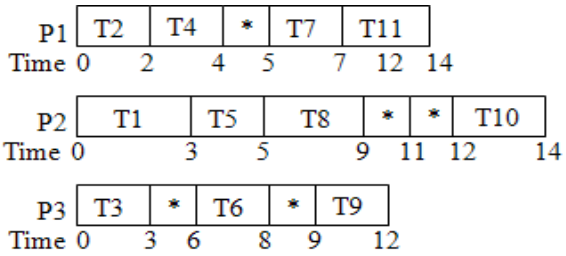


FIGURE 9(d): A Gantt chart after swap mutation operation

Makespan = 14 time units
Flowtime = 3+2+3+4+5+8+7+9+12+14+12 = 79 time units.
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 Heuristics based Genetic Algorithm is:

Step 1: Setting the parameter
Set 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
Step 2: Initialization
Generate pop_ size chromosomes using minimum 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

4. EXPERIMENTAL RESULTS & PERFORMANCE ANALYSIS

The final best schedule obtained by applying the suggested algorithm to the DAG of figure 2 onto the parallel multiprocessor system in figure 1, is shown in figure 10 & 11. The completion time obtained by heuristics based 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) [7] on parallel systems and execution of the schedule are shown in figure 12, 13, 14 & 15.

After applying the suggested heuristics based GA, the best schedule found using b-level precedence resolution is:
P1: T2→T4→T7→T11
P2: T1→T5→T8→T10
P3: T3→T6→T9

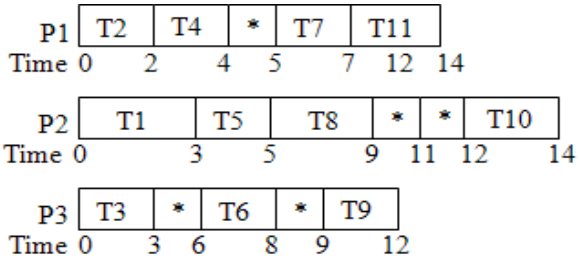


FIGURE 10: A Gantt chart of Suggested Heuristics based Genetic Algorithm using b-level resolution.

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

After applying the suggested heuristics based 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

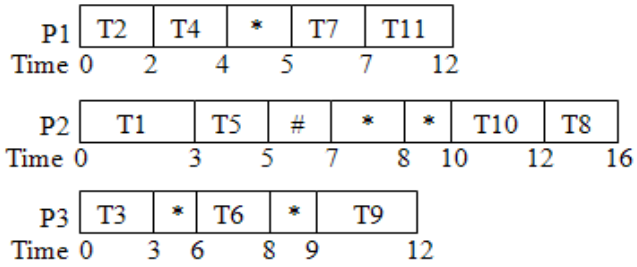


FIGURE 11: A Gantt chart of Suggested Heuristics based Genetic Algorithm using t-level resolution.

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 & p3 as:
P1: T2→T4→T6→T11
P2: T1→T5→T8→T10
P3: T3→T7→T9

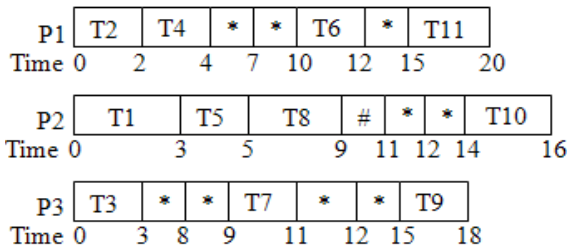


FIGURE 12: A Gantt chart of Min-min scheduler

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 & p3 as:
P1: T1→T4→T7→T10
P2: T2→T5→T8→T11
P3: T3→T6→T9

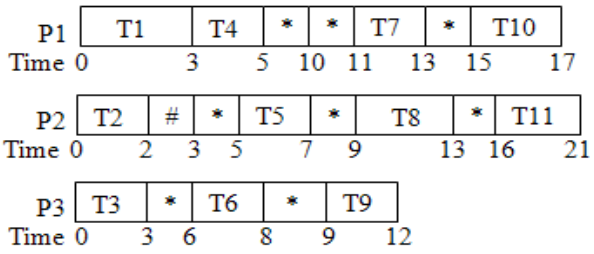


FIGURE 13: A Gantt chart of FCFS scheduler.

