INTERNATIONAL JOURNAL OF ADVANCE RESEARCH, IDEAS AND INNOVATIONS IN TECHNOLOGY

ISSN: 2454-132X

(Volume2, Issue3)

Available online at: www.Ijariit.com

EFFICIENT SCHEDULING BY GENETIC ALGORITHM AND SIMULATED ANNEALING

Pooja Nehra[1], Mr. Sunil Ahuja[2]

Student[1], Associate Professor[2]

Department Of Computer Science and Engineering, DIET, Karnal Haryana-132001

Abstract: Multiprocessing is the ability of a system to support more than one processor and the ability to allocate tasks between them. The main advantage of using multiprocessor system is to get more work done in shorter period of time. To improve the efficiency of CPU, we do scheduling by the use of genetic algorithms. Genetic algorithms are powerful and widely applicable stochastic search and optimization methods based on the concepts of natural selection and natural evaluation. Simulated annealing is a generic probabilistic metaheuristic for the worldwide optimization issue of finding a great approximation into the global optimum of a given function .In this paper, efficient genetic algorithm and simulated annealing have been proposed for solving the problem of CPU scheduling. The operator which are used for implementing the genetic algorithm such as real value encoding for encoding, Roulette wheel method is used for selection, Uniform crossover operator is used for crossover, interchange for mutation.

Keywords: Multiprocessor system, Processes, CPU scheduling, heuristic methods.

I. INTRODUCTION

Multiprocessing is the coordinated processing of programs by more than one computer processor. Multiprocessing is a general term that can mean the dynamic assignment of a program to one of two or more computers working in tandem or can involve multiple computers working on the same program at the same time (in parallel). The optimal rehearse of processors is also expected. After extra optimal task arranging in multiprocessors arrangements is NP-complete, that is, discovering optimal arranging of task for multiprocessor is era consuming. These delineate the setback of task arranging on multiprocessor arrangements to allocate a set of tasks to processors such that the optimal rehearse of processors and acceded computational era for arranging algorithm are obtained. Genetic algorithm, an evolutionary algorithm can be used to find out near optimal solution. To compare the performance of our algorithm, we have also implemented another scheduling algorithm HEFT which is a heuristic algorithm.

II. SCHEDULING

Process scheduling is an essential part of a multiprogramming operating system. Such operating system allow more than one process to be loaded into executable memory at a time and loaded process shares the CPU.CPU scheduling is a process which allows one process to use the CPU while the execution of another process is on hold (in waiting state) due to unavailability of any resource like I/O etc, thereby making full use of CPU. The aim of CPU scheduling is to make the

System efficient, fast and fair. The target of arranging is to allocate resources to the tasks such that one or supplementary aims are optimized. Arranging becomes a vital concern afterward resources are manipulated and demand to be utilized to finish countless contesting tasks efficiently.

Single machine scheduling

Single machine scheduling is the process of assigning a group of task to a single machine or resource. The task are arranged so that one or many performance measures may be optimized. The single machine scheduling problem involves scheduling a set of tasks to a single resource. This is accomplished by determining a sequence that includes each task, and then assigning the tasks to the resource. Each task can be given a priority, ready time, processing time and due date. The value of the performance measures can be computed based on this information and the sequence of tasks. This problem grows in complexity at an exponential rate as the number of tasks to be scheduled increases.

Parallel machine scheduling

Parallel machine scheduling involves scheduling a set of tasks on two or more machines that work in parallel with each other. The mechanisms present identical procedures and could or could not work at the alike pace. . An example layout is shown in Figure 1-2 – Parallel Contraption Layout. In this kind of setback, the tasks are allocated to whichever contraption for processing, and flow amid mechanisms is not allowed.

