Parallel Evolutionary Computation: Application of an EA to Controller Design

M. Parrilla, J. Aranda, and S. Dormido-Canto

Dpto. de Informática y Automática,
E.T.S. de Ingeniería Informática, UNED
{mparrilla, jaranda, sebas}@dia.uned.es

Abstract. The evolutionary algorithms can be considered as a powerful and interesting technique for solving large kinds of control problems. However, the great disadvantage of the evolutionary algorithms is the great computational cost. So, the objective of this work is the parallel processing of evolutionary algorithms on a general-purpose architecture (cluster of workstations), programmed with a simple and very well-know technique such as message passing.

1 Introduction

Efficiency of evolutionary algorithms in the optimization problem solution lead to consider them as an alternative method to solve control systems problems. The Evolutionary Algorithms (EA) present a series of advantages with respect to other methods that are most effective in some situations but present applicability limitations. Some of this methods are:

– **Linear Programming**. Only applicable to problems with linear functions.
– **Non-linear Optimization Methods Based on the Gradient**. Applicable to problems with non-linear functions. The functions must be continuous and differentiable, at least at the neighborhood of the optimum. The methods based on the gradient also work with linear problems, but in this case, the linear programming is preferable.
– **Exhaustive Search**. Applicable on those cases where there is a limited number of solutions to problem.

However, the evolutionary algorithms are independent of the function to optimize, and can be applied when the number of possible solutions is unlimited.

The application of evolutionary algorithms to control can be classified in two main groups: first, the *off-line* applications, the most cases are included in this group; in these applications the EA can be employed as a search and optimization engine to select suitable control laws for a plant to satisfy given performance criteria or to search for optimal parameter setting for a particular controller structure. And second, the *on-line* applications, where the EA may be used as a learning mechanism to identify characteristics of unknown or non-stationary systems or for adaptive controller tuning for known or unknown plants. The *on-line* methods present two essential problems: on the one hand, the high computational cost, making difficult that the parameters are available when they are needed, and on the other hand, and even a more important thing, is the fact that the stochastic nature of this kind of algorithms could lead to that the best solution obtained doesn't comply with the minimum requirements. These disadvantages have made to focus almost all works in the off-line methods, leaving the on-line methods for pure research works.

In certain circumstances, the algorithm execution time comes a very important factor. In the case of *on-line* methods is necessary to have the parameters in the appropriate moment, a higher power in computation is required. Other case is when the algorithm become more complex, as in [10], here the controller is unknown and the algorithm assumes responsibility for determining the number of zeros and poles in the controller and its tuning.

When computational complexity is increased, the use of multiple processors to solve the problem, acquire significance. Available options, can be grouped basically in two categories: to use a supercomputer with multiple processors, or to use a *cluster* of PC's.

The advantages of *clusters* of PC's over supercomputers are: a much lower cost, and a higher capacity to be upgraded. In this work, a *cluster* of 14 PC's was used. Regarding the *software*, Matlab with the PVMTB functions library were used.

2 Controllers Design by Evolutionary Algorithms

In the early 1990s, evolutionary algorithms were first investigated as an alternative method of tuning PID controllers.

Oliveira *et al* [8] used a standard genetic algorithm to get initial estimates for the values of PID parameters. They applied their methodology to a variety of linear time-invariant systems.

Wang and Kwok [14] tailored a genetic algorithm to PID controller tuning. They stressed the benefit of flexibility with regard to cost function, and alluded to the concept of Pareto-optimality to simultaneously address multiple objectives.

More recently, Vlachos *et al* [13] applied a genetic algorithm to the tuning of decentralized PI controllers for multivariable processes. Controller performance was defined in terms of time-domain.

Onnen *et al* [9] applied genetic algorithms to the determination of an optimal control sequence in model-based predictive control. Particular attention was paid to non-linear systems with input constraints.