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 & p3 as:
P1: T2→T5→T7→T8
P2: T4→T3→T10→T11
P3: T1→T6→T9

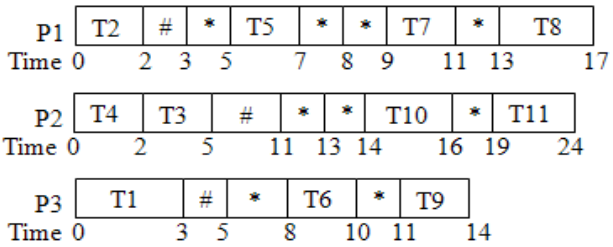


FIGURE 14: A Gantt chart of MET scheduler.

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

After applying the Basic GA, the best schedule found is:
P1: T1→T4→T7→T10
P2: T2→T5→T8→T11
P3: T3→T6→T9

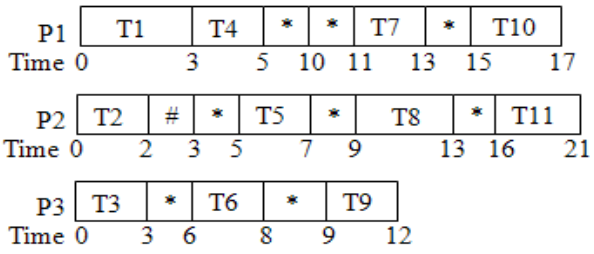
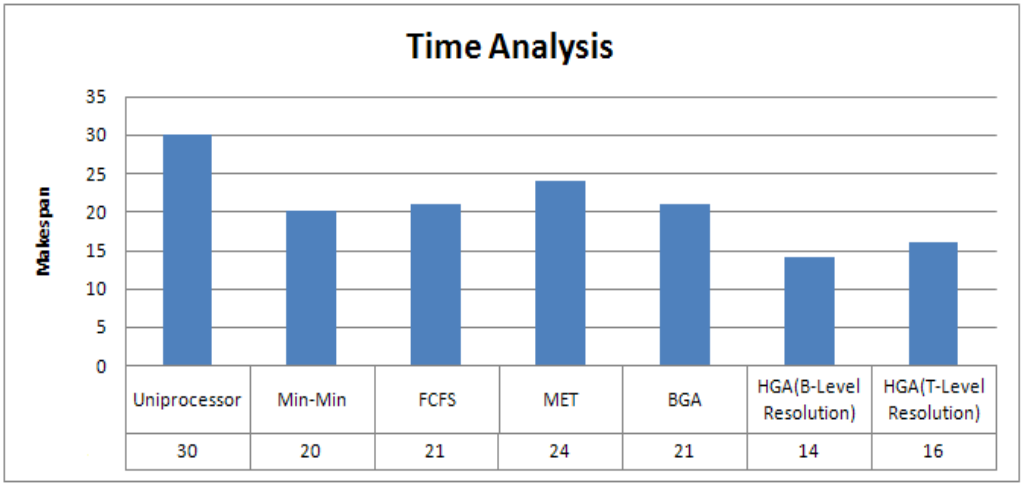