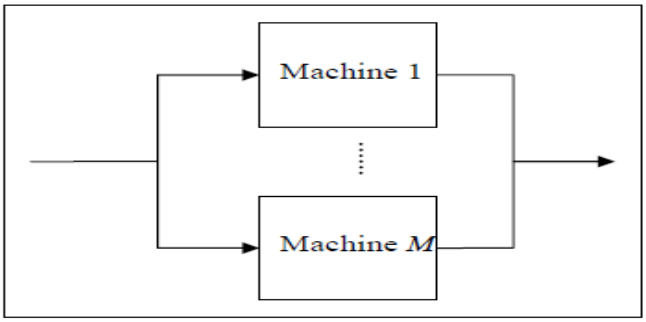


Fig 1: Parallel Machine Layout

Flow shop scheduling

A flow shop scheduling problem consists of two or more machines and a set of tasks that must be processed on each of these machines. This arrangement is called a flow shop because products flow along a specific unidirectional path. Each product must be processed on each machine in the same order e.g. 1st - machine 1, 2nd – machine 2, …, mth – machine m. The processing times for each job can vary from machine to machine and the processing times on each machine can vary from job to job.

Manufacturing Cell Scheduling

Manufacturing cell scheduling consists of a set of jobs that are grouped into part families and a set of machines that are grouped into machine cells. These part families are based on production similarities; that is, the number of operations that are performed using the same machines. The machines in the system are then grouped into cells based on the part families. Some of the Advantages of manufacturing cells include increased product variety, increased machine utilization, and reduced rework. Scheduling can be done based on the machine cell and the part family information.

Job shop scheduling

A job shop consists of two or more machines that perform specific operations, and a set of tasks that must be processed on some or all of these machines. Unlike the flow shop problem, there is no fixed path that products must follow through the system therefore the order of operations is not fixed. This type of layout is typically used when product variety is high and product volume is low.

III. GENETIC ALGOITHM

Genetic algorithm belong to the larger class of evolutionary algorithm (EA), which generate solutions to optimization problem using techniques inspired by natural evolution, such as inheritance , mutation, selection and crossover. The evolution usually starts from a population of randomly generated individuals, and is an iterative process, with the population in each iteration called a generation. In each generation, the fitness of every individual in the population is evaluated; the fitness is usually the value of the objective function in the optimization problem being solved. The more fit individuals are stochastically selected from the current population, and each individual's genome is modified to form a new generation. The new generation of candidate solution is then used in the next generation of the algorithm. Commonly, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population.

How genetic algorithm works?

Genetic algorithm is started with a set of solutions (represented by chromosomes) called population. Solutions from one population are taken and used to form a new population. This is motivated by a hope, that the new population will be better than the old one. Solutions which are selected to form new solutions (offspring) are selected according to their fitness – the more suitable they are the more chances they have to reproduce. This is repeated until some condition is satisfied. The general structure of the Genetic algorithms is as follow:

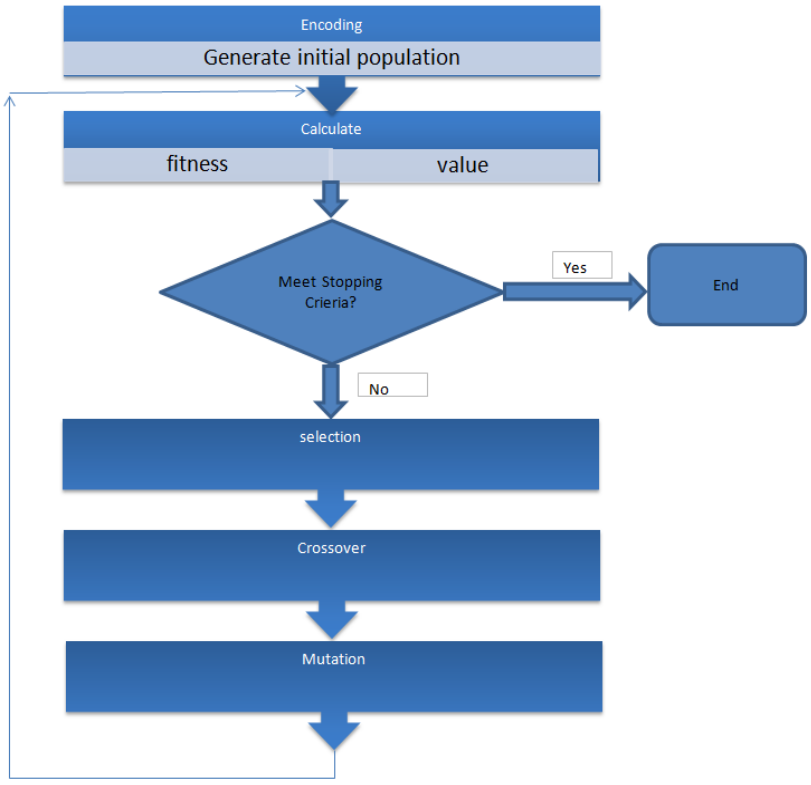


Fig 2: Flow chart of an Efficient Genetic algorithm

IV. GENETIC ALGORITHM OPERATOR

There are three basic genetic algorithms operators which are selection, crossover and mutation. The two operator's mutation and crossover are work together to explore and exploit the search space by creating new variants in the chromosomes.

A. Selection

Selection operators give preference to better solutions (chromosomes), allowing them to pass on their 'genes' to the next generation of the algorithm. The best solutions are determined using some form of objective function (also known as a 'fitness function 'in genetic algorithms), before being passed to the crossover operator. Different methods for choosing the best solution exist, for

example roulette wheel selection, fitness proportionate selection, different methods may choose different solutions are being 'best'. The selection operator may also simply pass the best solutions from the current generation directly to the next generation without being mutated; this is called elitist selection.

B. Crossover

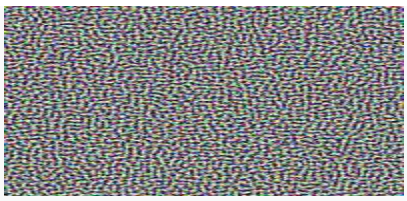
Crossover is the process of taking more than one parent solutions (chromosomes) and producing a child solution from them. By recombining portions of good solutions, the genetic algorithm is more likely to create a better solution. The chromosome method is often representation of the solution; this may become particularly important when variables are grouped together as building blocks, which might be disrupted by a non- respectful crossover operator.

C. Mutation

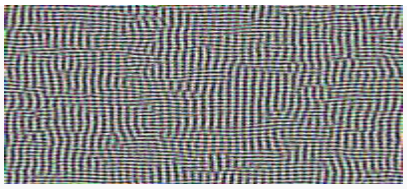
The mutation operator encourages genetic diversity amongst solutions and attempts to prevent the genetic algorithm converging to a local minimum by stopping the solutions becoming too close to one another. In mutating the current pool of solutions, a given solution may change entirely from the previous solution. By mutating the solutions, a genetic algorithm can reach an improved solution through the mutation operator.

V. SIMULATED ANNEALING

Simulated annealing is a probabilistic technique for approximating the global optimum of a given function. Specifically, it is a metaheuristic to approximate global optimization in a large search space. It is often used when the search space is discrete. For problems where finding the precise global optimum is less important than finding an acceptable local optimum in a fixed amount of time , simulated annealing may be preferable to alternatives such as brute –force search or gradient descent. Simulated annealing interprets slow cooling as a slow decrease in the probability of accepting worse solutions as it explores the solution space. Accepting worse solutions is a fundamental property of metaheuristics because it allows for a more extensive search for the optimal solution. The name and inspiration of the algorithm demand an interesting feature related to the temperature variation to be embedded in the operational characteristics of the algorithm. This necessitates a gradual reduction of the temperature as the simulation proceeds. The algorithm starts initially with T set to a high value (or infinity), and then it is decreased at each step following some *annealing schedule*—which may be specified by the user, but must end with $T = 0$ towards the end of the allotted time budget. In this way, the system is expected to wander initially towards a broad region of the search space containing good solutions, ignoring small features of the energy function; then drift towards low-energy regions that become narrower and narrower; and finally move downhill according to the steepest descent heuristic.



