British Journal of Applied Science & Technology
10(1): 1-12, 2015, Article no.BJAST.17171
ISSN: 2231-0843

SCIENCEDOMAIN *international*
www.sciencedomain.org

SCIENCEDOMAIN

An Optimized Genetic Approach for Scheduling Task Duplication in Parallel Systems

Jasbir Singh[1*] and Gurvinder Singh[2]

[1]Department of Computer Science, Guru Gobind Singh Khalsa College, Sarhali (Tarn Taran), Punjab, India.
[2]Department of Computer Science & Engineering, Guru Nanak Dev University, Amritsar, Punjab, India.

Authors' contributions

This work was carried out in collaboration between both authors. Author JS designed the study, performed the statistical analysis, wrote the protocol, and wrote the first draft of the manuscript and managed literature searches. Authors GS managed the analyses of the study and literature searches. Both authors read and approved the final manuscript.

Article Information

DOI: 10.9734/BJAST/2015/17171
Editor(s):
(1) Xu Jianhua, Depertment of Geography, East China Normal University, China.
(2) Orlando Manuel da Costa Gomes, Professor of Economics, Lisbon Accounting and Business School (ISCAL), Lisbon Polytechnic Institute, Portugal.
Reviewers:
(1) Anonymous, Xi'an University of Science and Technology, China.
(2) Anonymous, University of Helwan, Egypt.
(3) Anonymous, Roma Tre University, Italy.
(4) Anonymous, University of Turkish Aeronautical Association, Turkey.
Complete Peer review History: http://sciencedomain.org/review-history/9802

Original Research Article

Received 1st March 2015
Accepted 1st June 2015
Published 17th June 2015

ABSTRACT

Task Scheduling deals with the set of tasks assigned to parallel multiprocessor system and the execution order of the schedule so that the total execution time is minimized. The role of a good scheduling algorithm is to efficiently assign each task to a processor depending on the resources needed, the communication overhead between related tasks is reduced and the precedence relations among tasks are satisfied. It can be efficiently used for tasks that have a large calculation, and have time constraints to complete the schedule. The efficient execution of the task scheduling on parallel system takes the structure of the task and the performance characteristics of the proposed genetic algorithm. It falls in the category of NP-complete problem. This study proposes a parallel genetic algorithm-based approach to schedule tasks on parallel system with task duplication heuristics. Task duplication can minimize inter-processor communication and hence

**Corresponding author: E-mail: jasbir2gill@gmail.com;*

results in shorter finish times. Its performance is measured in comparison with the Round Robin (RR), First Come First Serve (FCFS), and Multi-level queue scheduling (MQS), Shortest Job First (SJF), Largest Job First (LJF) and Priority scheduling methods.

Keywords: Parallel Multiprocessor Systems; Task Duplication Heuristics; Directed acyclic graph (DAG); Task Scheduling; Simultaneous Optimization; Genetic Algorithm.

1. INTRODUCTION

Task scheduling for parallel systems can become a quagmire of heuristics, models, and methods that have been developed over the past decades. The computing demand has increased exponentially in the past decades and there are many applications that require more processing power than a uni-processor can provide. To respond this demand, multi-processing units employed simultaneously to collaborate on the execution of the single application. Computer systems that employed multiple processing units are known as parallel systems [1-5]. Their aim is to speed up the execution of an application through the collaboration of the processing units. Even though the area of parallel computing has existed for many decades, programming a parallel system for the execution of a single application is still a challenging problem. Fig. 1 illustrates the process of parallelization; the application must be divided into subtasks to allow the distribution of the application's computational load among the processors.

In many practical approaches to parallel programming [6], the steps of parallelization are not clearly separable. For example the decomposition into subtasks determines the precedence constraints among them, which in turn restrict the scheduling on the processors. In other words, different decompositions into subtasks can lead to different precedence constraints and schedules with different efficiencies. Summarizing, three main areas of the parallelization process are identified:

- Subtask decomposition
- Dependence analysis
- Scheduling

Subtask decomposition and dependence analysis build the foundation for scheduling.

2. PROBLEM DEFINITION

Task scheduling [7],[8] can be defined as allocating the tasks onto parallel systems and determining the sequence of task execution on each processor. The total completion time of the schedule is determined by the performance of the parallel system and the execution sequence of the tasks.

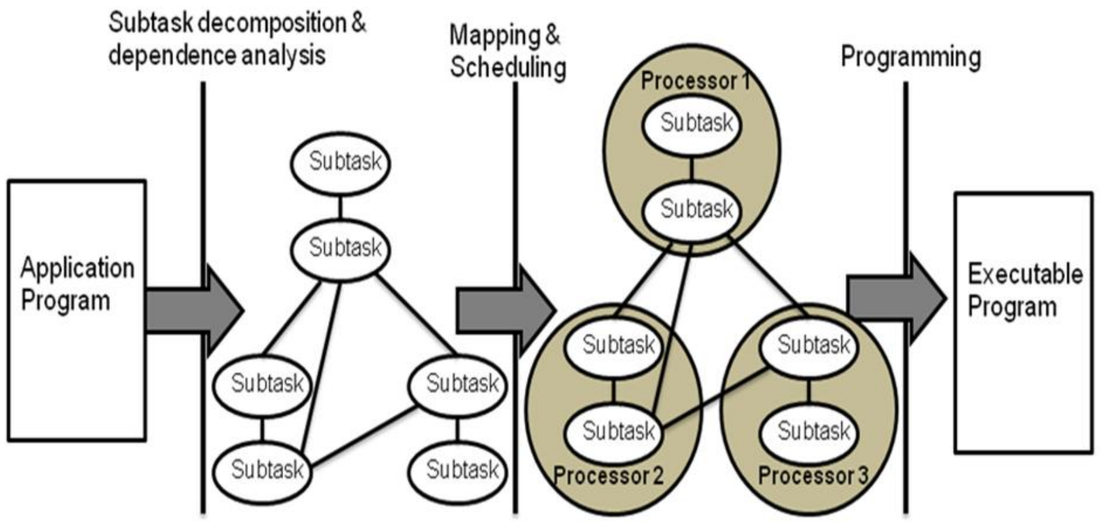

Fig. 1. Process of parallelization

Therefore, task scheduling on parallel system consists of three components:-

- Optimal mapping of the task on parallel system.
- Their sequence of execution.
- Optimal configuration of the parallel systems, in case when the parallel systems consist of heterogeneous processors, which vary in factors like processor's speed, available memory, operating system, etc.

