IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661, p- ISSN: 2278-8727Volume 13, Issue 4 (Jul. - Aug. 2013), PP 20-29
www.iosrjournals.org

Genetic Approach to Parallel Scheduling

Prashant Sharma[1], Gurvinder Singh[2]
[1]*(M. Tech, Computer Science and Engg. Guru Nanak Dev University, India)*
[2]*(Professor, HOD, Computer Science and Engg, Guru Nanak Dev University, India)*

Abstract: *Task Scheduling is Essential part for proper functioning of parallel processing system. Several approaches have been applied to solve this problem. Genetic algorithms have received much awareness as they are robust and guarantee for a good solution. In this paper a genetic algorithm is developed and implemented and performance of algorithm is measured under altering parameters. Adaptive parameter approach has been applied to enhance the performance of the genetic algorithm.*
Keywords: *Genetic Algorithm, Fitness Function, DAG, Crossover and Mutation.*

I. Introduction

Parallel Computing is the simultaneous use of multiple resources to solve a computational problem. Different approaches have been used to increase the performance, by upgrading the hardware, building more powerful hardware requires an intensive and extensive production process, but that has reached certain physical limits. Dividing the program into tasks to increase the performance has also reached the limits because a program can only be divided into certain parts. Parallel scheduling brings a smart way of increasing the performance by having an algorithm which helps in minimizing the completion time of a parallel application by properly allocation tasks to the processors and sequencing the execution of tasks.
Multiprocessing scheduling is a NP complete problem which has received a large amount of attention and is considered one of the challenging problems in parallel computing. There are many parallel computing architectures available such as shared memory architectures, message passing architectures etc. To effectively harness the computing power of all these parallel architectures, optimized scheduling of parallel program tasks to the processors is crucial. Indeed, if the tasks are not scheduled properly, the parallel program is likely to produce a poor speedup due to communication overheads which makes parallelization useless.

1.1 Genetic Algorithms

Evolutionary algorithms are optimization and search procedures inspired by genetics and process of natural selection. Basic Idea of genetic algorithms is to evolve a population of candidate solutions using the concept of survival of the fittest, variation and inheritance. The genetic algorithms behave much like biological genetics. The algorithm begins by initializing a population of individuals. Individual solutions are selected from the population, and then mate to form new solutions. The mating process, typically implemented by combining, or crossing over, genetic material from two parents to form genetic material for one or two new solutions, represents data from one generation to another.

The genetic algorithm includes some representation-specific and some representation-neutral operators. The operators like initialization, crossover and mutation are representation-specific operators. Selection, replacement and termination operators are all independent of the representation.

The traditional schedule optimization methods operate in a problem space; they require information about the schedule in order to get the good schedules. The genetic algorithms work on representation space. They care only about the structure of the solution. The performance of each solution is the only information genetic algorithms need to guide its search[1].According to evolutionary theories, only the most suited elements in a population are likely to survive and generate offspring thus transmitting their biological heredity to new generations[3]. The following table represents the mapping to genetic algorithms to the nature.

TABLE 1: Computer Nature Mapping for GA

Nature	Computer
Individual	Solution to a problem
Population	Set of solutions
Fitness	Quality of solution
Chromosome	Representation of a solution
Gene	Part of representation of a solution; decoding of solutions.

II. Literature Survey

(Y.K Kwak and Ahmed, 1994) proposed taxonomy of scheduling algorithms for allocating directed task graphs to multiprocessors. Various classes of algorithms have been presented which includes BNP, TDB, UNC and APN depending upon various parameters like communication, duplication etc [1].

(Y.K Kwak and Ahmed, 1999) analyzed and compared 15 scheduling algorithms implemented on SUN SPARK workstations with different benchmarks described my them. Comparisons were made using parameters like NSL, number of processors used, running time and scheduling scalability [2].

(Melanie Mitchell, 1988) laid the foundation for understanding of genetic algorithms. Elements of genetic algorithms like chromosome, fitness, mutation etc have been explained with examples. Various selection methods like elitism, tournament selection etc have been discussed. No prior knowledge of genetic algorithms is assumed [3].

(Gerasoulis and Yang, 1994) presented a low complexity heuristic called Dominant Sequence Clustering algorithm and compared the performance of the algorithm with algorithms like MD, ETF and EZ [4].

