Author(s): Varghes, B., Hossain, M. A. and Dahal, K. P.

Title: Scheduling of tasks in multiprocessor system using hybrid genetic algorithms.

Publication year: 2007

Book title: Applications of soft computing.

ISBN: 978-3-540-88078-3

Publisher: Springer Verlag.

Original publication is available at *http://www.springerlink.com*

Citation: Varghes, B., Hossain, M. A. and Dahal, K. P. (2007) Scheduling of tasks in multiprocessor system using hybrid genetic algorithms. In: Kacprzyk, J.(ed.) Advances in soft computing: Updating the state of the art. (12th Online World Conference on Soft Computing in Industrial Applications. (WSC12) October 16th-26th, 2007).Berlin: Springer. pp. 65-74.

Scheduling of Tasks in Multiprocessor System using Hybrid Genetic Algorithms

Betzy Varghes, Alamgir Hossain, and Keshav Dahal

Modeling Optimization Scheduling And Intelligent Control (MOSAIC) Research Centre
Department of Computing, University of Bradford, Bradford, BD7 1DP, UK
betzymol@yahoo.com;{m.a.hossain1,k.p.dahal}@bradford.ac.uk

***Abstract* — This paper presents an investigation into the optimal scheduling of real-time tasks of a multiprocessor system using hybrid genetic algorithms (GAs). A comparative study of heuristic approaches such as 'Earliest Deadline First (EDF)' and 'Shortest Computation Time First (SCTF)' and genetic algorithm is explored and demonstrated. The results of the simulation study using MATLAB is presented and discussed. Finally, conclusions are drawn from the results obtained that genetic algorithm can be used for scheduling of real-time tasks to meet deadlines, in turn to obtain high processor utilization.**

Index Terms — Optimal scheduling, hard real-time tasks, multiprocessor system, heuristics, genetic algorithm.

1 Introduction

Optimal 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, 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]. In multiprocessor systems, the scheduling problem is to determine when and on which processor a given task executes.

Real-time task scheduling could be done either statically or dynamically. Dynamic schedule for a set of tasks is computed at run-time based on the tasks that is really executing. Static schedule on the other hand is done at compile time for all possible tasks. In the case of preemptive scheduling, an executing task may be pre-empted and the processor allocated to a task with higher priority or a more urgent task [2].

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.

A number of algorithms have been proposed for dynamic scheduling of real-time tasks. It is said that there does not exist an algorithm for optimally scheduling dynamically arriving tasks with or without mutual exclusion constraints [6]. This has motivated the need for heuristic approaches for solving the scheduling problem. Page and Naughton [7] gives a number of references in which artificial intelligence techniques are applied for task scheduling. They have also reported good results from the use of GAs in task scheduling algorithms.

This paper aims to provide an insight into scheduling real-time tasks by using genetic algorithm incorporating traditional scheduling heuristics to generate a feasible schedule based on the work done by Mahmood [5]. That is, the use of a hybrid genetic algorithm to dynamically schedule real-time tasks in multiprocessor systems. The scheduling algorithm considered, aims in meeting deadlines and achieving high utilization of processors. The paper also provides a comparative study of on applications of heuristic approaches, such as 'EDF' and 'SCTF separately, and 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. The MATLAB software tool was used for the simulation study as it integrates computation, visualization and programming in an easy to use environment.

2 Related Research Work

Scheduling algorithms for multiprocessor real-time systems are significantly more complex than the algorithms for uniprocessor systems [5]. In multiprocessor systems, the scheduling algorithm, other than specifying the ordering of tasks must also determine the specific processors to be used.

Goossens and others have been studying the scheduling of real-time systems using EDF scheduling upon uniform and identical multiprocessor platforms. They justify that EDF remains a good algorithm to use in multiprocessor systems. They also propose a new priority-driven scheduling algorithm for scheduling periodic task systems upon identical multiprocessors [8].

