Niched Genetic Algorithms: a massive parallel approach to Schedulability Checking in Real-Time Systems

Renzo Davoli* **Fabio Tamburini**** **Andrea Frascari***
E-mail: {davoli,tamburin,frascari}@cs.unibo.it

Abstract

Si discutono gli Algoritmi Genetici (GAs) paralleli come strategia di ricerca per risolvere problemi di controllo dello scheduling in Sistemi Hard Real Time. La nostra attenzione sarà rivolta a fattori quali *la dimensione della popolazione, il mantenimento della diversità all'interno della popolazione stessa e l'indipendenza dalla generazione iniziale*. Questi fattori diventano ancora più cruciali quando si implementa una versione parallela di algoritmo genetico poiché la popolazione di individui è in qualche modo divisa fra i processori a disposizione. In particolare, vediamo come uno schema *a nicchie* sia in grado di sfruttare l'intrinseco parallelismo tipico degli algoritmi genetici.

Introduction

It is widely agreed that Genetic Algorithms are characterised by a degree of intrinsic parallelism[12],[6]. Nevertheless, attention should be paid to how to exploit this interesting feature. Since GAs' intrinsic parallelism mainly comes from the natural independence of the genetic manipulation operators, one could expect to achieve a good result simply by managing a unique population on a single master processor, and using parallel worker processes to execute cross-over and mutation operations. However, a closer look at the schema shows that, given such a low granularity of parallelism, communication costs would be predominant. We shall therefore consider another paradigm, which we will refer to as the Niched Parallel Genetic Algorithm (NPGA). In this paradigm, the population is sub-divided among the processors. Communication costs are in this case not predominant.

Our theoretical discussion will be based on the concept of *maintenance of diversity within the population*. GAs capacity not to be trapped into local minima of the Energy function is due to the fact that a population of diverse individuals can hold the whole information necessary to solve a particular problem. Lack of diversity leads to loss of information. This partly conflicts with the GAs typical strategy of prizing high-fitness individuals and eliminating low-fitness ones[6],[7], since the latter can hold important information, the loss of which could cause the GA to be trapped into local minima[1].

Although an analysis of GAs convergence is beyond the scope of this paper, it is our opinion that one should not expect a GA to end up with a population composed of several instances of a single high-fitness individual, since it would be difficult to show that such an individual represents a global minimum for non-trivial energy functions. Rather, it is our goal to develop a parallel GA able to evolve a generation of diverse individuals and to prevent each of them from taking over the entire population.

Maintenance of diversity is even more important when it comes to Parallel GAs, since the speed-up obtained is generally due to population decrease (achieved by assigning a sub-population to each available processor). Care should therefore be taken, as small sub-populations often imply a lack of information. A problem still arises as to ascertain whether a particular solution is optimal or not. However, since this paper focuses on evaluating GA parallelism rather than on providing GA convergence proofs, we have taken the previous considerations as a basis on which to develop our parallel algorithm, and chosen a problem for which optimal solutions have energy equal to zero, and are therefore easily recognisable.

---

* Department of Computer Science, University of Bologna - Italy.
** CILTA, University of Bologna - Italy.

[1]Throughout the paper, when referring to local and global minima, we intend local and global minima of the Energy function.

The problem is the *schedulability checking* of a set of parallel programs in a Real-Time system. The formal model of the problem was first introduced in [2]. This paper extends this research work considering a Parallel Genetic Algorithm. The remaining part of the paper is organised as follows: Section 2 describes the problem statement and the resulting Energy function that will be applied to the Parallel GA to determine the fitness of each individual. Sections 3 will describe and discuss the Niched Parallel Genetic Algorithm and Section 4 will show the experimental results.

Problem statement

The correctness of a computation in a Real-Time System[14],[15],[16] depends both on the results and on the time in which these results are produced. In particular, in Hard-Real-Time systems time constraints are crucial deadlines and it is compulsory for the system to respect them. The problem we are trying to solve is schedulability checking for a set of parallel programs in a Parallel Hard Real Time System. Programs are represented as Coarse Grain Data Flow (GCDF) graphs[2], whose edges represent dependences (time constraints and data exchange) among the procedures making up the programs, and whose nodes are the procedures themselves. Programs are therefore not atomic entities: they consist of procedures (nodes) that may be executed concurrently (if allowed by the precedences expressed in the graph), possibly in a parallel manner by different processors.
The problem we address is to find out whether a set of periodic HRT SG CGDF[2] programs can be scheduled in a feasible way on a pre-defined set of processors. The solution is given in terms of a mapping function that assigns each procedure to a processor (without overloading the latter and respecting all the time constraints), and gives each procedure its maximum execution time. Each procedure contributes to the load of the processor to which it is assigned. The scheduling policy assumed is pre-emptive (Earliest Deadline First). The programming environment we are considering is uniform multi-processor and loosely coupled. The conditions that our mapping function must fulfill can be summarised as follows: 1) Time specifications are positive; 2) All programs complete their jobs within their respective deadlines; 3) No processor is overloaded; 4) Each node is assigned to a single processor.
We shall now describe a mathematical formulation of the problem and deduce an energy function for the Genetic Algorithm. Let us consider the following notation:

- $n_1,\ldots,n_{NNODE}$ = all program nodes
- $c_1,\ldots,c_{NNODE}$ = computational costs

c_i represents the time for node n_i to be executed on a single dedicated processor, regarding its algorithm's worst case: the n_i effective computation time clearly depends on the input data. It generally varies between different instances. It is non-negative (it may be zero), but in no case can it exceed c_i.

- $P_1,\ldots,P_{NPROC}$ = system processors
- $J_1,\ldots,J_{NPGMS}$ = programs (Data Flow Graphs)

Each activation of J_i is to be completed within D_i time units. Moreover, D_i is also the minimum time between two sequential activations of J_i.
The problem solution is (if any) a vector $\bar{x}$ of time assignment values $x_1,\ldots,x_{NNODE}$ and a mapping function *map*:[1...NNODE] → [1...NPROC] that binds each node to a specific processor, such that:

$$\begin{cases} x_k > 0 & \forall k = 1,\ldots,NNODE \\ Time(\bar{x}, J_i) \le Di & \forall i = 1,\ldots,NPGMS \\ \sum_{i=1}^{NPGMS} Load(\bar{x}, J_i | map^{-1}(Pj)) \le 1 & \forall j = 1,\ldots,NPROC \end{cases}$$

$J_i|map^{-1}(P_j)$ is the restriction of program J_i to the set $map^{-1}(P_j)$; it selects the program substructure consisting of the nodes that are mapped on the same processor P_j, maintaining the precedence relation of the original graph of J(i). The *Time* function computes the maximum execution time for program J_i given a vector $\bar{x}$ of node time assignments: it adds the component of all the sequential nodes and takes the maximum value between concurrent

---

[2]Hard Real Time - Scatter Gather - Coarse Grain Data Flow

ones. The *Load* function computes the load for processor P_j executing the nodes of program J_i mapped on it. Each node load is obtained as the ratio c_k / x_k: this is the real amount of processor load necessary for each node n_k to perform its job. We shall now show how to express this non-linear equation system in terms of a single energy function applicable to a GA. Suppose each individual in the population is represented by a bit-string genoma subdivided into two sections: the first part holds the mapping information while the second part includes the processor load assignment for each task. Taking $\mu_{j,k}$ to be 1 if the task n_k is assigned to P_j and 0 if it is not, the mapping information is coded as the bit stream $\mu_{j,k}$ k=1,...,NNODE j=1,...,NPROC i.e. a row-by-row scan of the matrix μ. The load information is also coded as a bit-stream. Each task load is represented by a word, whose ***i-th*** bit represents a load of $\frac{1}{2^i}$. In the following, we will refer to the load part of the genoma as λ; λ_i represents the load assigned for the computation of node n_i. Let us name $\eta = \mu\lambda$.

The Fitness function is then: $f(\eta) = e^{\left(\frac{-E(\eta)}{T}\right)}$

The Energy function to which it refers is:

$$E(\eta) = E(\mu\lambda) = \alpha \sum_{i=1}^{NPGMS} \Phi\left(\frac{Time(\bar{x}, Ji) - Di}{Di}\right) + \beta \sum_{j=1}^{NPROC} \Phi\left(\sum_{i=1}^{MPGMS} Load(\bar{x}, J_i | \mu^{-1}(Pj)) - 1\right) + \gamma \sum_{J=1}^{NPROC} \Psi\left(1 - \sum_{k=1}^{NNODE} \mu_{i,k}\right)$$