(David. E. Goldberg, 1990) helped in understanding genetic algorithms, their mechanics and power. Basic genetic concepts are explained with examples. Most of the algorithms are presented in Pascal computer programs [5].

(F.Omara and Arafa, 2009) developed two genetic algorithms with added heuristic principles to improve the performance. Two fitness functions have been applied to reduce execution time and load balancing. Task duplication has also been applied to overcome the communication overheads [6].

(A.J. Page and M. Keane, 2010) presented a multi-heuristic dynamic task allocation in a heterogeneous distributed system. It utilizes GA with eight common heuristics to minimize the total execution time.[7]

(Gurvinder Singh and Jasbir Singh, 2012) proposed a Performance Effective GA considering three parameters depended on each other. The performance of PEGA was compared with FCFS, SJF and RR scheduling methods [8].

III. Problem Definition

The objective of scheduling is to minimize the overall finish-time of the program by proper allocation of the tasks to the processors and sequencing the execution of the tasks. The overall finish-time of a parallel program is commonly called makespan or schedule length.

The system consists of limited number of fully connected homogeneous processors; processors with same speed and capabilities. In Static scheduling, where the state of program and system is known in advance, a parallel program can be represented by a directed acyclic graph (DAG) G=(V, E), where V is a set of *t* nodes and E is a set of *e* edges. A node represents a task in DAG which in turn is a set of instructions which must be executed sequentially with no preemption in same processor. The weight assigned to the node n_i is called computation cost and is denoted by w (t_i). The edges in DAG denoted by (t_i, t_j), correspond to the communication messages. The weight of an edge is called the communication cost of the edge and is denoted by c (t_i, t_j). This cost is incurred if n_i and n_j are scheduled on different processors and is considered to be zero if t_i and t_j are scheduled on the same processor [1].

The objective of this problem is to find an assignment of tasks to processors such that the schedule length is minimized and, in the same time, the precedence constrains are preserved. A Critical Path (CP) of a task graph is defined as the path with the maximum sum of node and edge weights from an entry node to an exit node. A node in CP is denoted by CP Nodes (CPNs).

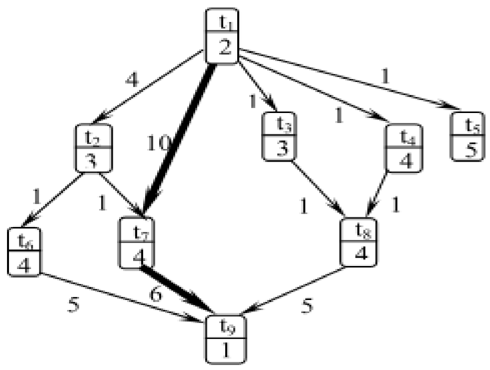


Figure1: DAG Example

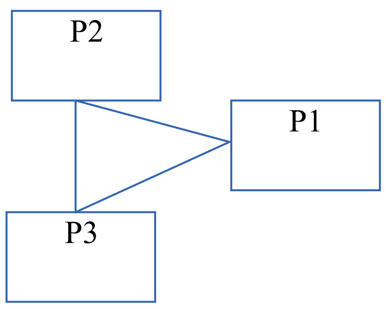


Figure 2: Fully Connected Parallel Processor

An example of DAG is represented in Fig.1, where bold line represents the critical path. Fig.2 represents the multiprocessor computing which consists of set of homogeneous processors.

IV. Genetic Approach For Scheduling

Genetic Algorithms are considered as heuristics which simulates evolution. They are for obtaining high quality solutions for a broad range of combinatorial optimization problems including the task scheduling problem. The main strength of genetic algorithms is their robustness. The implications of robustness for artificial systems are manifold. If artificial systems can be made more robust, costly redesigns can be reduced or eliminated. Self-repair, self-guidance, and reproduction are the rule in biological systems, where as they barely exist in most sophisticated systems [6].

4.1 Working of Genetic Algorithms

The GA operates on a population of solutions rather than a single solution. The genetic search begins by initializing a population of individuals. Individual solutions are selected from the population, then are mated to form new solutions. The mating process is implemented by combining or crossing over genetic material from two parents to form the genetic material for one or two new solutions; this transfers the data from one generation of solutions to the next.