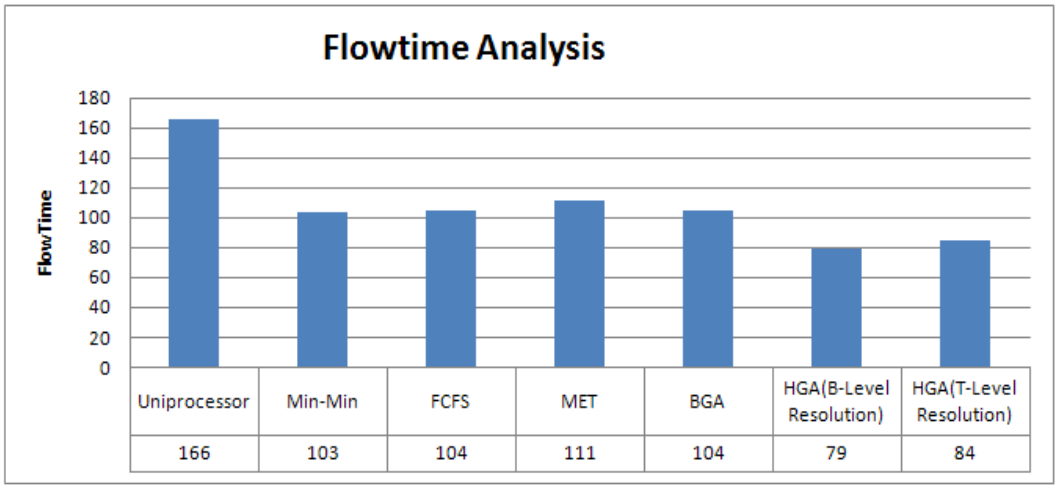
FIGURE 15: A Gantt chart of BGA scheduler.

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

In Figure 16 (a) & (b), it is clear that HGA can considerably decreases the scheduling time.



(a)



(b)

FIGURE 16: Experimental results for (a) Makespan (b) Flowtime

Performance Analysis

1. Suggested Heuristics based 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 %
2. Suggested Heuristics based GA using t-level resolution
   S = 30/16 = 1.875
   E = (1.875 * 100) / 3 = 62.5 %
3. Min-min Scheduler:
   S = 30/20 = 1.5
   E = (1.5 * 100) / 3 = 50 %
4. FCFS Scheduler:
   S = 30/21 = 1.428
   E = (1.428 * 100) / 3 = 47.61 %
5. MET Scheduler:

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

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

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

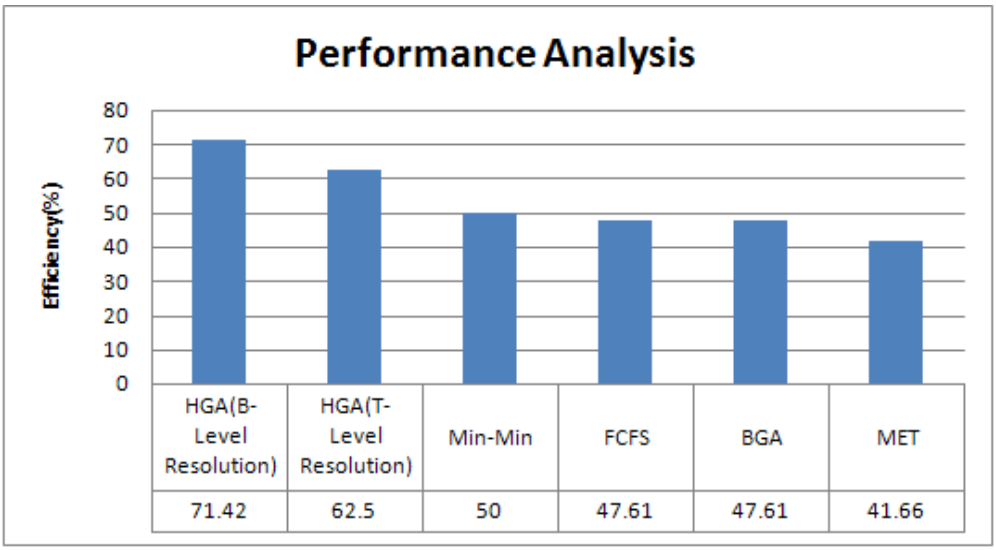


FIGURE 17: Performance analysis of min-min, FCFS, BGA, MET, HGA algorithms.