Fast



Slow

Example illustrating the effect of cooling schedule on the performance of simulated annealing. The problem is to rearrange the pixels of an image so as to minimize a certain potential energy function, which causes similar colours to attract at short range and repel at a slightly larger distance. The elementary moves swap two adjacent pixels. These images were obtained with a fast cooling schedule (left) and a slow cooling schedule (right), producing results similar to amorphous and crystalline solids, respectively.

VI. RESULT AND ANALYSIS

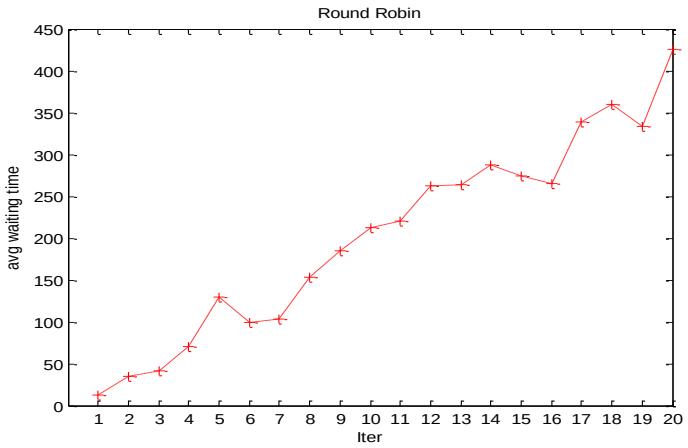


Fig 3: Average Waiting time for Round Robin Algorithm

This output displays that the Multilevel Queue Algorithm is run for 10 number of iteration .The bars display the average staying time. At every single iteration we are rising the number of processes. Next design them and computing the average staying period. In this we are implementing the multi level queue at two levels. At First level the processes are schedule with first come first serve and second level with round robin then calculating the average waiting time of these two.

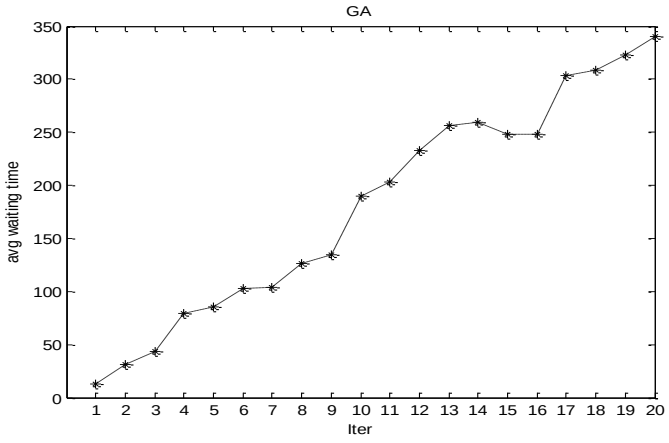


Fig 4: Average Waiting Time for Genetic Algorithm

This output displays that the GA algorithm is run for 20 number of iteration .The bars display the average staying time. At every single iteration we are rising the number of processes. Next design them and computing the average staying time. In this genetic algorithm we are employing disparate kind of operator.