Random mutation is applied periodically to promote diversity. The individuals in the population are replaced by the new generation. A fitness function, which measures the quality of each candidate solution according to the given optimization objective, is used to help determining which individuals are retained in the population as successive generations evolve.

4.2 Difference from other optimization techniques

GA and other search and optimization procedures differ in some important ways:

- working with a coding of parameter set, not the parameters themselves;
- parallel search from population of points, not a single point;
- fitness function(Objective function) information is used, not derivatives or other knowledge;
- probabilistic transition rules, not deterministic
- Coding.

The mechanism of simple GA is amazingly simple, involving nothing more than copy of strings and swapping of partial strings. Simplicity and robustness are the two attractions of the GA approach.

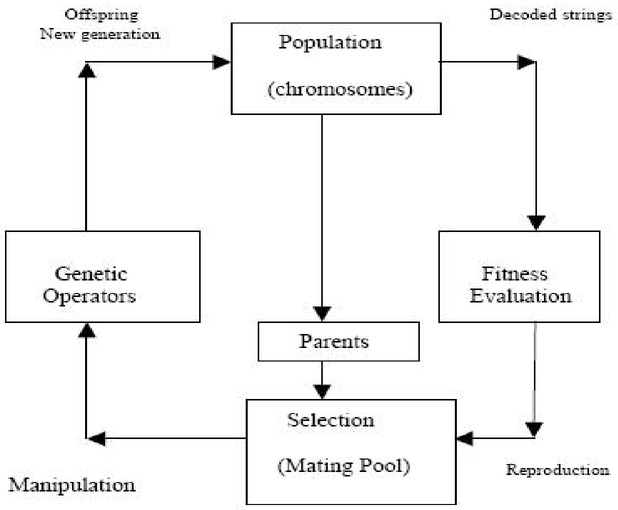


Fig. 3Genetic Algorithm Cycle

V. Proposed Genetic Algorithm

The proposed algorithm is described as follows:

Step1: Define parameters: Read DAG (number of tasks l, computation_time, communication_delay between tasks, number of processors p, crossover and mutation rate, population_size and number of generations (nm_gen).

Step2: Population generation: Generate initial_pop with random initialization of genes, cs.pop_generate(l,p).

Step3: while nm_gen not finished do

(a) Calculate the fitness of each chromosome individual in the initial_pop. fit_evaluate(initial_pop, computation_time, communication_delay, p) and save in initial_fit_value ArrayList.
(b) Selection: select a pair of chromosomes as parents using certain selection scheme or randomly. Parent = selection (initial_pop, initial_fit_value).
(c) Evolution1: Perform crossover operation on the selected parents with the selected rate (one point crossover). Crossover (parent1, parent2, crossover_site).
(d) Evolution2: Perform mutation on the offspring generated from crossover with selected rate. Mutate (offspring [0], location).
(e) Replace the entire initial_pop with new population; final_pop. Also apply elitism to retain certain individuals from initial_pop.

Step5: endfor
Step6: report the best chromosome with fitness value.

VI. Experimental Setup And Results

The genetic algorithm is coded in JAVA language. Parameters like population_size, nm_gen, mutation and crossover rate are kept constant over different scenarios.

6.1 Population Initialization

The initial population is constructed randomly. The chromosome value is chosen randomly from 1 to p, p is the number of processors in the system.

1 1 2 1 3 1 1 2 3 1 2 1 1 2 1

Figure 4: Chromosome Representation

Fig. 4 represents the chromosome for 15 task graph. The numbers represents the processor assignment; tasks t_1,t_2 will be scheduled on p1, then t3 is scheduled on p2. The length of chromosome is linearly proportional to number of tasks. Population_size kept is 50.