5. CONCLUSION

The task scheduling problem in the distributed systems is known to be NP-hard. The heuristic algorithms which obtain near-optimal solution in an acceptable interval time are preferred to the back tracking and the dynamic programming. The genetic algorithm is one of the heuristic algorithms which have the high capability to solve the complicated problems like the task scheduling.

In this paper, a new genetic algorithm, named Heuristics based Genetic Algorithm for Scheduling Static Tasks in Homogeneous parallel System is presented which its population size and the number of generations depends on the number of tasks. This algorithm 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 methods 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 suggested GA.

6. REFERENCES

[1] Ishfaq Ahmad, Yu-Kwong Kwok, Min-You Wu, “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, Page: 207, 1996, ISBN: 0-8186-7460-1, IEEE Computer Society Washington, DC, USA.

[2] Yu-Kwong Kwok and Ishfaq Ahmad, “Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors”, ACM Computing Surveys, vol. 31, Issue. 4, December 1999, ISSN: 0360-0300, ACM New York, NY, USA.

[3] D. E. Goldberg, “Genetic algorithms in search, optimization & machine learning”, Addison Wesley, 1990.

[4] Melanie Mitchell, "An Introduction to Genetic algorithms", The MIT Press, February 1998.
[5] Tracy D. Braunt, Howard Jay Siegel, Noah Beck, Bin Yao, Richard F. Freund, "A Comparison Study of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems", Journal of Parallel and Distributed Computing, Volume 61, Issue 6, June 2001, Pages: 810-837, ISSN: 0743-7315, Academic Press, Inc. Orlando, FL, USA.
[6] Ricardo C. Correa, Afonso Ferreira, Pascal Rebreyend, "Scheduling Multiprocessor Tasks With Genetic Algorithms", IEEE Transactions on Parallel and Distributed Systems, Vol. 10, Issue. 8, August 1999, Pages: 825-837, ISSN: 1045-9219, IEEE Press Piscataway, NJ, USA.
[7] Edwin S. H. Hou, Nirwan Ansari, Hong Ren, "A Genetic Algorithm for Multiprocessor Scheduling", IEEE Transactions on Parallel and Distributed Systems, vol. 5, Issue. 2, February2003, Pages: 113-120, ISSN: 1045-9219, IEEE Press Piscataway, NJ, USA.
[8] Amir Masoud Rahmani, Mohammad Ali Vahedi, "A novel Task Scheduling in Multiprocessor Systems with Genetic Algorithm by using Elitism stepping method", Science and Research branch, Tehran, Iran, May 26, 2008.
[9] Martin Grajcar, "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, New Orleans, Louisiana, United States, Pages: 280 – 285, 1999, ISBN: 1-58133-109-7, ACM New York, NY, USA.
[10] Martin Grajcar, "Strengths and Weaknesses of Genetic List Scheduling for Heterogeneous Systems", IEEE Computer Society, Proceedings of the Second International Conference on Application of Concurrency to System Design, Page: 123, ISBN: 0-7695-1071-X, IEEE Computer Society Washington, DC, USA, 2001.
[11] Hesam Izakian, Ajith Abraham, Vaclav Snasel, "Comparison of Heuristics for scheduling Independent Tasks on Heterogeneous Distributed Environments", Proceedings of the 2009 International Joint Conference on Computational Sciences and Optimization, Volume 01, Pages: 8-12, 2009, ISBN:978-0-7695-3605-7, IEEE Computer Society Washington, DC, USA.
[12] Yi-Hsuan Lee and Cheng Chen, "A Modified Genetic Algorithm for Task Scheduling in Multiprocessor Systems", Proc. of 6th International Conference Systems and Applications, pp. 382-387, 1999.
[13] Amir Masoud Rahmani and Mojtaba Rezvani, "A Novel Genetic Algorithm for Static Task Scheduling in Distributed Systems", International Journal of Computer Theory and Engineering, Vol. 1, No. 1, April 2009, 1793-8201.
[14] Michael Rinehart, Vida Kianzad and Shuvra S. Bhattacharyya, "A modular Genetic Algorithm for Scheduling Task Graphs", Technical Report UMIACS-TR-2003-66, Institute for Advanced Computer Studies University of Maryland at College Park, June 2003.
[15] Pai-Chou Wang, W. Korfhage, "Process Scheduling with Genetic Algorithms", Proceedings of the 7th IEEE Symposium on Parallel and Distributed Processing, Page: 638, ISBN: 0-8186-7195-5, October 2005, IEEE Computer Society Washington, DC, USA.
[16] Prof. Sanjay R Sutar, Jyoti P. Sawant, Jyoti R. Jadhav, "Task Scheduling For Multiprocessor Systems Using Memetic Algorithms", http://www.comp.brad.ac.uk/het-net/tutorials/P27.pdf
[17] Andrew J. Page and Thomas J. Naughton, "Framework for Task scheduling in heterogeneous distributed computing using genetic algorithms", 15th Artificial Intelligence and Cognitive Science Conference, 2004, Castlebar, Co. Mayo, Ireland, isbn = 1-902277-89-9 pages = 137-146.
[18] Clayton S. Ferner and Robert G. Babb, "Automatic Choice of Scheduling Heuristics for Parallel/Distributed Computing", IOS Press Amsterdam, The Netherlands, Volume 7, Issue 1, Pages: 47 – 65, January 1999, ISSN:1058-9244.
[19] C.L. McCreary, A.A. Khan, J. Thompson, M.E. McArdle, "A Comparison of Heuristics for Scheduling DAGs on Multiprocessors", Eighth International Proceedings on Parallel Processing Symposium, pages: 446-451, Location: Cancun, ISBN: 0-8186-5602-6, DOI: 10.1109/IPPS.2002.288264, 06 August 2002.
[20] Javier Carretero, Fatos Xhafa, Ajith Abraham, "Genetic Algorithm Based Schedulers for Grid Computing Systems", International Journal of Innovative Computing, Information and Control, ICIC International, Vol.3, No. 6, ISSN 1349-4198, pp. 1053-1071, December 2007.