All these three components of the simultaneous optimization [9,10] for scheduling problem are highly dependent on each other. Therefore, these should not be optimized separately.

A GA approach is used to handle the task scheduling problem on parallel system. A GA [11,12] starts with the generation of individuals. Individuals are encoded as strings called chromosomes. A chromosome refers to a solution to the given problem. Individuals are evaluated using fitness function. In general, GAs consists of crossover, selection, and mutation operations based on key parameters such as crossover probability, fitness function, and mutation probability.

Parallel Multiprocessor system scheduling [13-17] is based on the characteristics of the multiprocessor system, the tasks to be scheduled, and the information availability. The strategy behind the schedule execution on parallel multiprocessor system is the partitioning of the huge task, into set of sub-tasks of appropriate gain size. The abstract model of the partitioned tasks can be represented by Directed Acyclic Graph (DAG). The multiprocessor system consists of a set of m- parallel processor and can be represented as

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

They are connected with each other through identical links. Eight parallel systems connected with identical links, shown in Fig. 2.

The application can be represented by a DAG (directed acyclic graph) G as shown in Fig. 2(b):

$$G = (T, E, W, C)$$

Where set T is the vertices and consists of *n* tasks as:-

$$T = \{t_{j:}\ j = 1, 2 \ldots n\}$$

The set E is the directed edges consist of k edges:

$$E = \{e_{k:}\ k = 1, 2 \ldots r\}$$

This shows the precedence relationships between tasks. For example consider any two tasks t_i, t_{i+1} Є T, and having directed edge e_k (such that edge from task t_i to t_{i+1}). It means that task t_{i+1} cannot be scheduled until the task t_i has been completed. Task t_i is a predecessor of task t_{i+1} and task t_{i+1} is a successor of task t_i i.e. task t_i sends some information to task $t_{i+1,}$ whose contents are needed by t_{i+1} to start execution.

The set W be the weights of the vertices:

$$W = \{w_{i,j:}\ i = 1, 2 \ldots m, \text{ and } j: 1, 2 \ldots n\}$$

This shows the execution time of the i^{th} task on j^{th} processor, and are varies from processor to processor because of the processor environment is heterogeneous.

The following Table 1 shows a matrix of the sample space for 140 tasks, executed on 12 parallel multiprocessor systems, here task t_1 shows execution time of 4 time units for processor p_1, 5 time units for processor p_3 and 6 time units for processor p_4 and so on. This is because, processors are of heterogeneous nature and have different processing capabilities and speed.

The set C be the communication weight of the edges:

$$C = \{c_{k:}\ k = 1, 2 \ldots r\}$$

This shows the data communication weights between the two tasks, if the tasks are scheduled on different Processors. When both tasks are scheduled on the same processor, the weight associated to the edge become null.

3. OBJECTIVE AND SCOPE

The objective of this paper is to design an optimization algorithm, which minimize the overall execution time of the schedule. The proposed genetic algorithm-based scheduling technique generates a task graph [18] using set of computing resources or processors and maps the sub-tasks onto the available parallel processors, and also orders (or sequence) the execution of the tasks so that:

- The task precedence constraints are satisfied.
- The resource constraints are satisfied.
- A minimum schedule length is achieved.