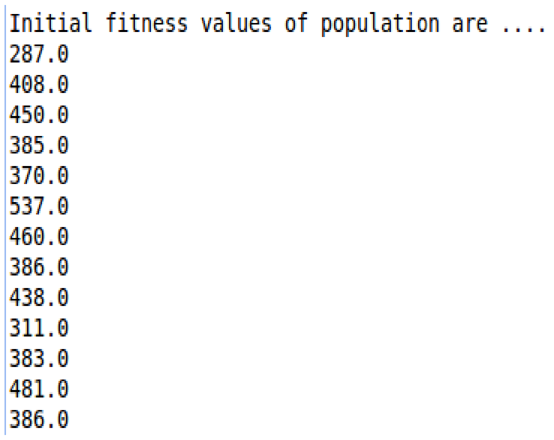
6.2 Fitness Function

This function provides the method to evaluate each chromosome in problem domain. In the minimization problem the most fit individuals would have the lowest values.
Summation fitness function is used given by:

$$\sum_{i=1}^{n} \text{computation\_cost} + \text{communication\_delay} + \text{waiting time}$$

Where n is number of tasks.

Communication delay between the tasks (ti, tj) is added only when they are scheduled on the different processors otherwise comnication delay is considered zero.



```
Initial fitness values of population are ....
287.0
408.0
450.0
385.0
370.0
537.0
460.0
386.0
438.0
311.0
383.0
481.0
386.0
```

Figure 5: Fitness Values generated.

6.3 Selection

"Survival of the fittest" mechanism is used where the fitter solutions survive while the weaker ones die. So the selection operator improves the average quality of the population by giving high-quality chromosome a better chance to get copied into next generation [4].

There are different approaches that can be applied in the selection phase. Fitness - proportional roulette wheel selection and tournament selection are compared such that the better method is used The tournament selection method produces better quality population than that produced by the roulette wheel selection. Therefore, the tournament selection method is used in the work of this paper.

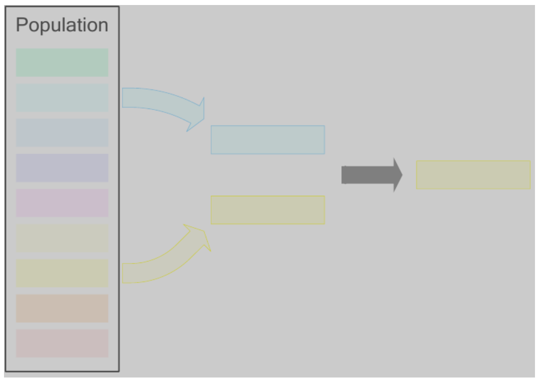


Fig. 6 Tournament Selection

6.4 Crossover

Crossover produces new individuals that have some portions of both parent's genetic material. The simplest form of crossover; single point crossover is used. The operator selects two individuals as parents and mates them to produce two child (offspring).For example, the strings 10000100 and 11111111 could be crossed over after the third locus in each to produce the two offspring 10011111 and 11100100. The crossover operator roughly mimics biological recombination between two single‒chromosome (haploid) organisms.

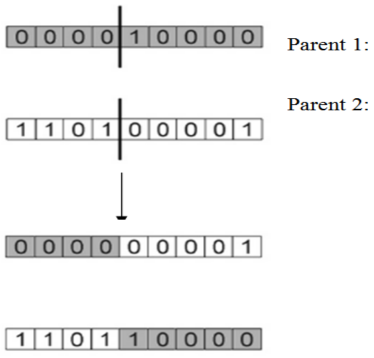
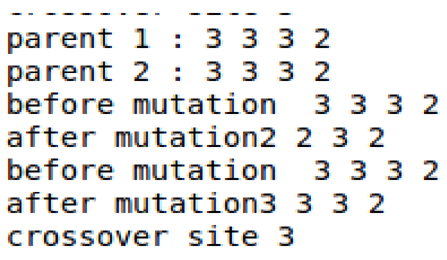


Figure 7: Single Point Crossover

6.5 Mutation

This operator randomly flips some of the bits in a chromosome. For example, the string 00000100 might be mutated in its second position to yield 01000100. Mutation can occur at each bit position in a string with some probability, usually very small (e.g., 0.001).



```
parent 1 : 3 3 3 2
parent 2 : 3 3 3 2
before mutation  3 3 3 2
after mutation2 2 3 2
before mutation  3 3 3 2
after mutation3 3 3 2
crossover site 3
```

Figure 8(a) : Mutation operation

Mutation ensures that the probability of generating an optimal solution is never zero.
Fig.8 shows mutation operator applied to selected individuals. Mutation parameter is kept 0.4; 20% of the individuals are mutated. Pseudo-code for mutation is:

```
Mutation()
select offspring from the crossover function;
calculate dependency of the offspring generated from crossover;
dependency=fc.comm_cost(p,comm_delay);
generate random float value dec between 0 and 1;
if(dec<0.4)
{
change the string with high values of dependency between 0 to p;
break;
}
```

The offspring generated after the crossover are used in mutation function. Firstly, the cost dependency of the chromosome is calculated by calling function in fitness class and then the random number is generated between 0 and 1. Then the value of the random number generated in compared with the mutation rate which is 20% here, then those strings with high dependency are mutated to another value between 0 to number of processors (p).

