Scheduling in Multiprocessor System Using Genetic Algorithms

Keshav Dahal[1], Alamgir Hossain[1], Benzy Varghese[1],
Ajith Abraham[2], Fatos Xhafa[3], Atanasi Daradoumis[4]
[1]University of Bradford, UK, {k.p.dahal; m.a.hossain1}@Bradford.ac.uk, betzymol@yahoo.com
[2]Norwegian University of Science and Technology, Norway, ajith.abraham@ieee.org
[3]Universitat Politècnica de Catalunya, Barcelona, Spain, fatos@lsi.upc.edu
[4]Universitat Oberta de Catalunya, Barcelona, Spain, adaradoumis@uoc.edu

Abstract

Multiprocessors have emerged as a powerful computing means for running real-time applications, especially where a uniprocessor system would not be sufficient enough to execute all the tasks. The high performance and reliability of multiprocessors have made them a powerful computing resource. Such computing environment requires an efficient algorithm to determine when and on which processor a given task should execute. This paper investigates dynamic scheduling of real-time tasks in a multiprocessor system to obtain a feasible solution using genetic algorithms combined with well-known heuristics, such as 'Earliest Deadline First' and 'Shortest Computation Time First'. A comparative study of the results obtained from simulations shows that genetic algorithm can be used to schedule tasks to meet deadlines, in turn to obtain high processor utilization.

1. Introduction

Real-time systems are software systems in which the time at which the result is produced is as important as the logical correctness of the result [1]. That is, the quality of service provided by the real-time computing system is assessed based on the main constraint 'time'. Real-time applications span a large range of activities which include production automation, embedded systems, telecommunication systems, nuclear plant supervision, surgical operation monitoring, scientific experiments, robotics and banking transactions [2].

Scheduling is an important aspect in real-time systems to ensure soft/hard timing constraints. Scheduling tasks involves the allotment of resources and time to tasks, to satisfy certain performance needs [1]. In a real-time application, real-time tasks are the basic executable entities that are scheduled [2]. The tasks may be periodic or aperiodic and may have soft or hard real-time constraints. Scheduling a task set consists of planning the order of execution of task requests so that the timing constraints are met. Multiprocessors have emerged as a powerful computing means for running real-time applications, especially where a uniprocessor system would not be sufficient enough to execute all the tasks by their deadlines [3]. The high performance and reliability of multiprocessors have made them a powerful computing means in time-critical applications [4].

Real-time systems make use of scheduling algorithms to maximize the number of real-time tasks that can be processed without violating timing constraints [5]. A scheduling algorithm provides a schedule for a task set that assigns tasks to processors and provides an ordered list of tasks. The schedule is said to be feasible if the timing constraints of all the tasks are met [2]. All scheduling algorithms face the challenge of creating a feasible schedule.

In multiprocessor real-time systems static algorithms are used to schedule periodic tasks whose characteristics are known a priori. Scheduling of aperiodic tasks whose characteristics are not known a priori requires dynamic scheduling algorithms [4]. Dynamic scheduling can be either centralized or distributed. In a distributed dynamic scheduling scheme, each processor has its own local scheduler that determines whether it can satisfy the requirements of the incoming task. If it cannot satisfy, then the scheduler tries to find another processor which can handle the task. In a centralized dynamic scheduling scheme, there is a central processor called the scheduler which determines which processor the task should be allocated for execution [4, 6]. Srinivasan and Anderson [7] have studied the dynamic scheduling of task systems on multiprocessors, considering the tasks that are allowed to join and leave the system. The two main objectives of task scheduling in real-time systems are meeting deadlines and achieving high resource utilization [4].

Various heuristic approaches have been widely used for scheduling. The use of genetic algorithm (GA) for real-time task scheduling is now been studied extensively. GAs seem attractive to real-time application designer as it relieves the designer from knowing how to construct a solution and the designer just requires knowing how to assess a given solution [8]. Page and Naughton presented a scheduling strategy which makes use of a GA to dynamically schedule heterogeneous tasks on heterogeneous processors in a distributed system [9]. Genetic algorithm has been utilized to minimize the total execution time. The simulation studies presented shows the efficiency of the scheduler compared to a number of other schedulers. However the efficiency of the algorithm for time critical applications has not been studied. A good tradeoff between exploitation and exploration in GA allows to accelerate the search, which is interesting in case of real-time scheduling.

