An Evolutionary Approach to Multiprocessor Scheduling of Dependent Tasks[⋆]

Roman Nossal

Vienna University of Technology, Treitlstr. 3/182-1, A-1040 Vienna, Austria
nossal@vmars.tuwien.ac.at

Abstract. The scheduling of application tasks is a problem that occurs in all multiprocessor systems. This problem becomes even more complicated if the tasks are not independent but are interrelated by mutual exclusion and precedence constraints.
This paper presents an approach for pre-runtime scheduling of periodic tasks on multiple processors for a real-time system that must meet hard deadlines. The tasks can be related to each other by mutual exclusion and precedence forming an acyclic graph. The proposed scheduler is based on Genetic Algorithms, which relieves the user from knowing how to construct a solution. Consequently, the paper focuses on the problem encoding, i.e., the representation of the problem by genes and chromosomes, and the derivation of an appropriate fitness function. The main benefit of the approach is that it is scalable to any number of processors and can easily be extended to incorporate further requirements.

1 Introduction

For real-time systems deployed in safety-critical application domains, a design based on a guaranteed-response rather than a best-effort paradigm has become common practice. One means to achieve a guaranteed behavior of the system even under peak load and fault conditions is static scheduling. With static scheduling the execution schedules for all processors are determined off-line.

In the past few years a lot of research related to multiprocessor scheduling for real-time systems has been done. From the vast number of papers on this topic the following selects some representative approaches. More information can be obtained from a survey, e.g., by Ramamritham et al. [13], or the one by Xu and Parnas [17], who focus on scheduling for hard real-time systems. Ramamritham [12] presents an algorithm based on a clustering heuristics for allocating and scheduling periodic tasks on nodes connected by a predictable network. DiNatale and Stankovic [2] use Simulated Annealing to schedule tasks that are related by precedence and mutual exclusion on multiple processors. Hou, Ren, and Ansari [5] base their algorithm, which is not intended for real-time problems, on Genetic Algorithms.

In this paper we present an algorithm that is capable of scheduling tasks that are interrelated by mutual exclusion and precedence constraints on multiple

[⋆] This work was in part supported by HP Labs, Palo Alto, California.

processors. We assume that the tasks have a given processor allocation. The problem of preemptively scheduling a set of tasks with arbitrary computation times on only two processors is known to be NP-complete [3]. The problem here is even more complicated since the number of processors may be larger than two and the tasks are not independent. Consequently, heuristic approaches have to be applied in order to solve the scheduling problem in polynomial time.

The algorithm presented in this paper is based on the heuristic optimization technique Genetic Algorithms (GA) [4]. GAs offer a number of advantageous features compared to other heuristic techniques. In particular, it is sufficient to be able to assess a given solution; there is no need to specify how solutions are constructed. The proposed scheduler tries to minimize the lateness of the entire real-time application while observing the periods and the inter-task relations.

The rest of the paper is organized as follows. The next section presents the system architecture and the task model. Section 3 gives an introduction to Genetic Algorithms and explains the scheduling algorithm. Special attention is drawn to the problem encoding and the fitness function. Section 4 elaborates on the advantages of a GA-based scheduler. The paper is concluded in Sect. 5.

2 System Architecture and Task Model

We assume a computer system that consists of N identical processors that are interconnected by a communication link. The algorithm presented in this paper does not explicitly take into account the effects of communication between the processors. A comprehensive treatment of the overall scheduling algorithm that also deals with communication is contained in [9]. The message scheduling algorithm itself, which is also based on Genetic Algorithms, is described in [10].

The envisioned real-time application comprises one or more *extended precedence graphs (EPG)*. An EPG is a time-constrained chain of interrelated tasks that form an acyclic graph. Each EPG starts with a stimulus task and ends with a response task. The *temporal constraints* of an EPG are the period p_{epg} at which the EPG must be executed and the deadline dl_{epg} applying to the EPG.

An EPG comprises several *simple tasks* [6, p.75], i.e., tasks that do not have any synchronization points in their bodies. A simple task executes from the time it is started to the time it finishes without being blocked. Hence the worst-case execution time of the task depends only on its structure. As static task scheduling is applied in the architecture, the *worst-case execution time (WCET)* must be known and bounded. Besides its worst-case execution time denoted by $wcet_t$, a task is characterized by its static allocation to a processor.

Tasks in the proposed task model are *preemptive*. A task may be interrupted by another task at any time during its execution. If certain parts of a task, so-called *critical sections*, must not be preempted by another task, the respective tasks must be related to each other by a mutual exclusion relation.

