Evolutionary algorithm for the parallel System Identification optimization

Juan A. Gómez Pulido, Germán Galeano Gil, Francisco Fernández de Vega, Juan M. Sánchez Pérez, and Miguel A. Vega Rodríguez

Dep. of Computer Sciences. University of Extremadura. Cáceres, Spain
{jangomez,ggaleano,fcofdez,sanperez,mavega}@unex.es

Abstract. In this work we present a parallel algorithm inspired on the evolutionary methodologies, for high precision computation of system identification algorithms. The parallel algorithm will be used on the environment of control process engineering because it is very useful when the mathematical modelation of dynamic complex systems is neccessary. Because of the good results found with the algorithm simulation, we are working now to synthetize it as a coprocessor using reconfigurable hardware.

1. Introduction

1.1. System Identification

In many engineering fields it is necessary to dispose of mathematical models for studying the behaviour of systems whose mathematical description "a priori" is not possible. In these systems only the input and output (I/O) values are known and its physical structure is not very known. This drives to the necessity of planning System Identification (SI) techniques [1]. The SI is born of the experimental method and it tries to find a system parametric mathematical model. Only the system behaviour for a set of given stimuli is known. We consider "single input single output" (SISO) sampled systems with period T and parametric polinomial ARMAX modelation [2].

$$A(q)\ y(k) = B(q)\ u(k-nk) + C(q)\ e(k) \qquad (\mathbf{1})$$

being

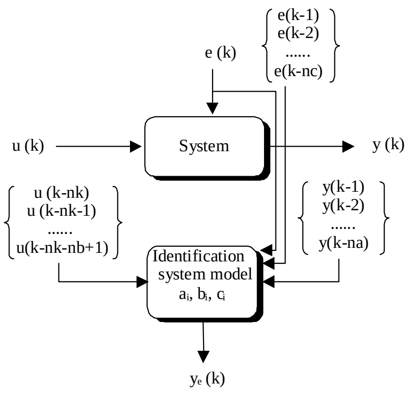
- $A(q) = 1 + a_1q^{-1} + ... + a_{na}q^{-na}$, the a_i represent the output parameters (size na)
- $B(q) = b_1 + b_2q^{-1} + ... + b_{nb}q^{-nb+1}$, the b_i represent the input parameters (size nb)
- $C(q) = 1 + c_1q^{-1} + ... + c_{nc}q^{-nc}$, the c_i represent the noise parameters (size nc)
- nk, output-input delay; q, delay unit (q^{-p} delays x(k) to x(k-p)); na ≥ nb and na ≥ nc

Basically the identification consists in determining the ARMAX model parameters (a_i, b_j, c_k) from measured inputs and outputs observation. With these parameters it is possible to compute the estimated output $y_e(k)$ and compare it with the real output y(k), computing the generated error (Fig.1).

$$y_e(k) = [- a_1 \cdot y(k-1) - ... - a_{na} \cdot y(k-na)] + [b_1 \cdot u(k-nk) + b_2 \cdot u(k-nk-1) + ... + b_{nb} \cdot u(k-nk-nb+1)] + [e(k) + c_1 \cdot e(k-1) + ... + c_{nc} \cdot e(k-nc)] \quad \textbf{(2)}$$

where

- ye(k) is the estimated output by the model
- y(k), u(k), e(k) are real output, input and noise at present time
- y(k-1),..., u(k-1),..., e(k-1),... are real outputs, inputs and noises at previous time



$$\theta = \begin{pmatrix} a_1 \\ a_2 \\ .. \\ a_{na} \\ b_1 \\ b_2 \\ .. \\ b_{nb} \\ 1 \\ c_1 \\ c_2 \\ .. \\ c_{nc} \end{pmatrix} \qquad \varphi(k) = \begin{pmatrix} -y(k-1) \\ -y(k-2) \\ \cdots \\ -y(k-na) \\ u(k-nk) \\ u(k-nk-1) \\ \cdots \\ u(k-nk-nb+1) \\ e(k) \\ e(k-1) \\ e(k-2) \\ \cdots \\ e(k-nc) \end{pmatrix}$$

Fig. 1. System identification algorithm based on parametric estimation: inputs and outputs are measured and computed to generate the estimated output from an ARMAX mathematical model. Estimated output, in its matricial notation, is $y_e(k) = \varphi^T(k) \cdot \theta$, where θ is the parameter vector and φ(k) is the data vector.