Genetic algorithms have also been successfully applied in the field of H-infinity control. Chen and Cheng [3] proposed a structure specified H-infinity controller. The genetic algorithm was used to search for good solutions within the admissible domain of controller parameters.

They have also been extended to simultaneously address multiple design objectives, achieved via the incorporation of multiobjective genetic algorithm

(MOGA). Multiple design objectives may be defined, in both the time and frequency domain, resulting in a vector objective function. In one such study, Fonseca and Fleming [4] applied a MOGA to the optimization of the low-pressure spool speed governor of a Rolls-Royce Pegasus gas turbine engine.

Research has also been directed toward the so-called intelligent control systems. The two most popular techniques are *fuzzy control* and *neural control*.

Ichikawa and Sawa [5] used a neural network as a direct replacement for a conventional controller. The weights were obtained using a genetic algorithm. Each individual in the population represented a weight distribution for the network.

Tzes *et al* [11] applied a genetic algorithm to the *off-line* tuning of Gaussian membership functions, developing a fuzzy model that described the friction in a dc-motor system.

Evolutionary methods have also been applied to the generation of control rules, in situations where a reasonable set of rules is not immediately apparent. Matsuura *et al* [7] used a genetic algorithm to obtain optimal control of sensory evaluation of the sake mashing process. The genetic algorithm learned rules for a fuzzy inference mechanism, which subsequently generated the reference trajectory for a PI controller based on the sensory evaluation. Varsek *et al* [12] also used genetic algorithms to develop rule bases, applied to the classic inverted pendulum control problem.

3 Problem Description

The problem formulation corresponds to the RCAM design problem proposed for the GARTEUR Action Group FM(AG08), [6]. The non-linear model proposed was used to generate a linear model around the following conditions: airspeed $V_A = 80$ m/s, altitude h $= 1000$ m, mass $= 120000$ kg, center of gravity cgx $=$ 0.23, cgz $= 0.1$ and transport delay $\delta = 0$. From the linearized model obtained, only the longitudinal mode was used, because is trivial to extend the algorithm to the lateral mode, once designed.

The linearized model, in the state-space representation, is:

$$\begin{pmatrix} \dot{q} \\ \dot{\theta} \\ \dot{u}_B \\ \dot{w}_B \\ \dot{X}_T \\ \dot{\chi}_{TH} \end{pmatrix} = \begin{pmatrix} -0.9825 & 0 & -0.0007 & -0.0161 & -2.4379 & 0.5825 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ -2.1927 & -9.7758 & -0.0325 & 0.0743 & 0.1836 & 19.6200 \\ 77.3571 & -0.7674 & -0.2265 & -0.6683 & -6.4785 & 0 \\ 0 & 0 & 0 & 0 & -6.6667 & 0 \\ 0 & 0 & 0 & 0 & 0 & -0.6667 \end{pmatrix} \begin{pmatrix} q \\ \theta \\ u_B \\ w_B \\ X_T \\ \chi_{TH} \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 6.6667 & 0 \\ 0 & 0.6667 \end{pmatrix} \begin{pmatrix} d_T \\ d_{TH} \end{pmatrix}$$

$$\begin{pmatrix} q \\ n_x \\ n_z \\ w_V \\ V_A \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0.0075 & 0 & -0.0033 & 0.0076 & 0.0187 & 2 \\ -0.2661 & 0 & -0.0231 & -0.0681 & -0.6604 & 0 \\ 0 & -79.8667 & -0.0283 & 0.9996 & 0 & 0 \\ 0 & 0 & 0.9996 & 0.0290 & 0 & 0 \end{pmatrix} \begin{pmatrix} q \\ \theta \\ u_B \\ w_B \\ X_T \\ \chi_{TH} \end{pmatrix} \tag{1}$$

where the states are: pitch rate (q), pitch angle (θ), x component of the inertial velocity in body-fixed reference frame (u_B), z component of the inertial velocity in body-fixed reference frame (w_B), state corresponding to the tailplane (X_T) and state corresponding to the engines throttles (χ_{TH}).