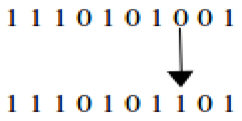
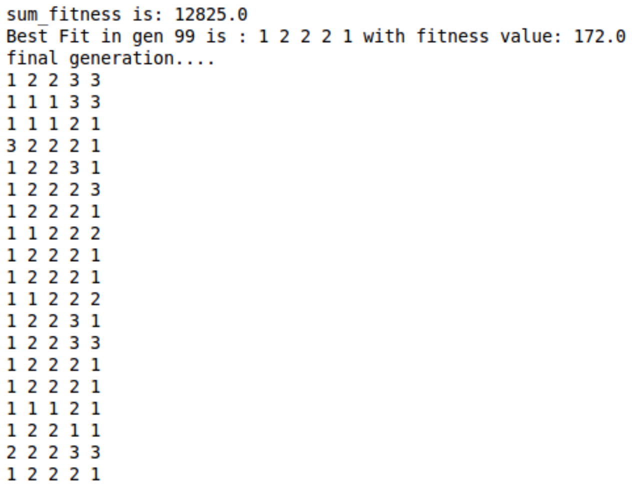


Figure 8(b) : Mutation

6.6 Final Generation

The Genetic Algorithm is a stochastic search method where the average fitness of the population is expected to improve after every generation. GA is terminated after certain number of generations or if the desired fitness value has obtained. Sum_fitness of all the individuals is calculated after every generation and is expected to improve after every generation, which indicates that population of solutions is becoming fitter.



```
sum_fitness is: 12825.0
Best Fit in gen 99 is : 1 2 2 2 1 with fitness value: 172.0
final generation....
1 2 2 3 3
1 1 1 3 3
1 1 1 2 1
3 2 2 2 1
1 2 2 3 1
1 2 2 2 3
1 2 2 2 1
1 1 2 2 2
1 2 2 2 1
1 2 2 2 1
1 1 2 2 2
1 2 2 3 1
1 2 2 3 3
1 2 2 2 1
1 2 2 2 1
1 1 1 2 1
1 2 2 1 1
2 2 2 3 3
1 2 2 2 1
```

Figure.9: Final Generation

Fig.9 shows the final_pop generated by GA. So the best schedule generated is 1 2 2 2 1 for a 5 task DAG with fitness value of 172.0. Sum_fitness is 12825.0 which is calculated after every generation. The algorithm is run for 100 generations.

6.7 Results

Following parameters for the setup was adopted: population size of 50, 100 generations, single-point crossover, mutation probability 0.4, and number of tasks were: 6,9,12,15,18,21 and tournament size of 2.
GA was implemented in JAVA language, data structure Array List was used to store and manage the data. To

calculate the reasonable average and certain values each GA was repeated 5 times.