1.2. Identification modes

The recursive parametric estimation estimates (and updates) θ in each time k, modelling so the system. Obviously, to more identification (more sampled data processed) better precision for the model because it has more information about the system behaviour history. The SI will be performed by the Recursive Least Squares (RLS) algorithm (also known *forgetting factor* –FF or λ-) because it is one of the most precise and most used. From the initial conditions (k = p = initial time, θ(p)=0, P(p)=10000·I, where I is the identity matrix), we start building φ(k), then:

$$y_e(k)=\varphi^T(k)\cdot\theta(k-1) \Rightarrow err(k)=(y(k)-y_e(k)) \Rightarrow K=\frac{P(k-1)\cdot\varphi(k)}{\lambda+\varphi^T(k)\cdot P(k-1)\cdot\varphi(k)} \quad \textbf{(3)}$$

$$\Rightarrow P(k) = \frac{P(k-1) - K\cdot\varphi^T(k)\cdot P(k-1)}{\lambda} \Rightarrow \theta(k) = \theta(k-1) + K\cdot err(k)$$

This algorithm, that we call CRLS (C means *Classical*), is specified with λ, P(0), θ(0) and the observed values u(k), y(k), e(k). There is not any fixed value for λ parameter (forgetting factor), even though the author of this algorithm recommends a value between 0.97 and 0.995 [3]. The cost function **F** is defined as the value to minimize:

$$F(\lambda) = \sum_{k=k_0}^{k=k_0+SN-1} \left| y_e(k) - y(k) \right| \qquad \textbf{(4)}$$

where SN is the sample number.

1.3. Utility of the recursive identification for the system knowledge

Real systems are complex, with a great variability in their responses. It is necessary to know the behaviour that can not be experimentally predicted in order to no endanger the system security. As an example, in the prevision of critical or emergency situations that may endanger the integrity of a process: For controlling it is necessary to predict and for predicting it is necessary to know. This acknowledge, adquired by means of the SI, consists in elaborating a mathematical model for covering the system behaviour under all working conditions, even under the more extremes.
The SI allows to find, in sample time, a mathematical model (θ(k)) from which is possible to elaborate future predictions. As identification advances in the time, the predictions improve using more precise models. As an example, an architecture with a specialized coprocessor that computes in sample time a system model and a central processor that performs (with the computed model) a simulation of the system future behaviour, forwarding real situations (Fig. 2).
For validating the SI evolutionary parallel architecture that we present, we use a benchmark set, corresponding to SISO systems, found from real data sets [4] (Fig. 2).

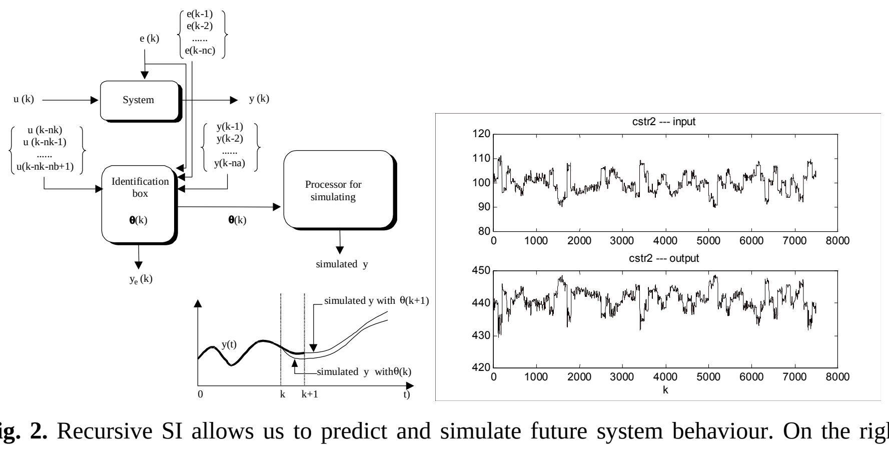


Fig. 2. Recursive SI allows us to predict and simulate future system behaviour. On the right side, a capture of a real system benchmark used.