All tasks of an EPG are related to each other by two types of relations, the above mentioned mutual exclusion and precedence relations. A variant of precedence relations is the so-called *timed precedence*. Timed precedence specifies a certain distance between the executions of the tasks the relation connects. This interval can be semi-open or closed.

The scheduler tries to build a task schedule for each processor that satisfies the given task relations while at the same time minimizing the lateness of the EPGs. Since the algorithm is intended for real-time scheduling the lateness of the entire application should ideally be 0, i.e., no EPG misses its deadline.

3 The Scheduling Algorithm

3.1 Introduction to Genetic Algorithms

Genetic Algorithms (GA) [4] are a heuristic method used to solve NP-hard optimization problems in many fields. GAs constitute a so-called "uninformed" search strategy. This term refers to the fact that the algorithm itself does not have any knowledge on the problem it solves. The problem specific knowledge is entirely incorporated into the *fitness function*. This function calculates a basic *fitness value* of a candidate solution produced by the GA according to an optimization criterion; additional constraints can be incorporated into the function by so-called *penalty terms*. A penalty term reduces the fitness of a solution provided that it violates a constraint. The resulting, overall fitness value is used as a feedback to the algorithm.

A problem to be solved by a GA has to be *encoded* in an appropriate way, because GAs act upon bit strings. Each bit string encodes one part of the information on the problem solution and is called a *gene*. Genes are grouped to *chromosomes*; one or more chromosomes form an *individual*. Each individual encodes a complete solution to the given problem. The quality of this solution, i.e., the fitness of the individual is assessed by the fitness function. A certain number of individuals form a *population*.

The GA imitates nature's process of evolution by taking one population as parent generation and creating an offspring generation. The algorithm selects the best, i.e., fittest, individuals of the parent population. Here the fitness of an individual comes into play. The better the fitness value of an individual, the greater is the probability that it is selected for reproduction. After the selection the GA mutates some of the genes of the selected individuals by flipping bits according to a given mutation rate and performs cross-over between the individuals by swapping parts of chromosomes. As a result of this process a new population, a child generation, is created, which is then evaluated. The whole process iterates until a solution of sufficient quality is found.

While the GA is capable of generating new solutions to a problem that has been properly encoded, the derivation of an encoding scheme and the selection of a fitness function are up to the user.

For a comprehensive description of GAs and their function the reader is referred to [15]. A GA that has been developed for the use as the basis of the scheduler is presented in [11].

3.2 Scheduler Implementation

The research on scheduling for real-time systems resulted in a number of algorithms intended for on-line scheduling, e.g., rate monotonic scheduling, deadline driven scheduling [7], or the least-laxity first algorithm [8].

The scheduler proposed in this paper was designed for static scheduling, nevertheless it incorporates a dynamic scheduling algorithm. A GA is used to derive an input sequence for a dynamic scheduling algorithm, which in turn creates the actual schedule. This approach was inspired by the work of Davis [1] and Smith [14], who proposed similar techniques for bin packing and job shop scheduling.

Problem Encoding Reviewing the discussion of GAs, one finds that a GA requires two features to be defined by the user, a problem encoding and a fitness function.

The problem encoding for the task scheduling GA represents the input sequence to a dynamic scheduling algorithm. Additional information on the structure of an EPG, is also incorporated into the encoding scheme. The input sequence is recombined by the genetic operators, creating new sequences. Before going into more detail three important notions are defined:

1. The *ready time* is the point in time, at which a task becomes ready for execution. A ready task can be executed whenever the CPU is available.
2. The *release time* of an EPG is the time at which the first task of an EPG becomes ready at the earliest. The deadline of the EPG is calculated relative to this point in time.
3. The *execution interval* of a task is the interval of time, during which the task may be executed. When exactly a task is scheduled during its execution interval is up to the scheduler.

The possible execution interval of a task is calculated off-line. Only if all tasks are scheduled in their respective execution intervals the resulting schedule can be feasible. The length of the execution interval reduced by the WCET of the task is referred to as *laxity* of the task. Four points in time determine the execution interval of a task, (1) the earliest start time (EST), (2) the earliest finish time (EFT), (3) the latest start time (LST), and (4) the latest finish time (LFT). Once the execution intervals have been determined the scheduling problems for the individual processors are no longer linked. The schedule for each processor can be created separately. Only when assessing the fulfillment of the inter-processor relation constraints the ensemble of processors must be regarded as a whole again. The use of execution intervals for the task scheduling process resembles the windows-based approach proposed by Verhoosel [16].