This paper proposes using genetic algorithm incorporating traditional scheduling heuristics to generate a feasible schedule based on the work done by Mahmood [6]. The scheduling algorithm considered, aims in meeting deadlines and achieving high utilization of processors. The paper also aims to provide a comparative study of incorporating heuristics such as 'Earliest Deadline First (EDF)' and 'Shortest Computation Time First (SCTF)' separately with genetic algorithms. The scheduler model considered for the study would contain task queues from which tasks would be assigned to processors. Task queues of varying length would be generated at run time. From the task queue only a set of tasks would be considered at a time for scheduling. The size of the task sets considered for scheduling would also be varied for a comparative study.

2. Task and scheduler models

The real-time system is assumed to consist of m, where $m > 1$, identical processors for the execution of the scheduled tasks. They are assumed to be connected through a shared medium. The scheduler may assign a task to any one of the processors. Each task T_i in the task set is considered to be aperiodic, independent and nonpreemptive. Each task T_i is characterised by: A_i : arrival time; R_i: ready time; C_i: worst case computation time; D_i: deadline.

The scheduler determines the scheduled start time and finish time of a task. If $st(T_i)$ is the scheduled start time and $ft(T_i)$ is the scheduled finish time of task T_i , then the task T_i is said to meet its deadline if $(R_i \leq st(T_i) \leq D_i - C_i)$ and $(R_i + C_i \leq ft(T_i) \leq D_i)$. That is, the tasks are scheduled to start after they arrive and finish execution before their deadlines [3]. A set of such tasks can be said to be guaranteed.

The static code analysis and the average of execution times under possible worst cases help to obtain the worst case computation time of a task. The actual computation time of a task could be more or less than its worst case computation time [10]. It is possible that the actual computation time is less than its worst case computation time due to various factors like dependable loops and conditional statements. The architectural features of the system such as cache hits and dynamic branch prediction may also account for the change in the actual computation time. There might also be cases where the actual computation time of a task is more than its worst case computation time. Manimaran and Murthy [10] have referred techniques such as "Task Pair" scheme to handle such situations. The tasks in the system are assumed to be nonpreemptive so that when a task starts execution on a processor, it finishes to its completion. Tasks may also have precedence constraints. This could be incorporated by modifying the ready time and deadlines of tasks so that they comply with the precedence constraints among them. As dealing with precedence constraints is equivalent to working with modified ready times and deadlines it could be applied for the tasks in the presented system. However this has not been done explicitly for the simulation study presented so the tasks are assumed to be independent.

This paper assumes a centralized scheduling scheme with each processor executing the tasks that fill its dispatch queue. The incoming tasks are held in the task queue and then passed on to the scheduler for scheduling of tasks. It is the central scheduler that allocates the incoming tasks to other processors in the system. Each processor has a dispatch queue associated with it. The processor executes tasks in the order they arrive in the dispatch queue. The communication between the scheduler and the processors is through these dispatch queues. The scheduler works in parallel with the processors. The scheduler schedules the newly arriving tasks and updates the dispatch queue while the processors execute the tasks assigned to them. The scheduler makes sure that the dispatch queues of the processors are filled with a minimum number of tasks so that the processors will always have some tasks to execute after they have finished with their current tasks.

The minimum capacity of the dispatch queues depends on factors like the worst case time complexity of the scheduler to schedule newly arriving tasks [10]. A feasible schedule is determined by the scheduler based on the worst case computation time of tasks satisfying their timing constraints.

The scheduler model showing the parallel execution of scheduler and processors in a centralized scheduling scheme is shown in Figure 1. The scheduling algorithm to be discussed has full knowledge about the set of tasks that are currently active. But it does not have knowledge about the new tasks that arrive while scheduling the current task set.

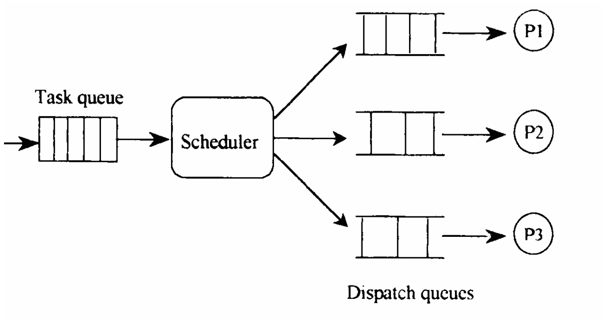

Figure 1. The scheduler model.