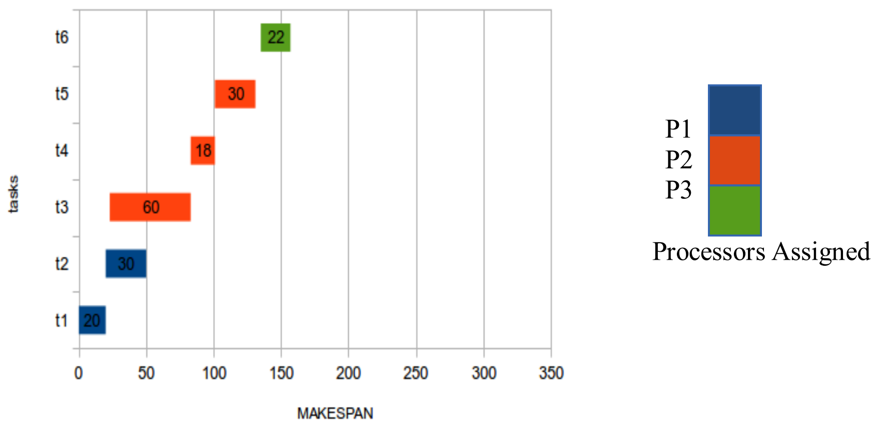


Figure 10. Makespan

Fig.10 shows the makespan of the 6 node DAG Scheduled by GA which took 0.6 sec. Fig. 11 shows that with increase in number of nodes the schedule-time also increases.

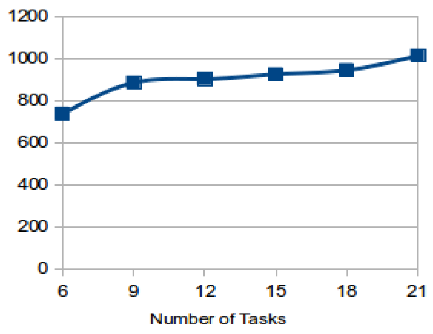
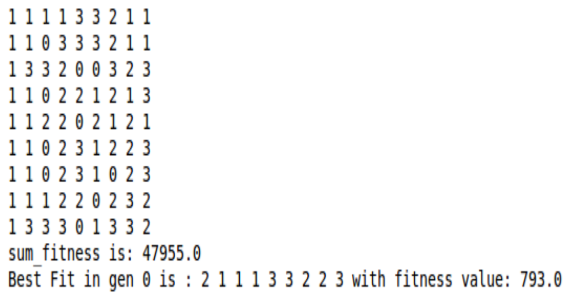
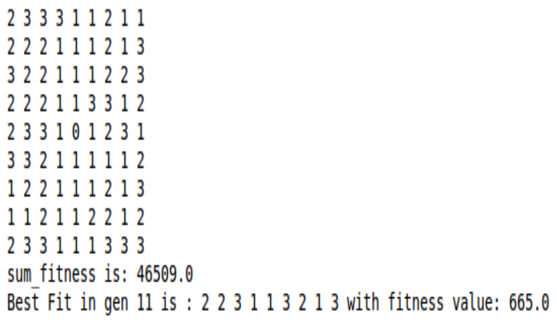


Figure 11 Average time **Vs** Tasks



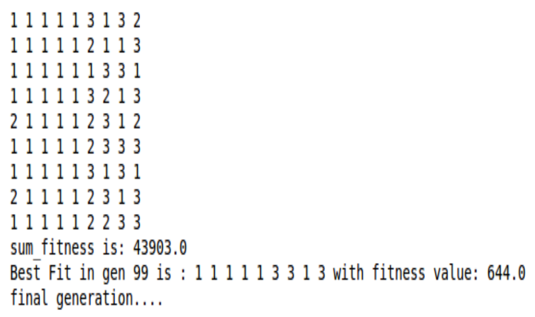
```
1 1 1 1 3 3 2 1 1
1 1 0 3 3 3 2 1 1
1 3 3 2 0 0 3 2 3
1 1 0 2 2 1 2 1 3
1 1 2 2 0 2 1 2 1
1 1 0 2 3 1 2 2 3
1 1 0 2 3 1 0 2 3
1 1 1 2 2 0 2 3 2
1 3 3 3 0 1 3 3 2
sum_fitness is: 47955.0
Best Fit in gen 0 is : 2 1 1 1 3 3 2 2 3 with fitness value: 793.0
```

Figure 12 Generation number 0



```
2 3 3 3 1 1 2 1 1
2 2 2 1 1 1 2 1 3
3 2 2 1 1 1 2 2 3
2 2 2 1 1 3 3 1 2
2 3 3 1 0 1 2 3 1
3 3 2 1 1 1 1 1 2
1 2 2 1 1 1 2 1 3
1 1 2 1 1 2 2 1 2
2 3 3 1 1 1 3 3 3
sum_fitness is: 46509.0
Best Fit in gen 11 is : 2 2 3 1 1 3 2 1 3 with fitness value: 665.0
```