The principle of the scheduler is to give the GA control over the release time of the first EPG instance and over the ready times of the tasks of the EPG. Firstly, the GA determines when the first instance of the EPG is released. Secondly, the scheduler selects the actual ready time for each task from the interval $[EST, LST]$, calculates the remaining laxity of the task and then applies least-laxity scheduling to the entire task set. The least-laxity scheduler creates the final schedule, which contains the actual start time of each task.

Derivation of the Execution Intervals. An EPG can contain two types of relations, precedence (timed or untimed) and mutual exclusion. For the problem encoding only precedence relations are considered. Mutual exclusion constraints are not dealt with in the problem encoding, their fulfillment is checked in the final schedule.

Depending on their timing requirements, five types of precedence relations can be distinguished, untimed, timed with lower bound on the task distance, timed with upper bound on the task distance, timed with both upper and lower bound, and timed with exact task distance. The task distance is the interval of time between the end of the task, from which the precedence relation originates, and the start of the task, at which the relation ends.

As the actual release time of an EPG is chosen by the task scheduler, it is not available during derivation of the execution intervals. Hence the execution intervals are calculated relative to an assumed release time of 0. The GA thus also selects relative ready times for the tasks, which are converted to absolute ready times during the construction of a schedule.

First the earliest start times and the earliest finish times of the tasks are determined by traversing the task graph from the left to the right. Next the latest finish and the latest start times are calculated; for this the task graph is traversed from the right to the left. The earliest and latest start and finish times of a task are derived using the following formulas:

$$
\begin{aligned}
EFT_x &= EST_x + wcet_x && (1)\\
EST_{stim} &= 0 && (2)\\
EST_x &= \max_{t \in PRED_x} (EFT_t + lb_{tx}) \qquad lb \in N_0 && (3)\\
LST_x &= LFT_x - wcet_x && (4)\\
LFT_{resp} &= dl_{epg} && (5)\\
LFT_x &= \min_{t \in SUCC_x} (LST_t - lb_{tx}) \qquad lb \in N_0 \ . && (6)
\end{aligned}
$$

$PRED_x$ is the set of all tasks immediately preceding task x, and $SUCC_x$ the set of tasks immediately succeeding x. lb_{tx} denotes the lower bound for the time distance of tasks t and x, $stim$ and $resp$ represent a stimulus and a response task of the EPG, respectively.

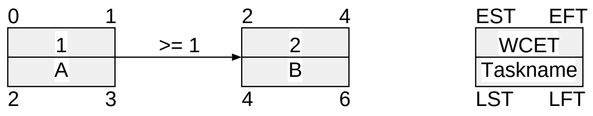

Fig. 1. An Example of a Timed Precedence Relation with a Lower Bound.

Figure 1 illustrates a precedence relation featuring a lower bound on the task distance. Note that a lower bound is considered during the calculation of the execution intervals, whereas an upper bound is not taken into account. It comes into play after a schedule has been constructed and is assessed for its constraint fulfillment.

Encoding Scheme. The above-derived execution intervals are the basis for the task scheduler. From the execution intervals the range of possible ready times for each task and the range of release times for each EPG are known. Both the ready time-ranges and the release times are encoded for use by the GA. The encoding scheme is explained in a top-down manner, starting from an individual to the meaning of a gene.

Individual. The entire population of the GA represents a set of potential task schedules. Each individual in turn encodes one candidate solution to the task scheduling problem, i.e., one task schedule. An individual consists of as many chromosomes as EPG instances are allocated to the processors. The number of instances of an EPG is derived by dividing the schedule length, i.e., the LCM of all EPG periods, by the period of the EPG under consideration.

Chromosome. One chromosome encodes the timing of one EPG instance. The number of genes of a chromosome is equal to the number of tasks of the EPG instance plus one gene for the determination of the release time of the EPG.

Gene. The genes of a chromosome encode the release time of the EPG and the ready times of the tasks respectively.
The release time of the first instance of an EPG can be chosen freely in the interval $[0, p_{epg}]$, since the EPG is repeated periodically with its period p_{epg}. A release time greater than p_{epg} would thus mean that there is a prior instance of the EPG that is released in the interval $[0, p_{epg}]$. For all instances of the EPG but the first the release time is calculated by adding the period of the EPG to the release time of the first instance.