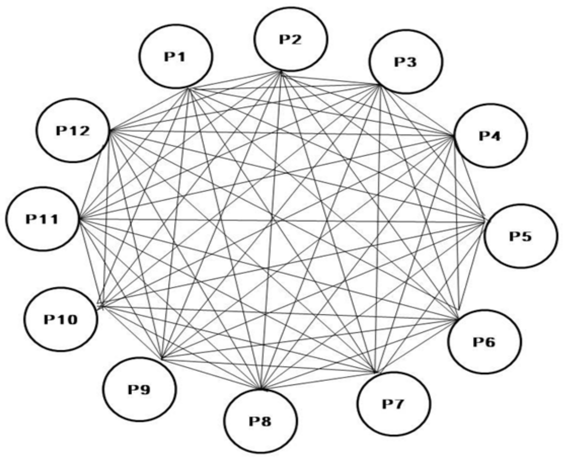

Fig. 2. Fully connected 12 parallel systems

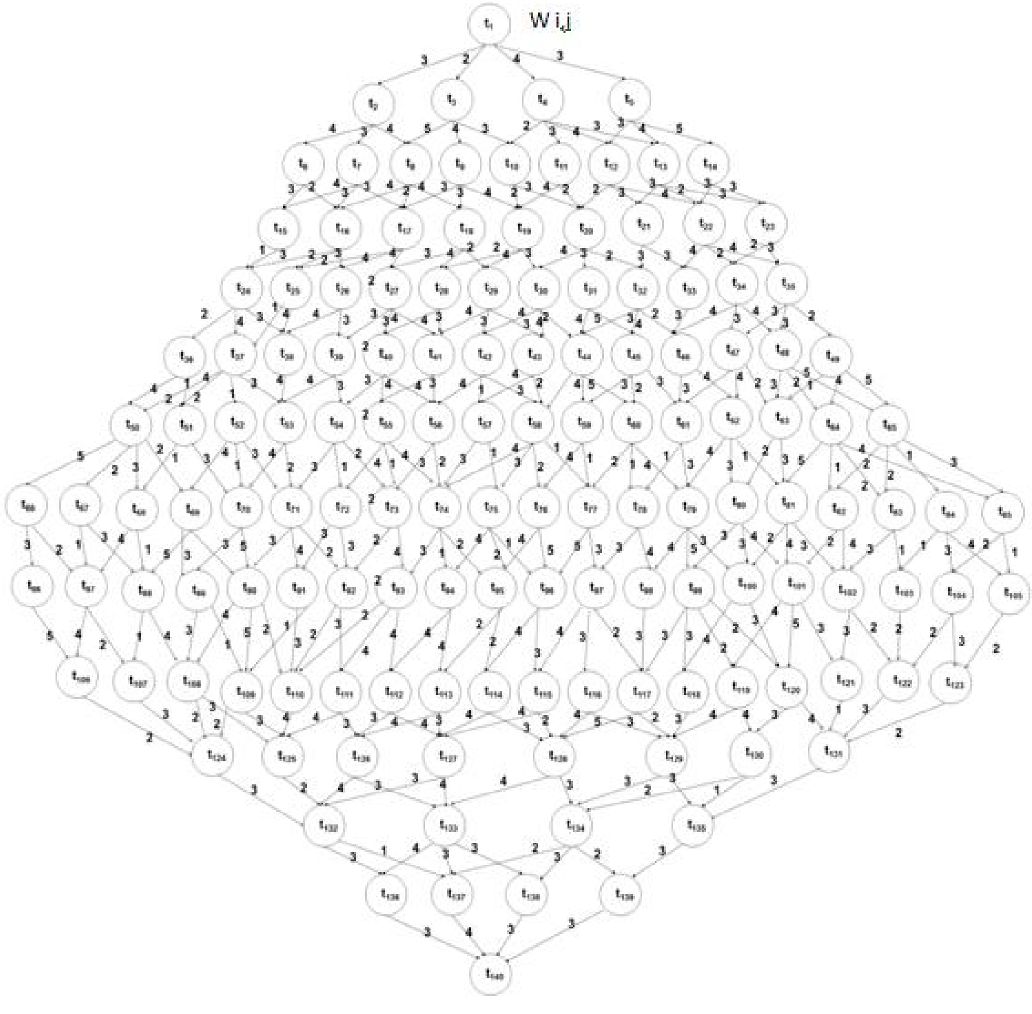

Fig. 2(b). A directed acyclic graph with task precedence. Where wi, j = execution time of different tasks on different processors as shown in Table 1

Table 1. Shows a tasks execution matrix on different processors with task size is 140 tasks

	t_{1}	t_{2}	t_{3}	t_{4}	t_{5}	t_{6}	t_{7}	t_{8}	t_{9}	t_{10}	t_{11}	t_{12}	t_{13}	t_{14}	t_{15}	t_{16}	t_{17}	t_{18}	t_{19}	t_{20}	t_{21}	t_{22}	t_{23}	t_{24}	t_{25}	t_{26}	t_{27}	t_{28}	t_{29}	t_{30}	t_{31}	t_{32}	t_{33}	t_{34}	t_{35}
p_{1}	4	3	6	5	2	8	7	4	9	10	11	6	12	13	10	16	10	7	6	3	9	12	13	15	7	10	6	8	4	5	6	7	10	13	9
p_{2}	4	4	7	6	3	9	8	5	10	11	12	6	13	13	11	17	10	8	7	4	10	13	13	15	8	10	6	9	5	6	6	8	11	14	10
p_{3}	5	5	7	7	3	9	9	5	10	11	12	7	15	14	12	18	11	8	8	5	10	13	14	17	8	11	7	9	6	6	7	8	12	14	10
p_{4}	6	5	8	7	4	10	9	6	11	12	14	8	16	15	13	18	11	9	8	5	11	14	14	17	9	12	8	10	7	7	8	9	14	16	11
p_{5}	7	6	9	8	4	11	10	6	11	13	14	9	17	16	14	20	12	10	9	6	12	15	15	18	10	13	9	11	7	8	9	10	14	17	12
p_{6}	7	7	10	8	5	11	10	7	12	14	15	10	19	18	15	20	13	11	10	6	13	16	15	18	11	14	10	12	8	8	10	11	15	18	13
p_{7}	8	7	10	9	5	12	11	8	13	14	16	10	19	19	15	22	14	12	10	7	14	16	16	20	12	15	10	13	8	9	11	12	16	19	14
p_{8}	9	8	11	10	6	12	11	9	13	15	16	11	20	19	16	22	14	13	11	7	14	17	17	20	13	15	11	14	9	10	12	13	17	20	15
p_{9}	9	8	12	10	6	13	12	9	14	15	17	12	21	20	17	23	15	14	12	8	15	17	18	21	14	16	12	14	10	11	12	13	17	21	16
p_{10}	10	9	12	11	7	13	13	9	15	16	18	12	22	21	17	24	16	15	12	9	16	18	18	22	14	17	12	15	10	11	13	14	18	21	16
p_{11}	10	9	12	11	7	14	13	9	15	16	18	13	22	22	18	24	16	15	13	9	16	18	18	22	15	17	13	15	11	11	14	14	18	21	17
p_{12}	11	9	13	12	8	14	14	9	16	17	18	13	23	22	18	25	17	16	13	9	17	19	19	23	15	18	13	16	12	12	14	15	19	22	18