Fig. 13 Generation number 11



```
1 1 1 1 1 3 1 3 2
1 1 1 1 1 2 1 1 3
1 1 1 1 1 1 3 3 1
1 1 1 1 1 3 2 1 3
2 1 1 1 1 2 3 1 2
1 1 1 1 1 2 3 3 3
1 1 1 1 1 3 1 3 1
2 1 1 1 1 2 3 1 3
1 1 1 1 1 2 2 3 3
sum_fitness is: 43903.0
Best Fit in gen 99 is : 1 1 1 1 1 3 3 1 3 with fitness value: 644.0
final generation....
```

Fig.14 Generation number 99

I n GA the fitness of population as improves with each generation, and fit chromosome is selected after every generation
Fig 12 shows sum_fitness of the population and the fittest chromosome in them with its fit_value.
Fig. 13 show at generation 11 the sum_fitness which corresponds to overall fitness of population has improved and the fitter chromosome has better value than in generation 0.
Fig 14 shows that in final generation, the overall fitness of the population has improved considerably and reported a fittest chromosome with value 644.0

VII. Performance Analysis Under Varying Conditions

Performance of GA is evaluated under various altering parameters and then compared to get the best GA possible.

7.1 Different Selection Methodologies

The GA was implemented by using two different methodologies; roulette wheel selection and tournament selection. It was observed that tournament selection produces much fitter population than roulette wheel method. The results are shown in Fig 15

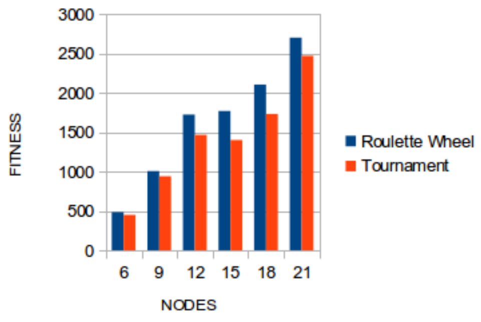


Figure15: Selection Methodologies

7.2 Different Population Size

GA was run five times for population size of 20 and 50. It was observed that GA produced better solution with population size of 50 than 20, but Increasing the population size to 100 doesn't improve the best solution much as providing 50 initial solutions seems sufficient to get an optimum solution. Fig 16. shows the performance of GA for different initial_pop.size().

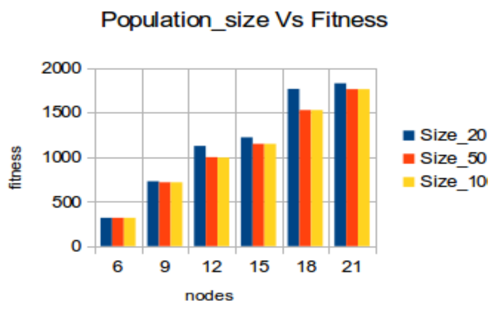


Figure 16: Population Size Vs Fitness

7.3 Different Crossover And Mutation Type

GA is implemented using two different types of method to tune crossover and mutation rate, based on idea of sustaining diversity and prevent premature convergence.

Adaptive method proposed by Srinivas and Patnaik [9] has been applied, the crossover rate C is defined as:

C=(kc*(f_max-f_c)/(f_max-f_avg));

where,

f_max is the maximum fitness value,
f-avg is average fitness value,
f_c is larger of the fitness values to be crossed and
kc is constant less than 1.0 to constraint C to 1.0.

The mutation rate M is defined as:

M=(km*(f_max-f_m)/(f_max-f_avg));
where,

f_m is larger of the fitness values to be muted and km is constant less than 1.0 to constraint C to 1.0.

Fig 17. shows the results of both methods; static as well as adaptive parameters applied in the GA.

The results show that GA with adaptive parameters converge to optimum much earlier than static parameters. So the adaptive parameters method is better than static parameters method. For each case GA was run 4 times and rest of the parameters like population size, number of generations, selection method was kept constant.

7.4 Speedup and Efficiency

The speedup value for a given graph is computed by dividing the sequential execution time, cumulative computation costs of the tasks in the graph, by the parallel execution time, the makespan of the output schedule. Efficiency, the ratio of the speedup value to the number of processors used [10].