Fitness Function The evolutionary process inherent to GAs creates new solutions in each generation. The GA determines an actual absolute release time for each EPG and an actual relative ready time for each task. Each of these solutions, which is encoded by one individual, is handed to the fitness function, which conducts the assessment in four steps:

1. Calculating the Absolute Ready Times of Tasks: For the first EPG instance the first gene of the chromosome encoding the EPG is interpreted as absolute release time of the EPG, whereas for the other instances the first gene is ignored, because their release time is the release time of the first instance plus the EPG period. The other genes are the relative ready times of the tasks, from which the absolute ready times are derived by adding the absolute release time of the respective EPG instance.

After that, the remaining laxity for each task is calculated as the difference of the LST and the ready time of the task.

2. Creating the Sets of Ready Tasks: Based on the knowledge of the ready time of each task the set of tasks becoming ready at each granule of the time base is created. This set comprises tasks of all EPGs allocated to a processor. Within each set the tasks are sorted by their laxity. This and the following step are carried out separately for each processor.

3. Least-Laxity Scheduling: Now the actual scheduling algorithm comes into play. It traverses the length of the scheduling interval, i.e., the LCM of all EPG periods. For each granule of the time base in this interval it builds the current set of ready tasks and selects the task with the smallest laxity for execution. For each ready task that is not executing the remaining latency is reduced by one time unit.

As a result each least-laxity scheduler produces a complete task schedule for one processor. The schedule accommodates all tasks, but it does not guarantee the fulfillment of the constraint imposed by the EPG. The least-laxity scheduler can of course be replaced by any other single-processor task scheduling approach that is capable of real-time scheduling, e.g., earliest deadline first.

4. Assessment of the Solution: The fulfillment of the temporal and functional constraints of the final schedule is assessed. This step considers all processors at the same time, because of the inter-processor relations. Note that the assessment can easily be done, since the result of the preceding steps is a complete schedule for all processors that contains full information on the task execution times.

The main optimization goal is the lateness of the EPGs. The lateness is the actual point in time when the last task of an EPG finishes compared to the absolute deadline. If this time is exceeded, the difference between the actual finish time and the deadline is the fitness value of the solution. This calculation is undertaken for all EPGs. The GA tries to minimize the resulting fitness value down to an optimum of 0.

Up to this point the fitness value does not account for the task relations of the EPG, namely mutual exclusion and precedence. Mutual exclusion was not considered in the problem encoding. The precedence constraints, on the other hand, influenced the calculation of the execution intervals, nevertheless they can be violated for the following reason: The problem encoding represents an execution interval for each task, but these intervals can overlap, since they comprise the "earliest" as well as the "latest" case. Thus scheduling all tasks in their execution intervals does not guarantee a correct task sequence.

For each mutual exclusion and precedence relation the actual task sequence in the schedule is considered. This check can be done regardless whether the involved tasks reside on one processor or are allocated to different processors. If the actual task sequence violates the constraint, a *penalty* is added to the fitness value of the schedule.

Stopping Criterion The task scheduler may stop its execution if a feasible schedule is found, i.e., a schedule that meets all temporal requirements of the EPG and that does not violate any relations. Obviously a fitness value of 0 for a schedule stands for such a schedule. In this case the lateness of all EPGs is 0 and no penalties apply. If an EPG was late, its lateness would be added to the fitness value. The same holds true for constraint violations, for which a fixed value is added to the fitness.

Furthermore, a second stopping criterion is employed. If the number of generations exceeds a certain limit without having produced a feasible schedule, the algorithm stops.

4 Assessment of the Approach

A number of different optimization techniques can be applied to solve NP-complete scheduling problems. The following summarizes the benefits of using GAs instead of other heuristic problem solving techniques.

Solution Evaluation Instead Of Construction. The most obvious benefit is that GAs relieve the user of knowing how to construct a solution. In order to implement a "standard" problem solving technique the user is required to know how to construct a solution to a problem. There has to be an algorithm that yields a valid solution to the given problem.

The situation is completely different when using GAs. GAs treat the problem in an encoded form, and operate on bit strings. Thus a GA constructs a solution by rearranging bit strings. These construction steps are not defined by the user, but are inherent to the GA. It is the task of the evaluation function to decode and to assess the solution. This means that the user has to provide knowledge on how to evaluate a given solution.

Assessing a solution instead of constructing it also eases the implementation of further constraints on the task schedule. A new constraint simply adds another penalty term to the fitness function. Thus it is accounted for automatically.