Continue…………

	t_{36}	t_{37}	t_{38}	t_{39}	t_{40}	t_{41}	t_{42}	t_{43}	t_{44}	t_{45}	t_{46}	t_{47}	t_{48}	t_{49}	t_{50}	t_{51}	t_{52}	t_{53}	t_{54}	t_{55}	t_{56}	t_{57}	t_{58}	t_{59}	t_{60}	t_{61}	t_{62}	t_{63}	t_{64}	t_{65}	t_{66}	t_{67}	t_{68}	t_{69}	t_{70}
p_{1}	12	10	4	3	8	11	9	12	2	14	13	9	7	10	9	6	15	18	19	20	7	10	13	12	18	8	11	17	15	6	27	23	16	13	27
p_{2}	12	10	5	4	9	12	10	13	2	14	13	10	8	11	10	6	15	18	19	20	7	10	13	12	18	8	11	17	16	6	27	23	16	13	27
p_{3}	13	11	5	4	9	12	11	13	3	16	14	11	9	11	11	7	16	19	20	21	8	11	14	13	19	9	12	18	16	6	28	24	17	14	28
p_{4}	15	12	6	5	10	13	12	14	3	16	14	11	10	12	12	8	16	19	21	21	8	11	15	14	19	9	12	18	16	7	28	24	17	14	28
p_{5}	15	13	6	5	10	15	13	15	4	17	16	13	11	13	13	8	17	20	22	22	9	12	16	15	20	9	13	18	17	7	29	25	18	15	29
p_{6}	16	13	7	6	11	15	14	15	4	17	16	14	11	14	13	8	18	20	23	23	9	12	16	15	20	10	13	19	17	7	29	25	18	15	29
p_{7}	17	14	8	6	12	16	15	16	5	18	17	15	12	15	14	9	1	21	24	24	9	13	16	15	21	10	14	19	18	7	30	25	19	15	29
p_{8}	18	15	8	7	13	16	16	17	5	19	18	16	13	15	15	9	19	22	24	25	10	14	17	16	21	11	14	20	19	8	30	26	19	16	30
p_{9}	18	16	9	7	14	17	16	18	6	20	18	17	14	16	15	10	19	23	25	26	10	15	17	16	22	12	15	20	19	8	31	26	20	16	30
p_{10}	19	16	9	8	14	18	17	18	6	21	19	17	15	16	16	10	20	23	25	26	10	15	18	17	23	12	15	21	20	9	31	27	20	17	31
p_{11}	19	17	9	8	15	18	17	19	7	21	20	18	15	16	17	11	20	24	26	27	11	16	18	17	23	13	16	21	20	9	32	28	21	18	31
p_{12}	20	17	9	9	16	18	18	20	8	22	21	19	16	17	18	11	21	25	26	28	11	17	19	18	24	14	17	22	20	10	33	29	22	18	32

Continue…………

	t_{71}	t_{72}	t_{73}	t_{74}	t_{75}	t_{76}	t_{77}	t_{78}	t_{79}	t_{80}	t_{81}	t_{82}	t_{83}	t_{84}	t_{85}	t_{86}	t_{87}	t_{88}	t_{89}	t_{90}	t_{91}	t_{92}	t_{93}	t_{94}	t_{95}	t_{96}	t_{97}	t_{98}	t_{99}	t_{100}	t_{101}	t_{102}	t_{103}	t_{104}	t_{105}
p_1	5	10	3	6	14	2	5	8	20	18	19	14	11	7	12	20	6	10	9	4	13	3	8	15	3	12	14	9	6	12	9	5	4	15	8
p_2	5	10	3	7	14	2	6	8	20	18	20	15	12	7	12	21	6	11	9	4	13	3	8	15	3	12	15	9	6	13	10	6	4	15	8
p_3	6	11	4	7	15	2	6	9	21	19	20	16	12	8	13	22	7	11	10	5	14	3	9	16	4	13	15	10	7	13	10	6	4	16	9
p_4	7	12	4	8	16	3	7	9	22	19	21	16	13	8	14	23	7	12	10	5	14	4	9	16	4	13	15	10	7	13	11	7	5	17	9
p_5	8	12	4	9	17	3	7	10	23	20	21	17	14	9	15	24	8	13	11	6	15	4	9	17	4	14	16	10	7	14	11	7	5	17	9
p_6	9	13	5	9	17	4	8	11	23	21	22	18	15	10	16	25	8	14	12	6	16	4	10	17	5	15	16	11	8	14	12	7	5	18	10
p_7	9	14	5	9	18	4	8	11	24	21	23	18	15	10	16	25	9	15	13	6	17	4	10	18	5	16	17	11	8	15	12	8	6	18	10
p_8	10	14	5	10	18	5	9	12	25	22	24	19	16	11	17	26	10	16	14	7	18	5	10	19	5	17	18	11	8	16	12	8	6	19	11
p_9	10	15	6	10	19	5	10	13	26	23	25	19	16	12	18	27	11	17	14	7	19	5	11	19	6	17	18	12	9	17	13	9	6	19	11
p_{10}	11	16	6	10	20	5	11	13	27	23	26	20	17	13	19	30	11	18	15	7	20	5	12	20	6	18	19	13	9	18	14	9	7	20	12
p_{11}	12	17	7	11	20	6	11	14	27	24	26	20	18	14	20	31	12	18	16	8	21	6	12	21	7	18	20	13	9	18	15	9	8	21	12
p_{12}	12	17	8	12	21	7	12	15	28	25	27	21	19	15	21	31	12	19	16	9	22	7	13	21	8	19	20	14	9	19	16	9	8	21	13