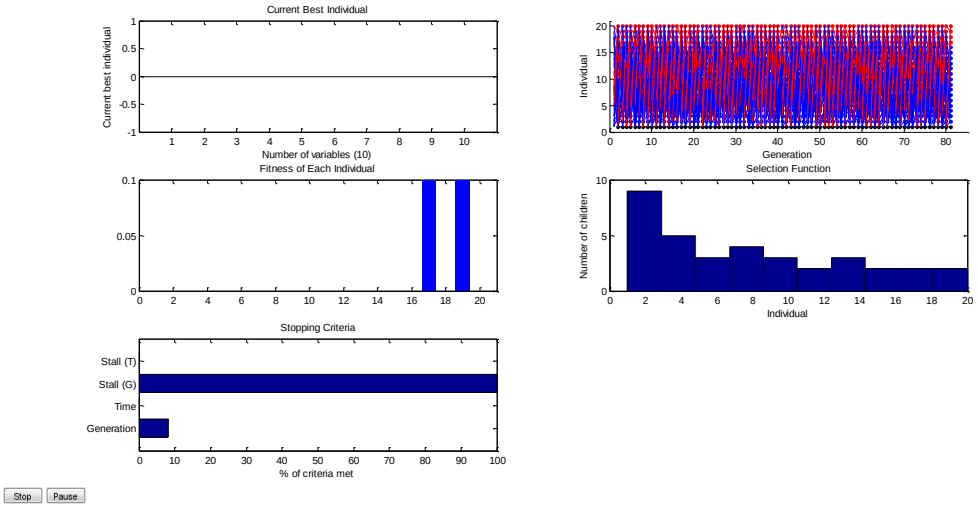


Fig 5: Best Solution and individual in Simulated Annealed Genetic Algorithm

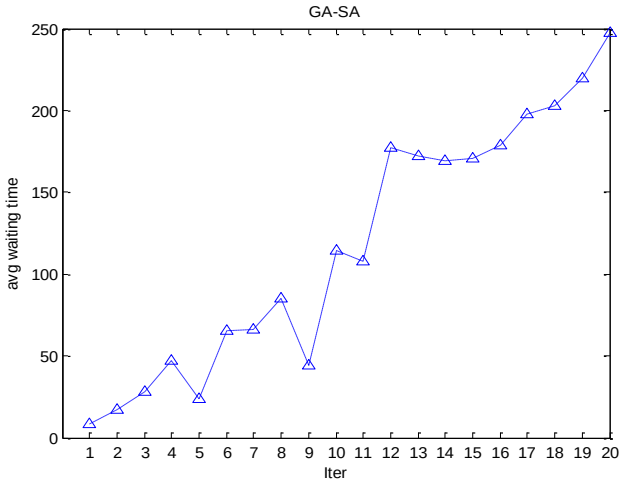


Fig 6: Average Waiting Time for Simulated Genetic Algorithm

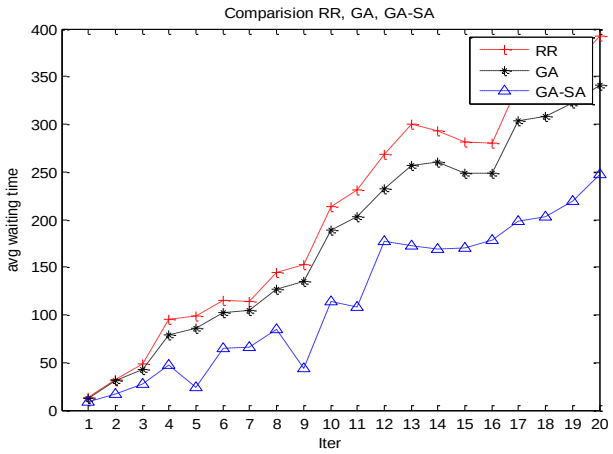


Fig 7 : Overall Comparison of Various Policies with respect to Iterations

VII. CONCLUSION AND FUTURE WORK

In this dissertation we are implementing three algorithms for CPU arranging such as genetic algorithm, round robin algorithm, and Simulated Genetic Algorithm .In this we are contrasting the consequence of these alongside every single supplementary and discovered that the presentation of the simulated annealed genetic is best among these two. In this implementation the procedures are rise by five at every single time. The new representation has primarily been tested on a data to assess its effectiveness. The simulation aftermath clearly display that the counselled way is able to find optimized solution. In this we are implementing the multilevel queue with two queues. So in future we can increase the number of queues with different level of priority. In this at each iteration five numbers of processes are increase, so we can change this factor. The performance of simulated genetic is best between RR, GA, so we can compare this with other scheduling algorithm.

REFERENCES

[1]. Bleuse, Raphaël, Thierry Gautier, Joao VF Lima, Grégory Mounié, and Denis Trystram. "Scheduling data flow program in xkaapi: A new affinity based Algorithm for Heterogeneous Architectures." In Euro-Par 2014 Parallel Processing, pp. 560-571. Springer International Publishing, 2014.

[2]. Hamilton, Brandon Kyle, Michael Inggs, and Hayden Kwok-Hay So. "Mixed-architecture process scheduling on tightly coupled reconfigurable computers." In Field Programmable Logic and Applications (FPL), 2014 24th International Conference on, pp. 1-4. IEEE, 2014.