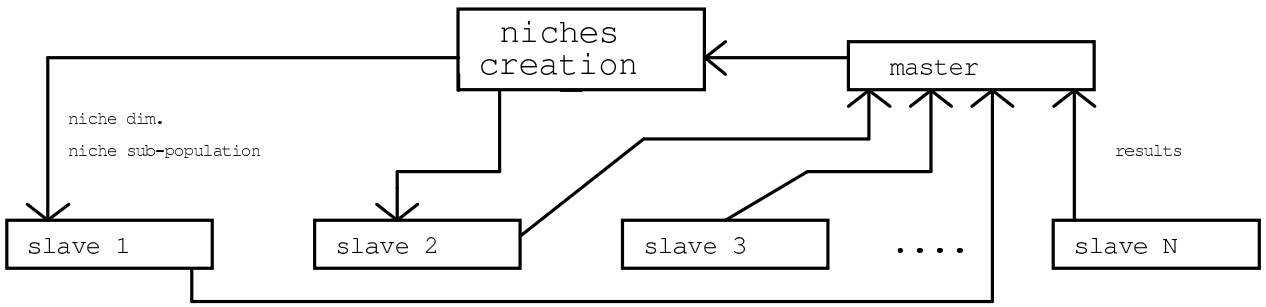
where: $x_i = \frac{1}{\lambda_i}$, $\Psi(x) = \Phi(|x|)$, $\Phi(z) = \begin{cases} 0 & for\ z \le 0 \\ e^z - 1 & for\ z > 0 \end{cases}$

and: $\lambda\ \beta\ \gamma$ are constants used to balance the relative strength of the constraints.

The fitness function follows a simulated annealing model[9],[10],[13]. The factor T ($T \le 1$) is used to speed up the convergence process. Using a standard technique (without this factor) there is a fast convergence up to a neighbourhood of the solution and then, given that all the individuals have about the same fitness, the population evolves very slowly. This phenomenon is much more evident in the case of a narrow solution space. This factor re-scales fitness so that the search process does not slow down too much. The fitness function takes its maximum (having value 1) when the energy is minimum (i.e. when the individual represents a schedulable mapping and time assignment). Optimal solutions are therefore easily detectable. Note that this is a peculiarity of the problem we are considering. In general one cannot expect to determine a fitness function capable of recognising optimal solutions (consider for instance a function minimisation problem: there is no way one can determine by the fitness values whether a solution is a global or a local minimum).

Niched Parallel Genetic Algorithm

Figure 1: the NPGA



The Niching schema provides a criterion for an efficient sub-division of the population among the available processors. As Jeffrey Horn claims in his paper [8], "... Niched GAs avoid the loss of information that results from convergence of the population to a single, hi-fitness individual. Rather, the niched GA seeks to maintain several sub-populations, or species, of individuals at different good solutions. When we use a niched GA we are asking the algorithm to tell us more about the fitness landscape than what the best solution is. [...]. Each peak in the fitness landscape, or region of high fitness, can be considered a "niche". We would like the niched GA to find the best niches, and fill such niches with individuals in proportion to the quality of each niche". This model clearly carries a considerable degree of intrinsic parallelism, since each processor can be assigned one or more niches.

Let us now see how a niched GA divides its population into niches. The niching mechanism we are considering is *fitness sharing*. Other mechanisms well known in GA literature are *crowding*[11] and *Boltzmann tournament selection*[5]. The fitness sharing schema creates niches by modifying the objective fitness according to the presence of nearby individuals. This implies that some kind of distance function is to be provided, so as to estimate *closeness* between individuals (in our work we have considered a genotypical distance, that is the distance between the strings representing the individuals). Niche creation is achieved as follows:

1) All the distances between each couple of individuals in the population are calculated. These distances determine the so-called Share Values of individuals within a fixed radius σ_{share} according to the following formulae:

- $sh(d_{ij}) = 1 - \left(\frac{d_{ij}}{\sigma_{share}}\right)^{\alpha}$ *if* $d_{ij} \le \sigma_{share}$
- $sh(d_{ij}) = 0$ *otherwise*

The shared values are then stored in a matrix square SH (having population size as its dimension). The radius σ_{share} works as an acceptance threshold and determines the niche dimension.

2) Niches are now determined by simply scanning rows or columns of matrix SH (which is clearly symmetrical). For each row or column i, all the individuals j that have non-zero matrix entries, SH[i,j], belong to the i-th niche.