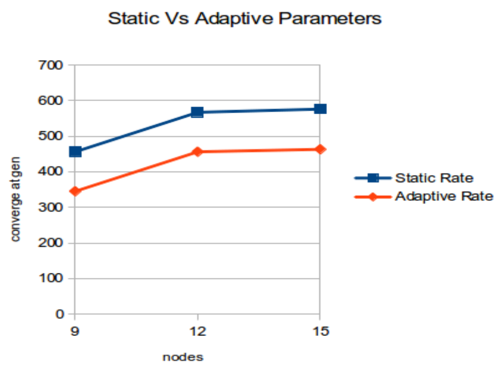


Figure 17: Static Vs Adaptive Parameters

Figure 17 shows that using adaptive parameters the algorithm converges at much lesser generations then static parameters.

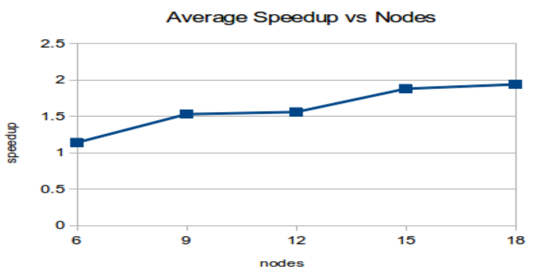


Figure 18: Average Speedup Vs Nodes

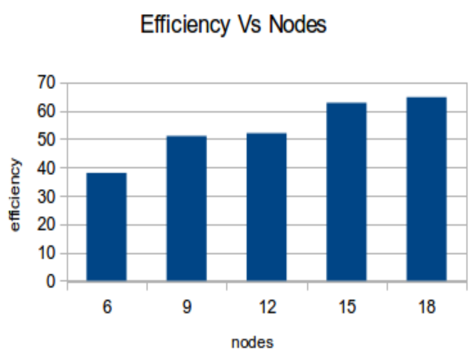


Figure 19: Efficiency Vs Nodes

VIII. Conclusion

This paper presents GA implementation of task scheduling problem. Some modifications have been added to improve the performance of GA. Proposed GA is implemented under different parameter values and performance is evaluated. To ensure diversity in the population, random initialization is used. Results show that with increase in number of nodes the speedup tends to increase but not linearly as communication overheads also increase. Using adaptive parameters ensure that solution don't stuck at local minima and ensure better convergence rate.

References

[1] I. Ahmad, Y. Kwok, A new approach to scheduling parallel programs using task duplication, in: Proceeding of the 23rd International Conf. on Parallel Processing, August 1994, North Carolina State University, NC, USA, 1994.

[2] I. Ahmad, Y. Kwok, Benchmarking and comparison of the task graph scheduling algorithms, J. Parallel Distrib. Comput. 95 (1999) 381–422.

[3] Melanie Mitchell, "An Introduction to Genetic algorithms", The MIT Press, February 1998.

[4] Y. Kwok, I. Ahmad, Dynamic critical path scheduling: An effective technique for allocating task graphs to multi-processors, IEEE Trans. Parallel Distrib. Syst. 7 (1996) 506–521.

[5] D. E. Goldberg, "Genetic algorithms in search, optimization & machine learning", Addison Wesley, 1990.

[6] F.A. Omara, M. Arafa, Genetic algorithms for task scheduling problem,J. Parallel Distrib. Comput. 70 (2010) 13–22

[7] A.J. Page, M. Keane,Multi-heuristic Dynamic task allocation using genetic algorithms in a heterogeneous distributed system,J. Parallel Distrib. Comput. 70 (2010) 758-765.

[8.] Gurvinder Singh, J. Singh, Task Scheduling using Performance Effective Genetic Algorithm for Parallel Heterogeneous System, International Journal of Computer Science and Telecommunications [Volume 3, Issue 3, March 2012].

[9] M. Srinivas, L.M. Patnaik, Adaptive probabilities of crossover and mutation in genetic algorithm, IEEE Trans. Syst. Man Cybern. 24 (4) (1994) 656–667.

[10] Haluk Topcuoglu, Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing,IEEE transactions on parallel and distributed systems, vol. 13, no. 3, march 2002.

[11] M. Wu, D.D. Gajski, Hypertool: A programming aid for message-passing systems, IEEE Trans. Parallel Distrib. Syst. 1 (1990) 381–422.