Manimaran and Siva Ram Murthy [4] says that there does not exist an algorithm for optimally scheduling dynamically arriving tasks with or without mutual exclusion constraints on a multiprocessor system. This has stimulated the need for developing heuristic approaches for solving the scheduling problem. In heuristic

scheduling algorithms, the heuristic function *H* evaluates various characteristics of tasks and acts as a decision aid for the real-time scheduling of the tasks.

Myopic scheduling algorithm is another heuristic scheduling algorithm for multiprocessor systems with resource constrained tasks. The algorithm selects a suitable process based on a heuristic function from a subset; referred to as window, of all ready processes instead of choosing from all available processes like the original heuristic scheduling algorithm. Another difference of the algorithm from the original heuristic algorithm is that the unscheduled tasks in the task set are always kept sorted by increasing order of deadlines. Hasan et al. [9] presents the impact of the performance in implementing the myopic algorithm for different window sizes.

Page and Naughton [7] gives a number of references to examples where artificial intelligence techniques are being applied to task scheduling. They say that techniques such as genetic algorithms are most applicable to the task scheduling problem because of the need to quickly search for a near optimal schedule out of all possible schedules. The paper presents a scheduling strategy which makes use of a genetic algorithm to dynamically schedule heterogeneous tasks on heterogeneous processors in a distributed system [7]. 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.

Oh and Wu [10] presents a method for real-time task scheduling in multiprocessor systems with multiple objectives. The objectives are to minimize the number of processors required and also minimize the total tardiness of the tasks. A multiobjective genetic algorithm has been made use of for scheduling to achieve optimization. The work considers scheduling tasks of precedence and timing constrained task graph. The algorithm was shown to give good performance. While Oh and Wu [10] focuses on multiobjective optimization, the algorithm discussed in the paper aims to meeting deadlines of tasks and achieving high resource utilization. There are also examples where genetic algorithm has been used for scheduling tasks in uniprocessor systems. Yoo and Gen [11] presents a scheduling algorithm for soft real-time systems. They have used proportion-based genetic algorithm and focused mainly on the scheduling of continuous tasks that are periodic and preemptive.

The scheduling algorithm presented in this paper is based on the work done by Mahmood [5]. The genetic algorithms in their purest form could be called as blind procedures. They do not make use of any problem specific knowledge which may speed up the search or which may lead to a better solution. That is why specialized techniques which make use of problem specific knowledge out-performs genetic algorithms in both speed and accuracy. Therefore it may be advantageous to exploit the global perspective of the genetic algorithm and the convergence of the problem specific techniques.

3 System Model

As discussed earlier, dynamically scheduling tasks in a multiprocessor system using a hybrid genetic algorithm presented in the following sections is based on the principle of the work done by Mahmood [5]. The task and scheduler model for the simulation system considered is discussed below.

Task Model: 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.

Scheduler Model: As discussed before the dynamic scheduling in a multiprocessor system could be either centralized or distributed. This paper assumes a centralized scheduling scheme with each processor executing the tasks that fill its dispatch queue. Since a centralized scheduling scheme is considered, all the tasks arrive at a central processor called the scheduler. The scheduler has a task queue associated with it to hold the newly arriving tasks. Thus 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. Thus the processing power can be utilized without making it idle.

The minimum capacity of the dispatch queues depends on factors like the worst case time complexity of the scheduler to schedule newly arriving tasks [6]. A feasible schedule is determined by the scheduler based on the worst case computation time of tasks satisfying their timing constraints.

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 may arrive while scheduling the current task set.

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.

4 The Scheduling Algorithm

A hybrid genetic algorithm for scheduling real-time tasks in multiprocessor system is discussed in this section. 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 length of the chromosome depends on the number of tasks selected from the sorted list. 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. The fitness value of a chromosome is the number of tasks in the chromosome that can meet their deadlines (i.e. the objective is to maximize the number of tasks in each chromosome that meet their deadlines). The chromosomes in the population are then sorted in the descending order of their fitness value.