The outputs are: pitch rate (q), horizontal load factor (n_x), vertical load factor (n_z), z component of vertical velocity in the vehicle-carried reference frame (w_V) and air speed (V_A).

The objectives considered are in the design specifications, in the document [6]. For longitudinal mode, the design specifications are summarized as follows:

- Closed-loop stability: it's the most basic objective to be satisfied.
- The control system should be able to track step reference signals: in V_A with a rise time $t_r < 12s$, a setting time $ts < 45s$ and overshoot $M_p < 5\%$, and in flight path angle (γ) with $t_r < 5s$, $t_s < 20s$ and $M_p < 5\%$. But γ isn't available in the model. To cope with such a problem, the relation $sin(\gamma) = \frac{-w_V}{V}$ can be used, where V is the total inertial velocity.
- Ride quality criteria: vertical accelerations would be minimized.
- Saturations limits in control signals, would be observed.
- Robustness criteria: the gain margin is required to be at least 10 dB and the phase margin is required to be at least 50.

In a previous work [1], the authors of this document solved the problem by means of an sequential evolutionary algorithm. They took a fixed controller structure for this, as showed in figure 1, where a static gain matrix (K_p) was directly applied on the 5 model outputs, and another gain matrix (K_i) was applied on the integral of the errors in V_A y w_V, to eliminate steady state errors. The algorithm got the gains, K_p y K_i, meeting the design specifications.

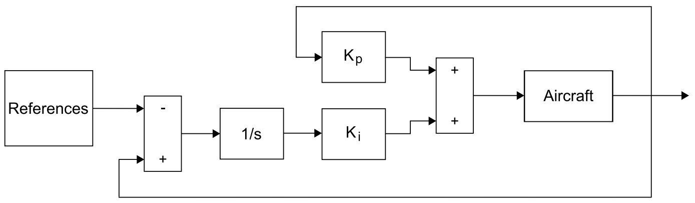

Fig. 1. Structure of longitudinal controller

The sequential evolutionary algorithm used, is shown in figure 2. A initial population of chromosomes is randomly generated, whose members would be possible solutions to the problem, properly coded. Chromosomes would be evaluated and sorted according to fitness. The chromosome with the best fitness would be established as the problem solution. After that a loop would start, where new generations of chromosomes would be obtained from previous generation, by applying evolutionary operators over the parents selected in a previous step. The new generation would be again evaluated and sorted according to fitness. If the best chromosome in the current population was more suitable than the previously established as solution to problem, it would replace it. Finally if end conditions are satisfied, the program would finish, otherwise a new loop iteration would start.

In a more ambitious project, it's possible to let the algorithm to find the controller structure, or in a multivariable system, ask the algorithm for the matrix of transfer functions representing the controller, taking the algorithm charge

```
$k \leftarrow 0$
get random initial population $P_k(x)$
for each $x_i$ in $P_k(x)$
    decode $x_i$
    evaluate $x_i \leftarrow$ get fitness
endfor
sort $P_k(x)$ according to fitness
solution $\leftarrow$ best $x_i$ in $P_k(x)$
while not end conditions
    $k \leftarrow k+1$
    select parents by tournament selection
    get new population $P_{k+1}(x)$
    for each $x_i$ in $P_k(x)$
        decode $x_i$
        evaluate $x_i \leftarrow$ get fitness
    endfor
    sort $P_k(x)$ according to fitness
    if the best $x_i$ in $P_k(x)$ is better than solution
        solution $\leftarrow$ best $x_i$ in $P_k(x)$
    endif
endwhile
```

Fig. 2. Sequential Evolutionary Algorithm

in that case of determining the number of zeros and poles for each transfer functions. To solve this kind of problems, the algorithm would have to tune a very high number of controllers, which meant that the time of execution increase dramatically. In that cases, the necessity to speedup the tune process arise.