Integration of Various Problems. GAs offer a simple way to integrate different search procedures into one optimization process yielding a global optimum instead of local optima for each search. Each of the involved problems is encoded into a number of chromosomes and has its own fitness function. The problems can be solved in one step by (1) constructing an individual that consists of the chromosomes of all involved problems and (2) by deriving an appropriate algebraic conjunction of the fitness functions.

Taking advantage of the scalability feature of GAs the scheduler described in this paper can be applied to single processor scheduling as well as multiprocessor problems without altering the fitness function and the problem encoding.

5 Conclusion

In this paper we have presented an algorithm for multiprocessor scheduling of dependent, periodic tasks. The algorithm is based on Genetic Algorithms.

The scheduling problem is encoded by deriving execution intervals for the tasks, which determine the temporal boundaries for the execution points in time. The Genetic Algorithm selects the actual start time for each task from within the interval. In the fitness function the resulting task invocation times are used as an input to a least-laxity scheduler, which creates the schedules for all processors. The schedule is then assessed with regard to the fulfillment of the deadlines of the tasks and the inter task relations.

The scheduler has been successfully applied in the Time-Triggered Architecture (TTA), a distributed real-time system. It is used in conjunction with a GA-based message scheduler to plan the execution of task sets on the nodes of the system.

References

1. L. Davis. Job-Shop Scheduling with Genetic Algorithms. In J. Grefenstette, editor, *Proceedings of the 1st International Conference on Genetic Algorithms and Their Applications*, pages 136–140. Lawrence Erlbaum, 1985.
2. M. DiNatale and J.A. Stankovic. Applicability of Simulated Annealing Methods to Real-Time Scheduling and Jitter Control. In *Proc. 16th Real-Time Systems Symposium*, pages 190–199. IEEE Computer Society Press, 1995.
3. M.R. Garey and D.S. Johnson. *Computers and Intractability – A Guide to the Theory of NP-completeness.* W.H. Freeman Co., San Francisco, CA, 1979.
4. J.H. Holland. *Adaptation in Natural and Artificial Systems.* The University of Michigan Press, Ann Arbor, 1975.
5. E.S.H. Hou, N. Ansari, and H. Ren. A Genetic Algorithm for Multiprocessor Scheduling. *IEEE Transactions on Parallel and Distributed Systems*, 5(2):113–120, February 1994.
6. H. Kopetz. *Real-Time Systems, Design Principles for Distributed Embedded Applications.* Kluwer Academic Publishers, Norwell, Massachusetts, 1997.
7. C.L. Liu and J.W. Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *Journal of the ACM*, 20(1):46–61, January 1973.
8. A.K. Mok. *Fundamental Design Problems of Distributed Systems for the Hard-Real-Time Environment.* PhD thesis, Department of Electrical Engineering and Computer Science, MIT, Cambridge, MA, May 1983.
9. R. Nossal. *An Interface-Focused Methodology for the Development of Time-Triggered Real-Time Systems Considering Organizational Constraints.* PhD thesis, Technisch-Naturwissenschaftliche Fakultät, Technische Universität Wien, Vienna, Austria, October 1997.
10. R. Nossal. Static Message Scheduling for a TDMA-Based Real-Time Communication System. Submitted for Publication at the 6th International Workshop on Parallel and Distributed Real-Time Systems, Orlando, Florida, March 1998.
11. R. Nossal and T.M. Galla. 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.
12. K. Ramamritham. Allocation and Scheduling of Precedence-Related Periodic Tasks. *IEEE Trans. on Parallel and Distributed Systems*, 6(4):412–420, April 1995.
13. K. Ramamritham and J.A. Stankovic. Scheduling Algorithms and Operating Systems Support for Real-Time Systems. *Proceedings of the IEEE*, 82(1):55–67, January 1994.
14. D. Smith. Bin Packing with Adaptive Search. In J. Grefenstette, editor, *Proceedings of the 1st International Conference on Genetic Algorithms and Their Applications*, pages 202–207. Lawrence Erlbaum, 1985.
15. M. Srinivas and L.M. Patnaik. Genetic Algorithms: A Survey. *IEEE Computer*, pages 17–26, June 1994.
16. J. Verhoosel. Deterministic Scheduling of Distributed Hard Real-Time Systems using Windows. In *Proceedings 1st IEEE Workshop on Parallel and Distributed Real-Time Systems*, pages 252–256, April 1993.
17. J. Xu and D.L. Parnas. On Satisfying Timing Constraints in Hard-Real-Time Systems. *IEEE Transactions on Software Engineering*, 19(1):70–84, January 1993.