2. Evolutionary computation for System Identification

In the SI, the model is generated “a posteriori” using the measured and stored I/O data. However we are interested in the system behaviour prediction in running time, that is, while the system is working and its I/O are being observed. This may be the case of some precision process controlled by its input signal for holding the output signal within some limits, because of its quick and unpredictable behaviour. So, it would be interesting to generate models in running time in such a way that a processor may simulate the system next behaviour.
In other way, the system precision degree is due to the following three situations:

- The λ parameter. There is not any determined value for this parameter. In [2] and [3] is recommended a value within 0.995 and 0.97. Other authors use 0.98. This value may be critical for model precision.
- The model dimensions. There is not any determined rule that recommends the model dimensions. It could be thought to more dimension better model, but it is possible to find a reduced model with more performance than a bigger one.
- The system. Given values for λ and the model dimensions may model satisfactorily a concrete system and no others. By that, it is foreseeable that system have its own optimum parameters.

Also, it may appear the precision problem when a system model is generated in sample time: If the system response changes quickly, then the sample frequency must be high for avoiding the key data loss in the system behaviour description. If the system is complex and its simulation from the model to be found must be very trustworthy, then the required precision must be very high and this implies a great computational cost. Sometimes the hardware resources does not allow the computational cost in the model generation and processing to be lower than the sample period.
We find the trade-off between a high sample frequency and a high precision in the algorithm computation. The required precision to the SI by the λ parameter optimization is a subject perfectly adapted to the use of techniques based on evolutionary algorithms. So, the final target is to achieve an architecture for SI in real time for processes in which will be necessary a high precision and a low sample time.

2.1. Choosing of optimization parameters

Fig. 3 show the estimated output for 4 values of λ for the *ball* benchmark and ARMAX model with na=3, nb=2, nc=1, nk=1 running CRLS. The estimated output is practically perfect according to the select value. Making an intensive computation of 155 values of λ in the range 0.6 – 1.2 we find as optimum values $\lambda = 0.98961$ and F = 1.40762. We can see that for finding an optimal value are necessary intensive computations, however in some situations (real time systems) this is not possible. Moreover, 0.98961 is an optimum value of λ only for modelling the *ball* benchmark with dimensions (3,2,1,1). Each system has its optimum values, so a predetermined general λ value for SI may drive to a precision loss in many SI. By that, it is interesting to research about algorithms for finding the best λ values for each system.

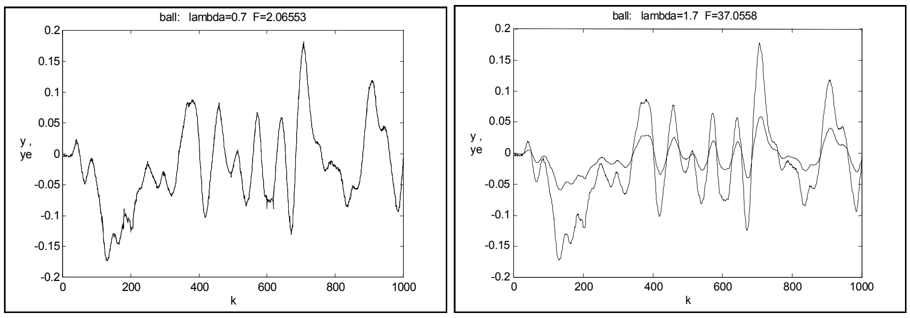


Fig. 3. System Identification CRLS of benchmark *ball* with different values for λ parameter.

In relation with the model dimensions, we can say, according to many computational experiments, that for an acceptable dimension (of reasonable computational cost), λ is more important as optimization parameter.

2.2. Proposal of an parallel evolutionary algorithm

For finding the optimum value of λ, we use an algorithm inspired on the evolution concept [5][6]. Our algorithm, named PERLS (Parallel Evolutive Recursive Least Squares), may be considered as an evolutionary algorithm because its optimization parameter λ is going evolving to new situations during the successive phases of algorithm execution. In PERLS, λ evolves at the same time that improves the cost function performance (Fig. 4).

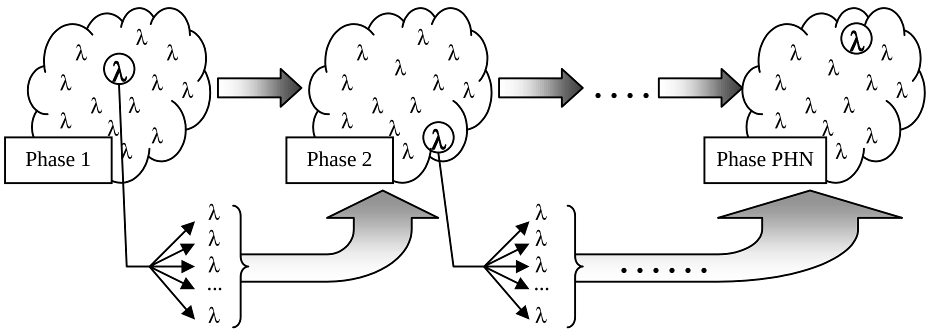


Fig. 4. PERLS evolution. In each phase, a set of λ values performs RLS identification during certain number of samples (PHS). Then, the λ value whose corresponding F is the minimum of all computed F is the optimal, and from it a new set of λ values is generated. They are used in the next PERLS phase to perform new identification during the following PHS samples.