3) For each niche i, the *i-th niche count* m_i is calculated as a sum over the i-th matrix SH row or column of all the non-zero entries:

$$m_i = \sum_{j=0}^{POPSIZE} sh(d_{ij})$$

4) For each individual j belonging to niche i, the shared fitness is:

$$f_{sh}(j) = \frac{f(j)}{mi}$$

It can be shown[11] that in Niched GAs the expected distribution of individuals (number of individuals in each niche) and its variance are independent of the previous generation, and in particular of the initial generation. This is clearly enough an important basis for any kind of convergence proof. We are interested in the fact that such a feature counteracts the GAs typical genetic drift. Niching mechanisms provide a sort of *restorative pressure*: a class (niche) temporarily holding a number of elements other than the expected one is restored to the mean class value. This hinders genetic drift in the sense that classes are preserved from extinction. Independence from the initial generation is important since we expect the GA to converge on every single trial, in order to be sure that convergence is not somehow accidental. The drawback is the complexity overhead introduced by the shared fitness calculation ($o(N^2)$ on the population size). Our parallel implementation exploits the intrinsic parallelism of the paradigm described above, making it simpler and faster. It follows a *master - slave* model, which makes more sense in this case since the master now plays an important role, that is niche creation (steps 1 and 2 of the previous algorithm). After creating the niches, the master assigns each of them to a processor and sends the individuals that compose the niche sub-population to the same processor, along with the sub-population size. Note that the niches are overlapping. From this point on, computation continues in parallel, since each processor is responsible for its own sub-population, evolving it by making use of its own copy of typical genetic operators (selection, mutation and cross-over). No shared fitness is ever calculated, since niches are kept separate by being physically treated on different processors. Moreover, processors do not need to communicate with each other (thus avoiding transmission overheads). The master waits for the first slave to finish off its computation and to send its results. Figures 1 represents the Niched Parallel Genetic Algorithm.

Experimental Results

We have run the algorithm described in this paper, together with a sequential version (SGA) that will serve as a basis for comparison, on a 64 processor Cray T3D system. The parallel program has been implemented using PVM library. The schedulability checking problem instance that we report consists of 8 procedures (belonging to the same Quicksort program) to be mapped onto 2 processors. The main parameter on which we have modelled our trials is *population size*. We consider this parameter significant as it directly reflects the degree of difficulty of the problem to be solved, but also because the whole of our discussion has been focused on issues regarding populations and sub-populations. Each problem seems to have a minimum population size that makes the GA reach its maximum convergence speed. Another factor that has to be taken into account when testing GAs is their heuristic nature. This means that the result of a single trial cannot be considered reliable, because chance plays an important role. Each convergence time we will exhibit is therefore the mean value over at least five trials. Our goal is anyway to give an idea of how the two paradigms of Parallel GAs perform. Figure 2 reports the different performances of the two programs for the schedulability checking of parallel Quick Sort, obtained varying the population size and the number of processors on which the parallel programs are run (8 and 64). As one can see from the graphics, convergence times for the SGA are proportional to the increase of population size. This is due to the fact that such an easy instance of the Schedulability Checking problem can be solved using a small population (8 individuals). Larger populations slow down the process of convergence. The Niched Parallel Genetic Algorithm does not seem to be significantly affected by the population increase. It is in fact capable of overcoming the negative effect of an over-sized population by shrinking the dimension of the niches. This dimension is controlled by the σ_{share} parameter. An important point needs to be made here: in order for the NPGA to achieve these results, parameter σ_{share} had to be finely tuned.

Although the Niched Parallel Genetic Algorithm we have proposed seems to exploit GAs intrinsic parallelism fairly well, the 15% speed-up it accompliches is still far from being a good result for a parallel program.