Genetic operators are then applied to the population of chromosomes until a maximum number of iterations have been completed. When applying genetic operators to the population, selection 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 tasks that are found infeasible are removed from the chromosomes so that they are not reconsidered for scheduling. For a task T_i to be feasible it should satisfy the condition that $(R_i \leq st(T_i) \leq D_i - C_i)$ and $(R_i + C_i \leq ft(T_i) \leq D_i)$ where R_i is the ready time, D_i is the deadline and C_i is the worst case computation time of task T_i . $st(T_i)$ and $ft(T_i)$ denoted the start time and finish time of task T_i respectively. If the condition is not satisfied it is said to be infeasible.

5 Implementation and Results

The simulation study (using MATLAB) 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, C_i, of a task T_i 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 T_i has been randomly chosen between ($R_i + 2 * C_i$) and ($R_i + r * C_i$) 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 genetic 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, i.e., 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 genetic 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 genetic algorithm. The algorithm was then implemented by incorporating the heuristic 'Shortest Computation time First' with genetic algorithm. Set of tasks were scheduled using the modified algorithm and the results were observed.

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 1. 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.

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 genetic algorithm. 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 shown in Figure 2.

As shown by the graph, a better performance is obtained by using genetic algorithm 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. These demonstrate that genetic algorithm could be used to schedule task to meet deadlines and also achieve better processor utilization. However, it is worth noting that genetic algorithms do have the disadvantage of spending much time in scheduling.

As mentioned earlier in the paper, the population size for the genetic algorithm 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.

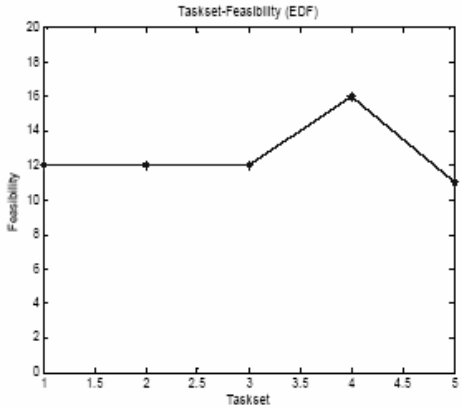

Fig. 1. Task set feasibility based on EDF

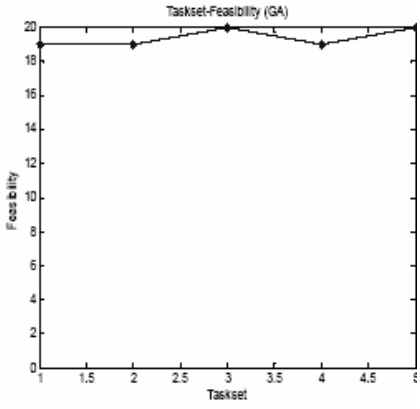

Fig. 2. Task set feasibility using hybrid GA

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 genetic algorithm for a good number of iterations. The number of iterations needed for the genetic 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 genetic algorithm 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 genetic algorithm for this.

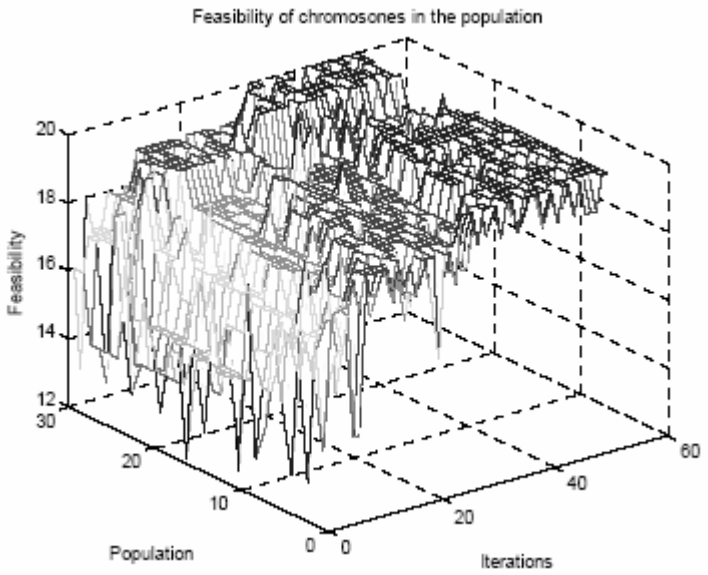

Fig. 3. Feasibility value vs change of chromosomes in the population

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. That is to say, 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 genetic algorithm to give feasible solutions. The graph of the comparison is shown in the Figure 4. This demonstrates a better overview of the results discussed above.