If the sequential algorithm in figure 2 is parallelized, and a good speedup is obtained, the parallelization result could be used as a component of a more complex program: the parallelized algorithm would be called every time a controller has to be tuned. Now the parallelization of the sequential algorithm is described and also the speedup obtained are shown.

4 Parallelizing the Algorithm

Matlab is a standard in control, thanks to its specialized *toolboxes*. However, it lacked ability to carry out parallel programing. To cover this gap, Baldomero [2], designed PVMTB (Parallel Virtual Machine ToolBox), a *toolbox* including almost all functionalities in PVM, the known parallelization library by message passing.

Thanks to PVMTB, all the control specialized Matlab functions can be used to design a parallel evolutionary algorithm. The parallelizing process will be described below.

Matlab with PVMTB, and a Master/Slave strategy were used.

In order to parallelize effectively the sequential algorithm previously described, a study to determine the most computing intensive parts was carried out, resulting that generating new chromosomes from parents and determining fitness were the most time consuming tasks. Therefore, this were the stages where parallelization would be focused.

An inherent feature of evolutionary algorithms was also taken into account: for each generation, all chromosomes will have to be created and evaluated before continuing with a new generation. If the workload isn't uniformly distributed between the different processes, those that firstly finish their work will have

```
start up PVM: pvm_start_pvmd();
start up slave tasks: pvm_spawn();
$k \leftarrow 0$
get random initial population $P_k(x)$
for each $x_i$ in $P_k(x)$
   decode $x_i$
   evaluate $x_i$ $\leftarrow$ get fitness
endfor
sort $P_k(x)$ according to fitness
solution $\leftarrow$ best $x_i$ in $P_k(x)$
while not end conditions
   $k \leftarrow k+1$
   select parents by tournament selection
   for each slave
      send a pair of parents: pvm_send();
   endfor
   while there is a pair of parents not used
      receive 2 evaluated chromosomes from one slave: pvm_recv();
      send a pair of parents to this slave: pvm_send();
   endwhile
   while num chromosomes asked for < size of $P_k(x)$
      receive 2 evaluated chromosomes from one slave: pvm_recv();
      ask for 2 inmigrants to this slave: pvm_send();
   endwhile
   while num chromosomes received < size of $P_k(x)$
      receive 2 evaluated chromosomes from one slave: pvm_recv();
   endwhile
   sort $P_k(x)$ according to fitness
   if the best $x_i$ in $P_k(x)$ is better than solution
      solution $\leftarrow$ best $x_i$ in $P_k(x)$
   endif
endwhile
for each slave
   send signal to quit
endfor
halt PVM: pvm_halt;
```

Fig. 3. Master Process Algorithm

to wait a lot, because they can't start with a new generation until the others processes finish with the current one. That obviously imply a decrease of the speedup obtained with parallelization.

An added difficulty is the fact that times to get the fitness of different chromosomes, can be very different. A chromosome implying a unstable system, would be quickly evaluated, but those that give rise to a stable system would have to be studied more slowly, to determine the system features, increasing the evaluation time. To minimize this problem, the master divided the work in small tasks, and quickly assign a task to slaves waiting for a work.

The master also had to receive the results from slaves, and to organize them as they arrived. It wasn't necessary to parallelize the evaluation of the initial population, because first chromosomes generally give rise to unstable systems, and are quickly evaluated. The algorithm corresponding to the master process is shown in figure 3.