[21] Annie s. Wu, Han Yu, Shiyuan Jin, Kuo-Chi Lin, and Guy Schiavone, "An Incremental Genetic Algorithm Approach to Multiprocessor Scheduling", IEEE Transactions on Parallel and Distributed Systems, Vol.15, No. 9, On page(s): 824 – 834, ISSN: 1045-9219, INSPEC Accession Number:8094176, Digital Object Identifier: 10.1109/TPDS.2004.38, 13 September 2004.

[22] Michael Bohler, Frank Moore, Yi Pan, "Improved Multiprocessor Task Scheduling Using Genetic Algorithms", Proceedings of the Twelfth International FLAIRS Conference, WPAFB, OH 45433, American Association for Artificial Intelligence, 1999.

[23] Marin Golub, Suad Kasapovic, "Scheduling Multiprocessor Tasks with Genetic Algorithms", OACTA Press, from proceeding (351) Applied Informatics, 2002.

[24] M. Nikravan and M. H. Kashani, "A Genetic Algorithm for Process Scheduling in Distributed Operating Systems considering Load Balancing", Proceedings 21st European Conference on Modelling and Simulation Ivan Zelinka, Zuzana Oplatkova, Alessandra Orsoni, ECMS 2007, ISBN 978-0-9553018-2-7, ISBN 978-0-9553018-3-4 (CD).

[25] Shuang E Zhou, Yong Liu, Di Jiang, "A Genetic-Annealing Algorithm for Task Scheduling Based on Precedence Task Duplication", CIT, Proceedings of the Sixth IEEE International Conference on Computer and Information Technology, Page: 117, 2006, ISBN: 0-7695-2687-X, IEEE Computer Society Washington, DC, USA.