Continue…………

	t_{106}	t_{107}	t_{108}	t_{109}	t_{110}	t_{101}	t_{112}	t_{113}	t_{114}	t_{115}	t_{116}	t_{117}	t_{118}	t_{119}	t_{120}	t_{121}	t_{122}	t_{123}	t_{124}	t_{125}	t_{126}	t_{127}	t_{128}	t_{129}	t_{130}	t_{131}	t_{132}	t_{133}	t_{134}	t_{135}	t_{136}	t_{137}	t_{138}	t_{139}	t_{140}
p_1	22	20	17	15	9	18	7	6	21	16	18	17	15	17	19	17	18	9	16	11	8	2	15	10	21	10	18	15	15	10	6	3	10	10	20
p_2	22	21	17	16	9	19	7	6	21	16	18	17	15	18	20	18	19	10	17	12	8	2	15	10	22	10	19	15	16	11	6	4	11	10	20
p_3	23	22	18	16	10	19	7	7	22	17	19	18	16	18	20	18	19	11	17	12	8	2	16	11	22	11	19	16	16	11	7	4	11	11	21
p_4	24	22	18	17	10	20	8	7	23	18	19	18	17	19	21	19	20	11	18	12	9	3	16	11	23	11	20	16	16	12	8	5	12	12	22
p_5	25	23	18	17	11	20	8	7	24	19	20	19	18	19	21	19	20	12	19	13	10	3	17	12	24	12	20	16	17	13	8	6	13	13	22
p_6	26	23	19	18	11	21	8	8	25	20	21	19	19	20	22	19	21	13	20	1	10	3	17	12	25	12	21	17	17	14	8	6	14	14	23
p_7	27	24	19	18	12	21	9	8	26	21	21	20	19	20	23	20	21	13	20	14	11	4	18	13	25	13	21	17	18	15	9	6	15	14	24
p_8	28	24	20	19	13	22	9	8	26	21	22	21	20	21	24	21	21	14	21	14	12	4	18	13	26	14	22	18	19	16	9	7	16	15	25
p_9	29	25	21	20	13	23	10	9	27	22	23	21	20	21	24	22	22	15	22	15	13	4	18	14	27	15	23	19	19	17	10	7	17	16	26
p_{10}	30	26	21	21	14	23	10	10	28	23	24	22	21	22	25	22	22	15	23	16	14	5	19	15	28	15	24	19	20	18	10	8	18	17	27
p_{11}	30	26	22	22	14	24	10	11	28	24	25	23	22	23	25	23	23	16	24	16	15	6	19	15	29	16	24	20	20	19	11	8	19	17	28
p_{12}	30	27	23	23	15	25	11	11	29	25	25	23	22	23	26	23	23	17	24	17	16	6	20	15	30	16	25	21	21	19	11	9	19	18	29

While considering the above constraints, the design of the scheduler depends upon the following issues:-

- The scheduling algorithm can be deterministic or nondeterministic. In a nondeterministic problem, all or some of factors like knowledge about the task, relation between tasks (dependent or non-dependent), number of processors etc. can be input-dependent and change according to run time conditions. Where as in deterministic task scheduling problem, knowledge related to tasks, relations among each other, timing and the availability of the number of processors are all a-prior knowledge.
- The tasks can be either preemptive or non-preemptive. In a non-preemptive task scheduling problem, task must be completely executed before another task to be scheduled. Where as in a preemptive task scheduling problem, the tasks to be cut off from execution and another task to start its execution cycle.
- The processors either are homogenous or heterogeneous. In a homogenous processor environment, all processors are assumed to have equal processing capabilities. Where as in heterogeneity of processors, all processors have different processing capabilities or speeds.

Proposed Genetic Algorithm is an efficient scheduling of a parallel task onto the parallel heterogeneous system that minimizes the overall execution time and achieve a high performance.

4. SIGNIFICANCE

Parallel computing has made a tremendous impact on a variety of areas ranging from computational simulations for scientific and engineering applications to commercial applications. The cost benefits of parallelism coupled with the performance requirements of applications present compelling arguments in favor of parallel computing. Parallel computers have been used to solve a variety of discrete and continue optimization problems. Algorithms such as Simplex, Interior Point Method for linear optimization and branch and bound and genetic programming for discrete optimization have been efficiently parallelized and are frequently used.

5. GENETIC ALGORITHM

A Genetic Algorithm is an optimization search technique and is a part of a class of evolutionary algorithms (EA) which use the mechanism and concept inspired by evolutionary biology such as selection, mutation, inheritance, and crossover. It requires an abstract representation (known as genotype) of candidate solutions (known as phenotypes). A fitness function evaluates how well each solution satisfies the search or optimization objectives. It starts with a group of randomly generated solutions; recombines results of existing solutions (known as crossover) and makes random changes (known as mutation). GA generates a new solution using fitness function and the evolutionary operators, which are increasingly closer to an optimal solution.

6. METHODOLOGY

The nature of the scheduling problem depends on the type of tasks to be scheduled, on the platform architecture, and on the aim of the scheduling policy. The tasks may be independent (e.g., they represent jobs submitted by different users to a same system, or they represent occurrences of the same program run on independent inputs), or the tasks may be dependent (e.g., they represent the different phases of a same processing and they form a task graph). The platform may or may not have a hierarchical architecture (clusters of clusters vs. a single cluster); it may or may not be dedicated. Resources may be added to or may disappear from the platform at any time, or the platform may have a stable composition. The processing units may have the same characteristics (e.g., computational power, amount of memory, multi-port or only single-port communications support, etc.) or not. The communication links may have the same characteristics (e.g., bandwidths, latency, routing policy, etc.) or not. The aim of the scheduling policy can be to minimize the overall finish time (makespan minimization), the throughput of processed tasks, etc. Finally, the set of all tasks to be scheduled may be known from the beginning, or new tasks may arrive all along the execution of the system.

Task scheduling on parallel multiprocessor system can be categorized into different classes based on the characteristics and type of the tasks to be scheduled, type of the multiprocessor systems i.e. either homogeneous or heterogeneous [19-21] and the availability of information. The focus is on the deterministic scheduling problem in which all information about the tasks such as precedence relation and execution time are known to the scheduler in advance, the application tasks should be non preemptive i.e. the execution of the task must be completely done before another application task takes control of the processor, type of the processor is heterogeneous i.e. processors have different processing capabilities and speed.

This study will try to assess the impact of the heterogeneity and volatility of the resources onto the scheduling strategies. When sets of computational tasks are scheduled, the traditional objective is to minimize the overall finish time also called the makespan.

The main objective of the proposed scheduling algorithm is to minimize the total schedule finish time of the tasks which includes execution time and waiting time or idle time on deterministic and non-preemptive tasks in a heterogeneous multiprocessor environment.