Slave processes for its part, were concerned with generating a pair of offsprings from the pair of parents passed by the master, evaluating them, and sending the new chromosomes obtained and its fitness to master. They also generated and evaluated pairs of immigrants, randomly obtained, when master requested them. Slaves would generate and evaluate chromosomes until the master send them the end signal. The algorithm used by slaves is shown in figure 4.

```
while not signal to quit
   get master message: pvm_recv();
   if parents provided
      generate 2 offsprings
   else
      generate 2 inmigrants
   endif
   for each generated chromosome
      decode chromosome
      evaluate chromosome
   endfor
   send chromosomes and fitness to master: pvm_send();
endwhile
quit
```

Fig. 4. Slave Process Algorithm

5 Hardware and Software Description

The *cluster* used in this work had 14 PC's with AMD K7 500 MHz processors, 384 MB of RAM memory and a hard drive of 7GB each of them. The nodes (1 Master + 13 Slaves) were connected by Fast-Ethernet switch. The operating system was Linux (Red-Hat 6.1).

The algorithm was implemented using a *toolbox* of parallel processing, developed in Matlab by Baldomero, J.F., [2]: PVMTB (Parallel Virtual Machine

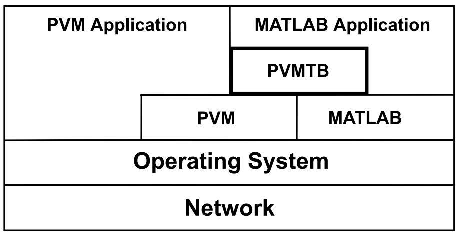

Fig. 5. High level overview of PVM

ToolBox), based on the standard library PVMTB. With PVMTB, Matlab users can quickly build parallel programs, using a message passing system like PVM.

Figure 5 shows a high level overview diagram of PVMTB. The *toolbox* makes PVM and Matlab-API (Application Program Interface) calls to enable messages between Matlab processes.

6 Performance Results

After 4 executions of the parallel algorithm, to tune the controller in figure 1, an average of the times of execution and the speedups obtained, was calculated. Each execution was repeated for each possible number of processors. Results are shown in figure 6, and its numeric values are grouped in table 1.

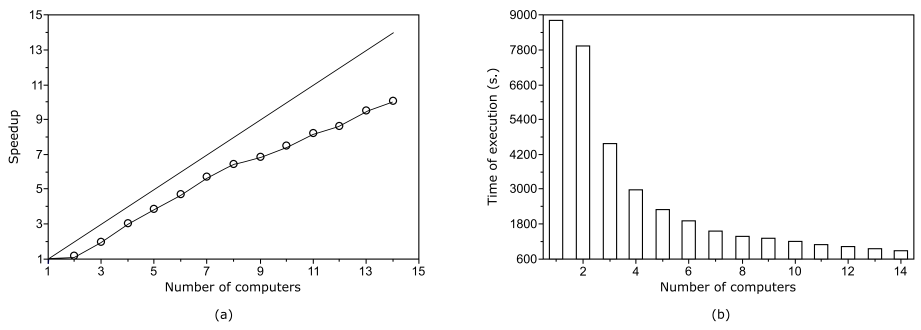

Fig. 6. (a) Speedup vs. number of computers. (b) Time of execution vs. number of computers

7 Conclusions

Features of evolutionary algorithms make them appropriated to deal with problems, difficult to be solved by other methods. A drawback is its high computational cost, making them impossible to be applied to solve complex problems, in some cases. But evolutionary algorithms are easily parallelized by nature, and *clusters* of PC's provide a low-cost alternative to supercomputers.

Table 1. Speedup and Time of execution vs. number of computers

Number of computers	Time of execution (seg.)	Speedup	Standard deviation (speedup)
1	8905.9259	1	0
2	7953.4766	1.1197555	0.0079589
3	4597.6172	1.9388577	0.0611588
4	2991.8349	2.9776865	0.0541746
5	2310.3739	3.8557793	0.0610961
6	1904.5179	4.6804295	0.1380201
7	1566.8359	5.6894018	0.1702718
8	1392.8489	6.3950083	0.0752763
9	1312.2312	6.7910607	0.1919798
10	1194.6286	7.4551005	0.0350726
11	1087.4028	8.1928836	0.1823867
12	1034.5336	8.6091675	0.0984071
13	941.04137	9.4676986	0.2028066
14	890.80885	10.023033	0.5184515

Acknowledgment

Part of this work was supported by MCyT of Spain under contract DPI2003-09745-C04-01.