The objective of the dynamic scheduling is to improve or maximize what is called the guarantee ratio. It is defined as the percentage of tasks arrived in the system whose deadlines are met. The scheduler in the system must also guarantee that the tasks already scheduled will meet their deadlines.

3. Implementation of scheduling algorithm

Initially a task queue is generated with tasks having the following characteristics namely, arrival time, ready time, worst case computation time and deadline. The tasks are sorted in the increasing order of their deadlines. The tasks are ordered so that the task with the earliest deadline can be considered first for scheduling. The algorithm considers a set of tasks from the sorted list to generate an initial population. In the initial population, each chromosome is generated by assigning each task in the task set to a randomly selected processor and the pair (task, processor) is inserted in a randomly selected unoccupied locus of the chromosome. The size of the chromosome depends on the number of tasks selected from the sorted list. This ensures that part of the chromosome does not remain empty. The tasks in each chromosome are then sorted based on their deadline. This is done because the chromosome representation also gives the order in which the tasks are executed in a processor. The sorting ensures that the tasks with earliest deadline are given priority. The fitness evaluation of the chromosomes in the population is then performed. This helps to determine the number of tasks in each chromosome that meet their deadlines. The chromosomes in the population are then sorted based on the fitness values. The chromosomes are sorted in the descending order of their fitness value.

GA operators are then applied to the population of chromosomes until a maximum number of iterations have been completed. When applying GA operators to the population, reproduction is applied first followed by crossover, partial-gene mutation, sublist-based mutation and then order-based mutation. In each iteration, the tasks in the chromosomes are sorted based on their deadline and the evaluation of the chromosomes and sorting of the chromosomes based on fitness value is performed. After number of iterations the best schedule for the set of tasks is obtained.

The steps in the scheduling algorithm explained above could be summarized as follows:

1. Generate a task queue
2. Sort the tasks in the increasing order of their deadlines
3. Select a suitable number of tasks for a fixed chromosome size
4. Generate chromosomes for the population
5. Sort the genes in each chromosome based on deadline
6. Determine the fitness value of each chromosome in the population
7. Sort the chromosomes within the population depending on fitness value
8. Apply GA operators for a number of iterations:
   Sort the genes in each chromosome based on deadline; Determine the fitness value of each chromosome in the population; Sort the chromosomes within the population depending on fitness value.
9. Choose the best chromosome

The tasks that are found infeasible are removed from the chromosomes so that they are not reconsidered for scheduling. For a task Ti to be feasible it should satisfy the condition that ($Ri \leq st(Ti) \leq Di - Ci$) and ($Ri + Ci \leq ft(Ti) \leq Di$) where Ri is the ready time, Di is the deadline and Ci is the worst case computation time of task Ti . st(Ti) and ft(Ti) denoted the start time and finish time of task Ti respectively. If the condition is not satisfied it is said to be infeasible.

The effectiveness of the scheduling algorithm was studied by conducting simulation studies. The various parameters that have been used in the implementation of the algorithm are discussed in this section. The simulation study considers the assigning of a set of tasks to a number of processors. For these, task queues of different lengths were generated at run time from which a set of tasks were chosen at a time for scheduling. The lengths of task queues considered were 100, 200, 400 and 600. The worst case computation time, Ci, of a task Ti has been chosen randomly between a minimum and maximum computation time value denoted by MIN_C and MAX_C. The values of MIN_C and MAX_C were set to 30 and 60 respectively. The value for the deadline of a task Ti has been randomly chosen between (Ri + 2 * Ci) and (Ri + r * Ci) where $r \geq 2$. This ensures that the computation time is always less than the deadline. For the study, the value of r has been chosen to be 3. The mean of the arrival time was assumed to be 0.05. The number of processors, m considered was 10.