PERLS considers as state a λ value. Starting of an initial R value (being R the interval of λ value generation) and an initial λ, a set of λ values is generated. This set covers uniformly all the R interval, centered at λc (being λc the central value of R). The number of λ values generated is equal to the parallel process units (PU) number

(PUN). Each phase of the PERLS loop corresponds to a given number of sample times. In these times the SI uses the considered λ. We use the following nomenclature:

R	generation interval
λc	λ central in R
PHS	phase samples
PUN	number of process units
TSN	total number of samples
RED	reduction factor of R

Table 1. PERLS nomenclature.

In each phase, R is reduced by the RED factor, in such a way that the generated set of λ will be more and more near of the previous optimum found. In each PU, during each phase, the cost function F is computed. This cost function F is defined as the accumulated error of the samples that constitutes each phase. From eq. (4), we have:

$$F(\lambda_{PUx}) = \sum_{k=k_0}^{k=k_0+PHS-1} \left| y_e(k) - y(k) \right| \quad \textbf{(5)}$$

At the end of each phase, the best λ is choosed. This is the corresponding value to the lower F. From this λ, new values are generated in a more reduced new R interval (Fig. 5 and 6). The goal is that the identifications performed by the PUs will converge to optimum λ parameters when a given stop criterium will be achieved. So the identification will be of high precision. It is clear that the larger PUN, the better precision in the identification (better λ).

```
λc;     /* initial λ */
R;      /* initial generating λ interval */
do  {
    Generating PUN λ’s from λc, covering all R interval
    do  {  /* phase = Identification or PHS samples */
        Computing acumulated error (F) in each PU
        during all samples of the phase
        }
        while(processing PHS samples);
    F(λ) = Accumulated error for each PU
    Determining λopt of all PUs for which F(λ) is minimum
    λc = λopt;
    R = R/RED;    /* R is reduced in a RED factor */
    }
    while(!stop criterium)
```

Fig. 5. PERLS pseudo-code.

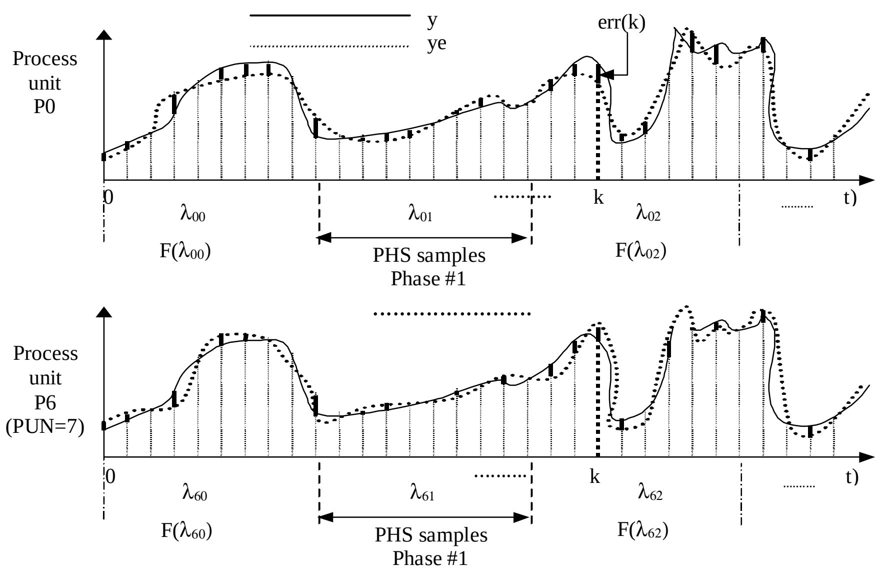


Fig. 6. The identification uses different λ values in each phase within each PU. All the λ values in the same phase running in the PUN process units are generated in the R interval from the previous phase optimum λ found, corresponding with the smallest computed cost function.