References

1. Aranda, J.; De la Cruz, J.M.; Parrilla, M. and Ruipérez, P.: *Evolutionary Algorithms for the Design of a Multivariable Control for an Aircraft Flight Control*, AIAA Guidance, Navigation, and Control Conference and Exhibit, Denver, CO. (August 2000)
2. Baldomero, J.F.: *PVMTB: Parallel Virtual Machine ToolBox*, II Congreso de Usuarios Matlab'99, Dpto. Informática y Automática. UNED. Madrid. (1999) pp. 523-532
3. Chen, B. S., and Cheng, Y. M.: *A Structure-Specified H-Infinity Optimal Control Design for Practical Applications: A Genetic Approach*, IEEE Transactions on Control Systems Technology, Vol. 6. (November 1998) pp707-718
4. Fonseca, Carlos M., and Fleming, Peter J.: *Multiobjective Optimization and Multiple Constraint Handling with Evolutionary Algorithms-Part I: A Unified Formulation and Part II: Application Example*, IEEE Transactions on Systems. Man and Cybernetics. Part A: Systems and Humans. Vol. 28. No. 1. (January 1998) pp26-37 and pp38-47
5. Ichikawa, Y., and Sawa, T.: *Neural Network Application for Direct Feedback Controllers*, IEEE Transactions on Neural Networks, Vol. 3, No. 2. (March 1992) pp224-231
6. Lambrechts, P.F. et al.: *Robust flight control design challenge problem formulation and manual: the research civil aircraft model (RCAM)*. Technical publication TP-088-3, Group for Aeronautical Research and technology in EURope GARTEUR-FM(AG-08). (1997)

7. Matsuura, K.; Shiba, H.; Hirotsune, M., and Nunokawa, Y.: *Optimal control of sensory evaluation of the sake mashing process*, Journal of Process Control, Vol. 6, No. 5. (1996) pp323-326
8. Oliveira, P.; Sequeira, J., and Sentieiro, J.: *Selection of Controller Parameters using Genetic Algorithms*, Engineering Systems with Intelligence. Concepts, Tools, and Applications, Kluwer Academic Publishers, Dordrecht, Netherlands. (1991) pp431-438
9. Onnen, C.; Babuska, R.; Kaymak, U.; Sousa, J. M.; Verbruggen, H. B., and Isermann, R.: *Genetic Algorithms for optimization in predictive control*, Control Engineering Practice, Vol. 5, Iss. 10. (1997) pp1363-1372
10. Parrilla, M.; Aranda, J. and Díaz J.M. *Selection and Tuning of Controllers, by Evolutionary Algorithms: Application to Fast Ferries Control.* CAMS2004, IFAC. (2004)
11. Tzes, A.; Peng, P. Y., and Guthy, J.: *Genetic-Based Fuzzy Clustering for DC-Motor Friction Identification and Compensation*, IEEE Transactions on Control Systems Technology, Vol. 6, No. 4. (July 1998) pp462-472
12. Varsek, A.; Urbancic, T., and Fillipic, B.: *Genetic Algorithms in Controller Design and Tuning*, IEEE Transactions on Systems, Man, and Cybernetics, Vol. 23, No. 5. (September/October 1993) pp1330-1339
13. Vlachos, C.; Williams, D., and Gomm, J. B.: *Genetic approach to decentralized PI controller tuning for multivariable processes*, IEEE Proceedings - Control Theory and Applications, Vol. 146, No. 1, (January 1999) pp58-64
14. Wang, P., and Kwok, D. P.: *Autotuning of Classical PID Controllers Using an Advanced Genetic Algorithm*, International Conference on Industrial Electronics, Control, Instrumentation and Automation (IECON 92), Vol. 3. (1992) pp1224-1229