The procedure of the proposed genetic algorithm is:

```
Step 1: Setting the parameter
        Read directed acyclic graph i.e. task execution matrix which include number
        of tasks n, number of processors m and communication cost c, population size,
        probability $p_m$, mutation, crossover probability $p_c$, and maximum generation
        max_gen.
        Set generation gen = 0, maxeval = 0
Step 2: Initialization
        Generate randomly population size chromosomes.
Step 3: Evaluate
        Step 3.1: Evaluate the fitness function of each chromosomes
        Step 3.2: Evaluate task fitness
        Step 3.2: Evaluate processor fitness
Step 4: Compute
        Perform crossover operation on the chromosomes, having probability $p_c$.
Step 5: Compute
        Perform mutation operation on chromosomes, having probability $p_m$.
Step 6: Selection
        Select population size chromosomes from offspring for the next generation
        and parents.
Step 7: Check Condition
        Test: If gen = max_gen, then output best-solution and stop.
        Else
        Compute: gen = gen + 1 and return to step 3
```

The problem of optimal scheduling of the tasks on parallel multiprocessor systems with m processors is to allocate all the computational tasks to the available processors, so that the precedence relations among the tasks are maintained and all the tasks are executed in shortest possible time.

7. PROPOSED GENETIC ALGORITHM FOR THE OPTIMAL SOLUTION

Genetic Algorithm operates through a simple cycle of stages:

- Create population string
- Evaluate each string
- Select the best string
- Reproduction to create a new population.

The individuals are encoded in the population string known as chromosomes. Once the chromosome has been coded, it is possible to evaluate the performance or fitness of individuals in a population. A good coding scheme [22],[23] will benefit operators and make the object function easy to calculate. During selection, each individual is assigned a fitness value given by the objective function and choose the fittest individual of the current population to serve as parent of the next generation. Reproduction involves two types of operators namely crossover & mutation. The crossover operator chooses randomly a pair of individuals among those selected previously and exchange some part of the information. The mutation operator takes an individual randomly and alters it.

7.1 Population Initialization

The first step in the genetic algorithm is to create initial population, number of processors; number of tasks and population size is needed to generate initial population. The initial population is initialized with randomly generated individuals whereas the length of the individuals is equal to the number of application tasks in the directed acyclic graph. Each task is randomly assigned to a processor.

7.2 Fitness Evaluation

The fitness function used for proposed parallel genetic algorithm is based on the total finishing time of the schedule, which includes execution time and communication delay time. The fitness function separates the evaluation into two parts:

7.2.1 Task fitness

The task fitness focus that all tasks are performed and scheduled in a valid order. A valid order means that pair of tasks is independent and no any task get data output from the other task for execution. The task scheduling of a pair of tasks to a single processor is valid, if the pair is independent or the order in which they are assigned to the processor matches the order of their dependency.

7.2.2 Processor fitness

The processor fitness component of the fitness function attempts to minimize processing time. Let us consider the following scheduler S1 & S2 for single processor and multiprocessor tasks schedule respectively (here, we consider the case when fitness function assigned all tasks to a single processor and randomly generated tasks to heterogeneous parallel system.) The processor chosen for schedule S1 is p_1 and the processors chosen for scheduler S2 are same as given in Table 1 and proposed genetic algorithm uses node duplication heuristics to reduce the data response time of its descendant nodes and permitting them to start as earlier as possible. The total finish time of schedule S1 & S2 is

S1: $t_1 \rightarrow t_2 \rightarrow t_3$ ----------------------------$\rightarrow t_{139} \rightarrow t_{140}$

Total Finish Time = Execution time + Comm. Time = 4+3+6+5+2+8+7+4+9+10+11+6+12+13+10+16+10+7+6+3+9+12+13+15+7+10+6+8+4+5+6+7+10+13+9+12+10+4+3+8+11+9+12+2+14+13+9+7+10+9+6+15+18+19+20+7+10+13+12+18+8+11+17+15+6+27+23+16+13+27+5+10+3+6+14+2+5+8+20+18+19+14+11+7+12+20+6+10+9+4+13+3+8+15+3+12+14+9+6+12+9+5+4+15+8+22+20+17+15+9+18+7+6+21+16+18+17+15+17+19+17+18+9+16+11+8+2+15+10+21+10+18+15+15+10+6+3+10+10+20 = 1540 time units

Here communication time = 0, because all tasks are executed on processor p1.

S2: Total finish time = Execution time + Communication Time = 319 time units.

The scheduler S1 shows a total finish time of 1540 time units, where scheduler S2 shows a total finish time of just 319 time units. Hence proper fitness function reduces the total finish time very well.

Thus, the fitness values (task & processor fitness) have been evaluated for all chromosomes and the probability of higher fitness is to be selected for reproduction from current generation to the next generation.

7.3 Selection Operator

The fitness function is the base of selection operator. The performance of genetic algorithm is affect by the design of the fitness function. Selection operator selects the superior and eliminates the inferior. Individual are selected as per their fitness value. When the fitness values have been computed for all chromosomes, we can select good chromosomes through rotating roulette wheel strategy. The selection operator produces next generation through selecting the best chromosomes from offspring and parents.

7.4 Crossover Operator

It randomly selects two parent chromosomes, with higher values of chromosomes have more chance for the selection and randomly choose their crossover points, and mate them to produce two child (known as offspring) chromosomes. We examine one and two point crossover operators. In one point crossover operation, the right segments of the crossover points are swapped to produce two offspring as shown in Fig. 3 (a) and in two point crossover [24,25], the central portions of the crossover points are swapped to produce two offspring as shown in Fig. 3 (b).