The values for the number of iterations for the application of the GA operators have been based on number of trials. For the value of 'x', which denotes the percentage of tasks to be killed before applying reproduction operator, it has been reported in [5] that best results were obtained with x = 20. Therefore the value of 20 percent has been considered for the algorithm presented in the paper. The chromosome size has been assumed equal to the number of tasks considered at a time for scheduling. Depending on this, the value for the chromosome size has been varied between 20 and 60. As mentioned before the fitness value determines the number of tasks in the chromosome that can meet their deadlines. That is the number of tasks that are feasible. Hence here, for chromosome size 20 the maximum fitness value that can be obtained is 20. The population size for the algorithm has been assumed to be 30. That is 30 chromosomes have been considered at a time for the application of GA operators. Thus the tasks which have been generated with the values for their characteristics chosen appropriately have been considered for scheduling. Initially the tasks were assigned to processors based on 'Earliest Deadline First'. After the results have been observed, the tasks were scheduled using the proposed hybrid GA. A comparison has been made with the latter results obtained which are discussed in the next section. The algorithm was then implemented by incorporating the heuristic 'Shortest Computation time First' with GA. Set of tasks were scheduled using the modified algorithm and the results were observed. A comparison of the results observed by considering the two heuristics separately with GA has been presented in the following section.

4. Results and discussion

For an initial evaluation the fitness value by assigning tasks based on Earliest Deadline First (EDF) was calculated. For this, a task queue of 100 tasks was generated randomly and it was divided into task sets of 20 each. The tasks were ordered in the increasing order of their deadlines and assigned to processors considering earliest deadline first. The processors were chosen randomly between 1 and 10. The fitness value obtained for each task set is shown in Figure 2. The graph shows that the maximum number of tasks that meet their deadlines is 16 when considering 20 tasks for scheduling. The majority of the task sets gave a fitness value of 12 (that is, 12 tasks out of 20 met their deadlines).

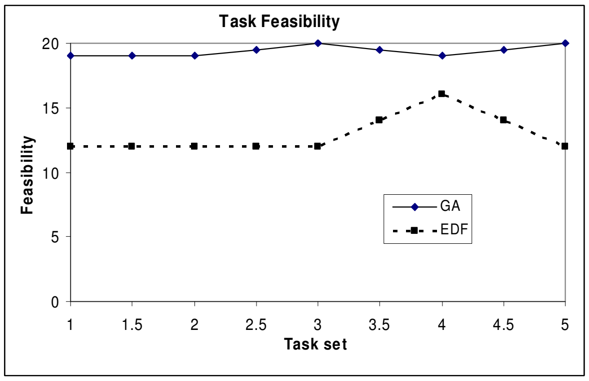

Figure 2. Task feasibility of EDF and GA.

The hybrid algorithm presented in the paper was then used to schedule the same task sets. The algorithm incorporates the heuristic 'Earliest Deadline First' and also GA. Here also a set of 20 tasks was considered at a time. The graph showing the fitness value of tasks obtained using the algorithm is also shown in Figure 2. As shown by the graph, a better performance is obtained by using GA with the heuristic. Thus it could be seen that, the percentage of tasks that are feasible is 95 percent and above.

The algorithm was also studied for different task sets with the same chromosome size. In all the cases the percentage of tasks that are feasible was always 90 percent and above when the chromosome size considered was 20. The results obtained by varying the chromosome size are discussed later. The above results show that GA could be used to schedule task to meet deadlines and also achieve better processor utilization. However, it is worth noting that GAs do have the disadvantage of spending much time in scheduling.

As mentioned earlier in the paper, the population size for the GA was taken to be 30. In the initial population the fitness value of chromosomes were low. As the number of iterations increases a better solution is obtained. The number of iterations considered for the algorithm was 50.

A graph which depicts the change in the feasibility value from the initial to the final iteration for a particular task set of 20 tasks is shown in Figure 3. The graph shows that the fitness value of chromosomes changes gradually from a minimum value of 12 to a maximum value of 20. Thus a better solution can be obtained by applying GA for a good number of iterations. The number of iterations needed for the GA operators was based on a trial method. This was mainly considered for the chromosome size 20. The results of incorporating the heuristic 'Earliest Deadline First' with GA demonstrated better performance. This motivated to study the efficiency of the algorithm by incorporating other heuristics. The heuristic, Shortest Computation time First (SCF) was incorporated with GA for this.