[3]. Bhoi, Sourav Kumar, Sanjaya Kumar Panda, and Debashee Tarai. "Enhancing CPU Performance using Subcontrary Mean Dynamic Round Robin (SMDRR) Scheduling Algorithm." arXiv preprint arXiv:1404.6087 (2014).

[4]. Deng, Yunhua, and Rynson WH Lau. "Dynamic load balancing in distributed virtual environments using heat diffusion." ACM Transactions on Multimedia Computing, Communications, and Applications (TOMCCAP) 10, no. 2 (2014): 16.

[5]. Ahmad, Hifzan, and Neelendra Badal. "CAPS: A Tool for Process Scheduling in Distributed Environment." (2014).

[6]. Panda, Sanjaya Kumar, and Sourav Kumar Bhoi. "An effective round Robin algorithm using min-max dispersion measure." arXiv preprint arXiv:1404.5869 (2014).

[7]. Panda, Sanjaya Kumar, Debasis Dash, and Jitendra Kumar Rout. "A Group based Time Quantum Round Robin Algorithm using Min-Max Spread Measure." arXiv preprint arXiv:1403.0335 (2014).

[8]. Küçüksayacigil, Fikri, and Gündüz Ulusoy. "A genetic algorithm application for multi-objective multi-project resource constrained project scheduling problem." (2014): 49-52.

[9]. Tseng, Po-Hsien, Pi-Cheng Hsiu, Chin-Chiang Pan, and Tei-Wei Kuo. "User-centric energy-efficient scheduling on multi-core mobile devices." In Proceedings of the The 51st Annual Design Automation Conference on Design Automation Conference, pp. 1-6. ACM, 2014.

[10]. Kumar, Sandeep. "Fuzzified Job Shop Scheduling Algorithm." ... 7 (2014): 1-20.

[11]. Mustafa, Besim. "YASS: a system simulator for operating system and computer architecture teaching and learning." European Journal of Science and Mathematics Education 1, no. 1 (2013): 34-42.

[12]. Levine, John, Graeme Ritchie, Alastair Andrew, and Simon Gates. "New Results for the Heterogeneous Multi-Processor Scheduling Problem using a Fast, Effective Local Search and Random Disruption." arXiv preprint arXiv:1312.6246 (2013).

[13]. Yue, Dajun, Gonzalo Guillén-Gosálbez, and Fengqi You. "Global optimization of large-scale mixed-integer linear fractional programming problems: A reformulation-linearization method and process scheduling applications." AIChE Journal 59, no. 11 (2013): 4255-4272.

[14]. Goel, Neetu, and R. B. Garg. "A Comparative Study of CPU Scheduling Algorithms." arXiv preprint arXiv:1307.4165 (2013).

[15]. Panwar, Poonam, A. K. Lal, and Jugminder Singh. "A Genetic Algorithm Based Technique for Efficient Scheduling of Tasks on Multiprocessor System." In Proceedings of the International Conference on Soft Computing for Problem Solving (SocProS 2011) December 20-22, 2011, pp. 911-919. Springer India, 2012.

[16]. Zhuravlev, Sergey, Juan Carlos Saez, Sergey Blagodurov, Alexandra Fedorova, and Manuel Prieto. "Survey of scheduling techniques for addressing shared resources in multicore processors." ACM Computing Surveys (CSUR) 45, no. 1 (2012): 4.

[17]. Qi, Wei, Jinfeng Liu, and Panagiotis D. Christofides. "Supervisory predictive control for long-term scheduling of an integrated wind/solar energy generation and water desalination system." Control Systems Technology, IEEE Transactions on 20, no. 2 (2012): 504-512.

[18]. Zhang, Yun, Soumyadeep Ghosh, Jialu Huang, Jae W. Lee, Scott A. Mahlke, and David I. August. "Runtime asynchronous fault tolerance via speculation." In Proceedings of the Tenth International Symposium on Code Generation and Optimization, pp. 145-154. ACM, 2012.