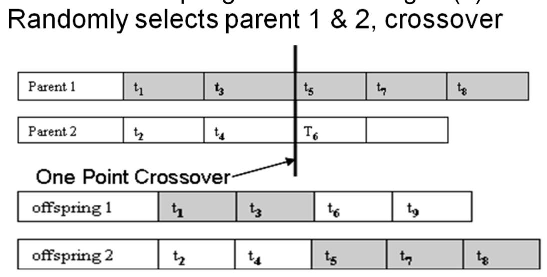

Fig. 3 (a). One point crossover

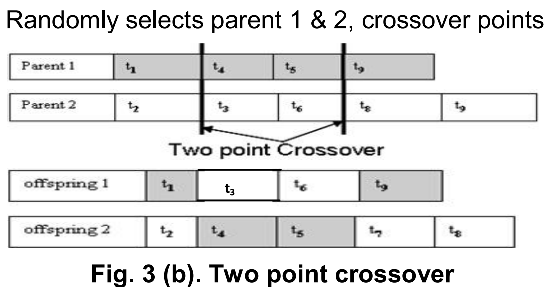

Fig. 3 (b). Two point crossover

7.4 Mutation Operator

A mutation operation is designed to reduce the idle time of a processor waiting for the data from other processors. It works by randomly selecting two tasks and swapping them. Firstly, it randomly selects a processor, and then randomly selects a task [26,27] on that processor. This task is the first task of the pair to be swapped. Secondly, it randomly selects a second processor (it may be the same as the first), and randomly selects a task. If the two selected tasks are the same task the search continues on. If the two tasks are different then they are swapped over (provided that the precedence relations must satisfy). Consider the following example of six tasks DAG with tasks precedence and the execution times of tasks t_1 to t_6 on processor p_1 and p_2 are given in Table 1. Fig. 4(a), (b), (c) and (d) demonstrates the mutation operation.

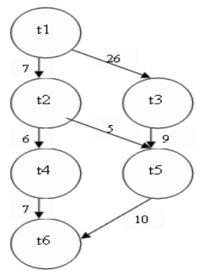

Fig. 4 (a). A DAG with tasks precedence

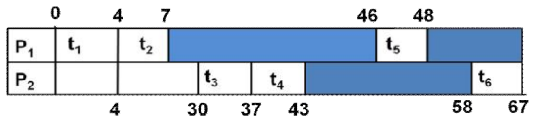

Fig. 4 (b). A Gantt chart before mutation operation, which takes 67 time units to complete the schedule

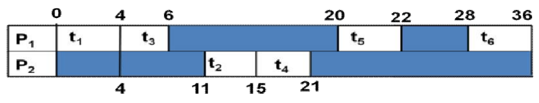

Fig. 4 (c). A Gantt chart after mutation operation, which takes 36 time units to complete the schedule

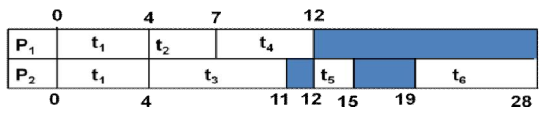

Fig. 4 (d). A Gantt chart after mutation operation with task duplication, which takes 28 time units

Here the mutation operation swaps application task t_2 on processor p_1 to application task t_3 on processor p_2.

8. EXPERIMENTAL RESULTS AND PERFORMANCE ANALYSIS

The simulation of the proposed algorithm was performed on simulation environment that was developed using MATLAB programming environment. To find the best solution, we use different values of population (50 to 500 generations), crossover rate (0.1 to 0.5) and mutation rate (0.01 to 0.03) and varying task size. Simulation results using genetic algorithm are based on the test runs and study the effects of changing these parameters. We have studied the effect of genetic based approach for scheduling tasks on parallel multiprocessor systems that can minimize the total finish time compared to the other traditional approaches. In this experiment, we have shown a sample (In this case of 140 task size) of task execution time on different processor (In this case 12 processor) in Table 1. Fig. 5, 6 and 7 shows the time taken to complete the schedule as well performance and efficiency of the proposed GA with other algorithms.

The final best application-task schedule is obtained by applying the Genetic Algorithm (GA) to the directed acyclic graph with execution time shown in the above Table 1 onto the twelve parallel multiprocessor systems is shown in Fig. 6 for task size is 140 tasks. We also compare the results with Round Robin (RR), First Come First Serve (FCFS), and Multi-level queue scheduling (MQS), Shortest Job First (SJF), Largest Job First (LJF) and Priority scheduling methods on parallel systems.

8.1 Performance Analysis

The performance of the proposed parallel genetic algorithm is carried out by evaluating the speedups and efficiency, compared with the with Round Robin (RR), First Come First Serve (FCFS), Multi-level queue scheduling (MQS), Shortest Job First (SJF), Largest Job First (LJF) and Priority scheduling algorithms.

8.1.1 Speed up (T_{SP})

Speed up [28] is defined as the finish time on a single processor divided by finish time on a multiprocessor system. It is denoted as T_{sp} in case of homogeneous multiprocessor system as:

$T_{sp} = p(1)/p(m)$

But, in case of heterogeneous multiprocessor system. It will be:

$T_{sp} = (\min (p(1)) / p(m)$

That is the best single processor finish time divided by the finish time on a heterogeneous multiprocessor system. The speedup shows the finish time of 1540 units of the 140 tasks assigned on single processor, divided by finish time units on GA, Round Robin (RR), First Come First Serve (FCFS), Multi-level queue scheduling (MQS), Shortest Job First (SJF), Largest Job First (LJF) and Priority scheduling algorithms as shown in Fig. 6.