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 genetic algorithm. This shows that the heuristic shortest computation time first is a better option for incorporating with genetic algorithm. Fig. 5 shows the comparison of the heuristics based on fitness value for chromosome size 40.

Table 1 shows the comparative fitness function of SCF and EDF. It is noted 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.

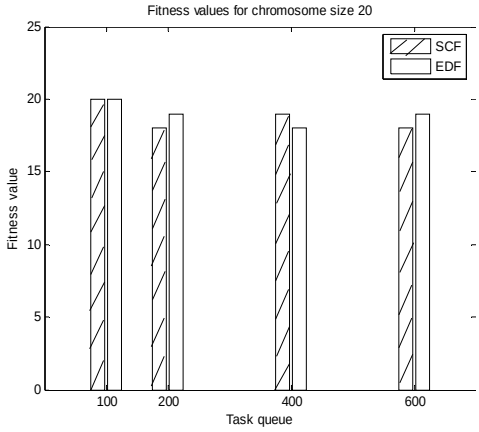

Fig. 4: Comparative fitness values for chromosome size 20

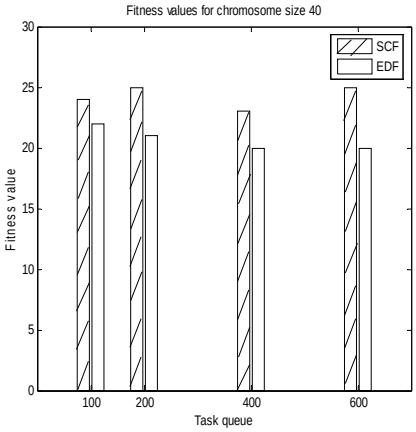

Fig. 5. Comparative fitness values for chromosome size 40

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

Table1. Fitness value obtained for different Chromosome size

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

6 Conclusion

A hybrid genetic algorithm for scheduling tasks in multiprocessor system has been presented based on the work done by Mahmood [5]. The paper has discussed that genetic algorithm incorporating traditional heuristics could be used to obtain optimal solutions. A comparative performance of using heuristics EDF and SCTF with genetic algorithm has been presented and discussed through a set of experiments. It is noted that incorporating SCTF with genetic algorithm 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.

However it is noted that the implementation of the genetic algorithm is quite costly since populations of solutions are coupled with computation intensive fitness evaluations. This can be overcome by employing high performance computing platform or 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.http://www.ici.ro/; accessed on 27/06/2005.

[6] 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.

[7] Page, A., J. and Naughton, T., J., April 2005, *Dynamic task scheduling using genetic algorithms for heterogeneous distributed computing*, 8th International Workshop on Nature Inspired Distributed Computing, proceedings of the 19th International Parallel & Distributed Processing Symposium, Denver, Colorado, USA. IEEE Computer Society. 27 July 2005

[8] Goossens, J., Baruah, S. & Funk, S., 2002, Real-time Scheduling on Multiprocessors. <http://citeseer.ist.psu.edu/; accessed on 12/08/05.

[9] Hasan, M., S., Muheimin-Us-Sak, K. & Hossain, M., A., 2005, *Hard Real-Time Constraints in Implementing the Myopic Scheduling Algorithm*. International Journal of High Performance Computing Applications, SAGE Publications (to appear)

[10] Oh, J. and Wu, C., May 2004, *Genetic-algorithm-based real-time task scheduling with multiple goals*, The Journal of Systems and Software, Volume 71, Issue 3, pp. 245-258.

[11] Yoo, M., R. and Gen, M., 2001, *Bicriteria real-time tasks scheduling using proportion-based genetic algorithm,* 15 Aug. 2005, pp. 213-222. http://www.complexity.org.au /conference/upload/yoo01/yoo01.pdf