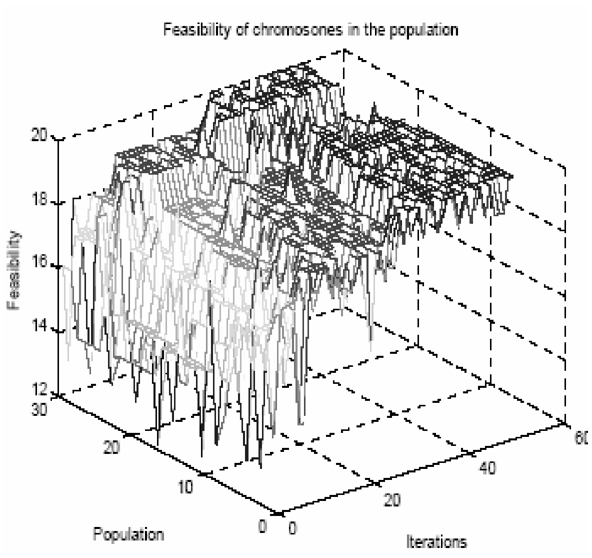

Figure 3. Feasibility value vs change of chromosomes in the population

For the study, the chromosome size was kept at 20. The length of the task queue considered was 100, like before. The algorithm was slightly modified to incorporate SCF heuristic. In the case where the tasks were sorted based on the deadline, the algorithm was modified so that the tasks were sorted based on their computation time. The tasks were sorted in the increasing order of computation time. The fitness function was not changed. It determines the number of tasks that can be scheduled without missing their deadline. It was seen that, the result was almost similar to that obtained in the case of using earliest deadline first, i.e., it gave almost similar performance.

It was then decided to change the length of the task queue while maintaining the chromosome size at 20 and the not altering anything else. The results were compared for the two cases, that is, using earliest deadline first and shortest computation time first. The task queue lengths considered were 100, 200, 400 and 600. The comparison of the heuristics has been made based on the fitness value. As the chromosome size has been fixed at 20, the maximum value for fitness that can be obtained is 20. It could be seen that for all the cases the number of tasks that were feasible was 90 percent and above for both the heuristics. This gives the impression that the heuristic shortest computation first could also be incorporated with GA to give feasible solutions. The graph of the comparison is shown in the Figure 4. The bar graph shown gives a better overview of the results discussed above.

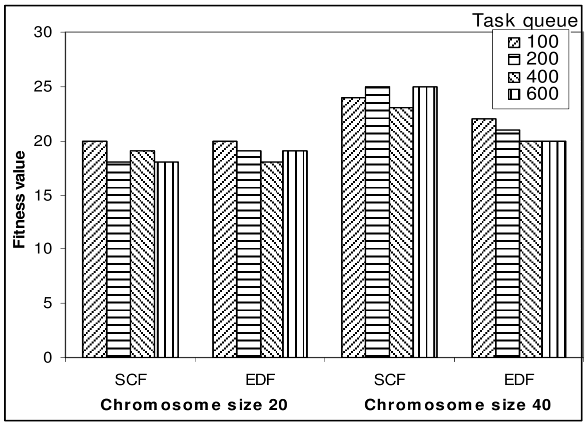

Figure 4. Comparative fitness values for chromosome sizes 20 and 40.

The results were then compared for task queues of different length by changing the chromosome size. The lengths of task queue considered were same as before namely, 100, 200, 400 and 600. The chromosome size chosen were 40 and 60. Though both the heuristics showed almost similar performance in the case of chromosome size 20, the result was not same for higher values of chromosome size. It could be seen that the use of heuristic shortest computation time first gave better fitness values compared to earliest deadline first when incorporated with GA. This shows that the heuristic shortest computation time first is a better option for incorporating with GA. The comparison graph which shows the heuristics based on fitness value for chromosome size 40 is shown in Figure 4.

A study of the fitness value obtained for different chromosome sizes has also been done. As mentioned before, the fitness value denotes the number of tasks that are feasible, out of a certain set of tasks considered for scheduling. It is not however possible to schedule all the tasks without missing deadlines when there are resource restrictions as here, the number of processors available is limited. The scheduling algorithm faces the challenge to schedule maximum number of tasks using the limited resources. It could be seen that as the chromosome size is increased the number of tasks that are feasible also increases slightly when the number of processors considered is fixed. This is the case when considering the heuristic shortest computation time first. When comparing the results, shown in Table 1 it could be seen that only 48 percent of the tasks could be scheduled when the chromosome size is 60, whereas in the case with chromosome size 20, nearly 100 percent of the tasks could be scheduled. It should be mentioned that the result considers a fixed number of processors, i.e. 10. Thus a comparative study shows that best results are obtained with chromosome size 20. It could also be noted that better results are obtained when the length of the task queue is 100.