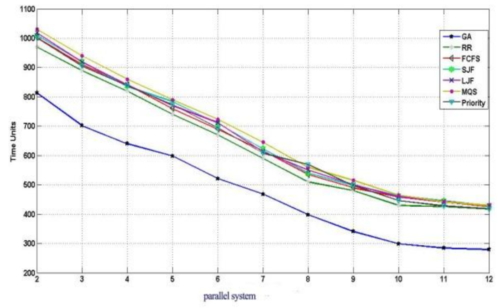

Fig. 5. Time taken to complete the schedule

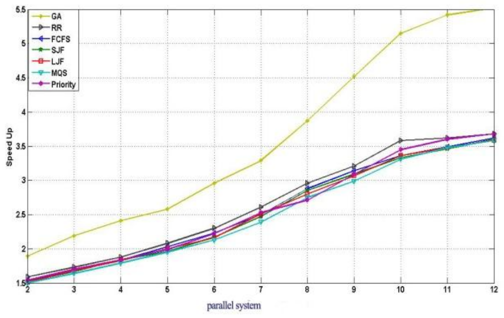

Fig. 6. Speedup v/s number of parallel multiprocessor system

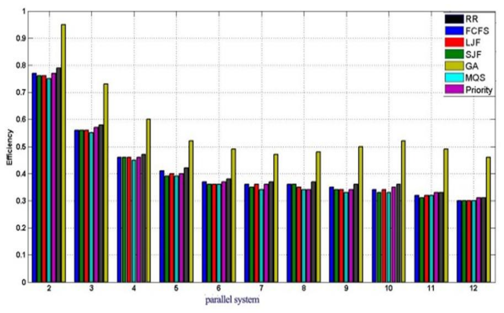

Fig. 7. Performance comparisons

8.1.2 Efficiency (¢) (T_{sp} / m)

During the past 30 years, the trend indicates that ever faster networks, multi-processor computer architectures, and distributed systems, clearly show that parallelism is the future of computing. In this paper, we give successful examples of this approach i.e. successful examples correspond to problems where optimal throughput is determined and reconstructing the final schedule.

We have considered scheduling techniques that maps a set of application graphs onto parallel multiprocessor platforms. We have shown that the general instance of this problem is NP-complete. However, in many situations, it is amenable to be optimal.

9. CONCLUSION

In this study we have proposed a genetic algorithm for task scheduling in heterogeneous parallel multiprocessor systems with task duplication heuristics to minimize the finish time of the schedule including waiting or idle time and execution time, increase the throughput of the system and method found a better solution for assigning the tasks to the parallel multiprocessor system. Experimental results and performance analysis of the proposed genetic algorithm is compared with Round Robin (RR), First Come First Serve (FCFS), Multi-level queue scheduling (MQS), Shortest Job First (SJF), Largest Job First (LJF) and Priority scheduling algorithms methods. The results are based on the best randomly generated schedule by the proposed algorithm.

COMPETING INTERESTS

Authors have declared that no competing interests exist

REFERENCES

1. Qumn MJ. Parallel computing theory and practices. 2nd ed. Tata McGraw Hill Education Private Ltd. 2002;346-364.
2. David A Pattern, Hennessy JL. Computer Architecture 3rd ed. Morgan Kaufmann & Elsevier India. 2011;528-590.
3. Culler DE. Parallel Computer Architecture. Morgan Kaufmann & Elsevier India; 2010.

4. Hayes JP. Computer Architecture and Organization. McGraw Hill International Edition; 1996.
5. Carpinalli JD. Computer System Organization & Architecture. Pearson Education; 2006.
6. Sara Baase, Allen Van Gelder. Computer Algorithms. Addison Wesley; 2000.
7. Sartaj Sahni. Algorithms Analysis and Design. Galgotia Publications Pvt. Ltd., New Delhi; 1996.
8. George Karypis, Ananth Grama, Vipin Kumar, Anshul Gupta. Introduction to parallel computing. Pearson Education; 2009.
9. Kalyanmoy Deb. Optimization for engineering design. PHI; 2003.
10. Terence Fountain, Peter Kacsuk, Dezso Sima. Advanced computer architectures. Pearson Education; 2009.
11. Goldberg DE. Genetic algorithms in search, optimization and machine learning. Pearson Education. 2004;60-83.
12. Ishraq Ahmad, Yu-Kwong. Static scheduling algorithms for allocating directed task graphs to multiprocessors. ACM Computing Surveys. 1999;31(4).
13. Tack-Don Han, Sung-Bong Yang, Sung-Ho Woo, Shin-Dug Kim. Task scheduling in distributed computed system with a genetic algorithm. IEEE Trans on parallel System. 1997;301-305.
14. Ahmadabadi N, Fakhraie SN, Salmani Jelodar M, Fakharie SM. A representaion for genetic–algorithm-based multiprocessor task scheduling. IEEE Proceeding. 2006;340-347.
15. Arif Ghafoor, Muhammad K Dhodhi, Imtiaz Ahmad. Task assignment in distributed computing systems. IEEE Proceeding. 1995;49-53.
16. Fikret Ercan M, Ceyda Oguz. A genetic algorithm for multi-layer multiprocessor task scheduling. IEEE Proceeding. 2004;168-170.
17. Heng-Nlan QI, Jian-Gang Yang, YI-Wen Zhong. A hybrid genetic algorithm for task scheduling in heterogeneous computing system. IEEE Proceeding. 2004;2463-2468.
18. Mitchell, Melanie. An introduction to genetic algorithm Bu. MIT Press; 1996.
19. page J, Naughton TJ. Framework for task scheduling in heterogeneous distributed computing system using genetic algorithm. Kluwer Academic Publisher, Printed in the Netherlands; 2005.
20. Chi-Kwong Li, Wai-Yip Cha. Proceeding IEEE scheduling tasks in DAG to heterogeneous processor system; 1998.
21. Wojciech Cencek. High-Performance computing on heterogeneous systems; Computational Methods in Science and Technology; 1999.
22. Arjan JC. van Gemund, Andrei R. Fast and effective task scheduling in heterogeneous systems. IEEE Proceeding; 2000.
23. Frank Moore, Pan Yi, Michael Bohler. Improved multiprocessor task scheduling using genetic algorithms. Proceedings of the Twelfth International FLAIRS Conference; 1999.
24. Laurence tianruo Yang, Man Lin. Hybrid genetic algorithm for scheduling partially ordered tasks in a multi-processor environment. IEEE Proceeding. 1999;382-387.
25. Maode Ma, Rongbo Zhu, Yajun Li, Yuhang yang. A Problem-Specific genetic algorithm for multiprocessor real-time task scheduling. IEEE Proceeding; 2008.
26. Hong Jiang, Jameela Al-Jaroodi, David Swanson, Nader Mohamed. Modeling parallel applications performance on heterogeneous systems; Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'03); 2003.
27. M. J Qumn. Parallel Programming. Tata McGraw Hill Education Private Ltd; 63-89.
28. Faye A Briggs, Kai Hwang. Computer architecture and parallel processing. McGraw Hill. 1985;445-47,612.