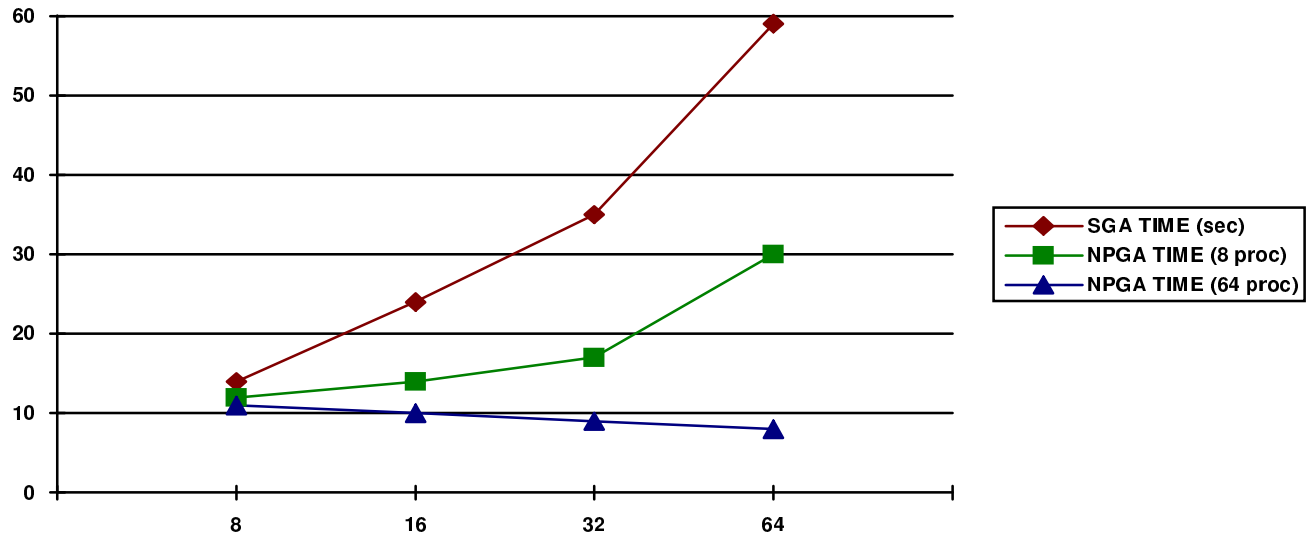


Figure 2: A single instance of the Quicksort program to be mapped onto 2 processors: the parallel algorithm is run on 8 and 64 processors.

Acknowledgements

We would like to thank Luigi-Alberto Giachini, co-member of the research team that investigates HRT schedulability problems. We are also grateful to Giovanni Erbacci and Angelo Neri of the CINECA (Interuniversity Consortium of the Northeastern Italy for Automatic Computing) for their technical help and precious suggestions.

References

[1] R. Davoli, F. Tamburini, L. A. Giachini, F. Fiumana, "A Neural Network Approach to schedulability Checking in Real-Time Systems," Journal of Artificial Neural Networks, special issue on Neural Networks for Optimization, to appear.

[2] R. Davoli and L. A. Giachini, "A Schedulability Algorithm for Data Flow Hard-Real-Time Distributed Programs," In *Proc. IFAC Real Time Programming*, pp. 39--43, Lake Constance, Germany, June 1994.

[3] R. Davoli and L. A. Giachini, O. Babaoglu, A. Amoroso, L. Alvisi, "Parallel Computing in Networks of Workstations with Paralex", IEEE Transactions on Parallel and Distributed Computing, to appear.

[4] R. Davoli and L. A. Giachini, "Schedulability Checking of Data Flow Tasks in Hard-Real-Time Systems", Tech. Rep. UBLCS-94-4, Bologna (Italy), 1994, available through ftp anonymous at ftp.cs.unibo.it.

[5] D.E. Goldberg, "A Note on Boltzmann Tournament Selection for Genetic Algorithms and Population-Oriented Simulated Annealing," Complex Systems 4 (1990) pp. 445-460.

[6] D.E. Goldberg, "Genetic Algorithms in Search, Optimization, and Machine Learning," Addison-Wesley, Reading Mass., 1989.

[7] J.H. Holland, "Adaptation in Natural and Artificial Systems," the University of Michigan press, Ann Arbour, 1975.

[8] J. Horn, "Finite Markov Chain Analysis of Genetic Algorithm with Niching," *Proceedings of the Fifth International Conference on Genetic Algorithm* (ICGA 5), San Mateo, CA, February 1993.

[9] L. Ingber, "Adaptive Simulated Annealing," Technical Report, Lester Ingber Research, McLean, VA, 1993.

[10] L. Ingber, "Very Fast Simulated Re-Annealing," *Mathl. Comput. Modelling*, 12(8), pp. 967 --973, 1989.

[11] Samir W. Mahfoud, "Simple Analytical Models of Genetic Algorithms for Multimodal Function Optimization," IlliGAL Report No. 93001, February 1993.

[12] J. Ribeiro, C. Alippi, P. Treleaven, "Genetic Algorithm Programming environment."

[13] B. Rosen, "Function Optimization based on Advanced Simulated Annealing," In *IEEE Workshop on Physics and Computation - PhysComp 92,* pp. 289--293, 1992.

[14] K. Schwan and H. Zhou, "Dynamic Scheduling of Hard Real-Time Tasks and Real-Time Thread," *IEEE Transactions on Software Engineering*, vol. 18(8), August 1992.

[15] J.A. Stankovic, M. Spuri, M. Di Natale and G.C. Buttazzo, "Implications of Classical Scheduling Results for Real-Time Systems," *IEEE Computer*, pp. 16--25, June 1995.

[16] J. Zhu, T. Lewis, W. Jackson and R. Wilson, "Scheduling In Hard Real-Time Applications," *IEEE Software*, vol. 12(3), pp. 54--63, May 1995.