From the above results it could be said that traditional scheduling heuristics could be incorporated with GA to schedule real-time tasks if the scheduling time used by GA is reduced by some efficient method.

Table 1. Fitness value obtained for different chromosome sizes.

Task queue	Fitness value			
	Chromosome size 20		Chromosome size 40	
	SCF	EDF	SCF	EDF
100	20	20	24	22
200	18	19	25	21
400	19	18	23	20
600	18	19	25	20

5. Conclusion and future work

Scheduling is an important topic that has applicability in a wide variety of domains. Generally, scheduling problems are NP-hard and there are no general algorithms that can guarantee an optimal solution. This is same in the case of scheduling real-time tasks as well. The widely used algorithms for scheduling real-time tasks have been discussed in the paper. A hybrid GA for scheduling tasks in multiprocessor system has been presented based on the work done by Mahmood [5]. The paper has discussed that GA incorporating traditional heuristics could be used to obtain feasible solutions. A comparative performance of using heuristics EDF and SCTF with GA has been presented and discussed through a set of experiments. It is noted that incorporating SCTF with GA offered better performance as compared to the EDF. The algorithm presented in the paper has been successful in obtaining feasible solutions for a task set of 20 and also achieving high utilization of processors. It is noted that the implementation of the GA is quite costly since populations of solutions are coupled with computation intensive fitness evaluations. This can be overcome by employing parallel processing technique in multiprocessor computing domain.

References

[1] Ramamritham, K. and Stankovic, J. A., 1994, Scheduling Algorithms and Operating Systems Support for Real-time Systems, Proceedings of IEEE, Vol.82, No.1, pp. 55-67.

[2] Cottet, F., Delacroix, J, Kaiser, C., Mammeri, Z. 2002, Scheduling in Real-time Systems, John Wiley & Sons Ltd, England, pp. 1-64.

[3] Eggers, E., January 1999, Dynamic Scheduling Algorithms in Real-time, Multiprocessor Systems, Term paper 1998-99, EECS Department, Milwaukee School of Engineering, North Broadway, Milwaukee, WI, USA.

[4] Manimaran, G., Siva Ram Murthy, C., March 1998, An Efficient Dynamic Scheduling Algorithm for Multiprocessor Real-time Systems, IEEE Transactions on Parallel and Distributed Systems, Vol.9, No.3, pp.312-319.

[5] Mahmood, A., 2000, A Hybrid Genetic Algorithm for Task Scheduling in Multiprocessor Real-Time Systems, Journal of Studies in Informatics and Control, Vol.9, No.3.

[6] Eggers, E., January 1999, Dynamic Scheduling Algorithms in Real-time, Multiprocessor Systems, Term paper 1998-99, EECS Department, Milwaukee School of Engineering, North Broadway, Milwaukee, WI, USA.

[7] Srinivasan, A. and Anderson, J., H., 2005, Fair scheduling of dynamic task systems on multiprocessors, The Journal of Systems and Software, Vol. 77, pp. 67-80.

[8] Nossal, R. and Galla, T., M., 1997, Solving NP-Complete Problems in Real-Time System Design by Multichromosome Genetic Algorithms. In Proceedings of the SIGPLAN 1997 Workshop on Languages, Compilers, and Tools for Real-Time Systems, pages 68-76, ACM SIGPLAN, June 1997

[9] Page, A., J. and Naughton, T., J., 2005, Dynamic task scheduling using genetic algorithms for heterogeneous distributed computing, The proceedings of the 19th International Parallel & Distributed Processing Symposium, Denver, USA. IEEE Computer Society.

[10] Manimaran, G., Siva Ram Murthy, C., November 1998, A Fault-tolerant Dynamic Scheduling Algorithm for Multiprocessor Real-time Systems and its Analysis, IEEE Transactions on Parallel and Distributed Systems, Vol.9, No.11, pp.1137-1152.