2.3. Criteria to be considered

- Optimum λ criterium: Is the λ value that produces a minimum F. The precision of this value depends of the assumed computational effort. The previous obtaining of the optimal λ value for a system will allow us to compute the performance and efectiveness of the evolutionary methodology.
- Stop criterium: It indicates when a PU stops the work, and it will be estimated by the assumeable computational cost.
- Model generation (in the phase) criterium: In each phase of PERLS, a number of identifications equal to PUN are performed. As CRLS uses initial values for θ and P, and θ contains the ARMAX model parameters, there is the possibility to use the θ found at the end of a phase as the initial θ for identification in the next one. The two different configurations of this criterium are the following (Fig. 7):
  - GMF1 criterium: In any phase and for all PUs, the identification will be made by means of CRLS with θ initialized to 0.
  - GMF2 criterium: For any PU, the identification in each phase will be made with the θ found in the previous phase. So the identification in each phase “remembers” the happened in previous phases.
- Optimum F definition (in the phase) criterium: When a phase ends in PERLS, the optimal λ value must be choosed, that is the λ value that computes the minimum cost function F. We consider two criteria for deciding the minimum F:

˜ FOP1: At the end of each phase, the optimum F is the lowest F in all phases and in all PUs from the starting PERLS execution until the present time.

˜ FOP2: At the end of each phase, the optimum F is the lowest F computed by all PUs in the present phase.

It seems more logical to use the FOP2 criterium. In any phase we need know the best λ for the present I/O data range. Then, if we decide the best λ corresponds to a past phase (with a lower F), we are not considering the same conditions, because in the past phase it was used another I/O data range for computing λ. However, this λ would be able to find worse results for the present I/O data.

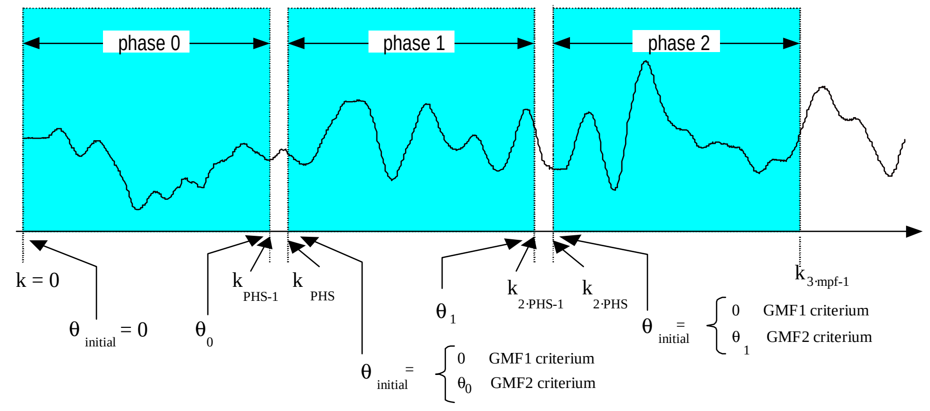

phase 0
phase 1
phase 2
k = 0
θ initial = 0
θ 0
k PHS-1
k PHS
θ initial = 0 GMF1 criterium, θ0 GMF2 criterium
θ 1
k 2·PHS-1
k 2·PHS
θ initial = 0 GMF1 criterium, θ 1 GMF2 criterium
k 3·mpf-1

Fig. 7. Model generation (in the phase) criteria.

3. Parameter tuning

PERLS offers a great variability for its parameters. We show a case for analyzing this variability and may guide us about the best values for the parameters: Benchmark *ball*, with 4000 samples, ARMAX sizes {na=3, nb=2, nc=1, nk=1}, initial interval {λc=1, R=0.8}, 20 values of λ to compute, and optimal λ value = 1.0288 and the corresponding F = 1.435 (these data were obtained computing 20,000 consecutive λ in the range from 0.9 to 1.1).

For the experiments will be comparables (valid for extracting conclusions), the total number of processed λ must be the same in all experiments. We fix this number to 20. As the number of evaluated λ by PERLS is PUN x PHN, only 6 combinations are possible for PUN and PHS. These combinations are analyzed for 7 different values of RED, moreover to contemplate the two criteria FOP and the two criteria GMF.

 GMF1. In Fig. 8a the F values for all experiments are shown. F corresponds to the CRLS computation for the optimal λ value found at the end of PERLS evaluation and for all benchmark samples. Analyzing the two graphics, it may be observed that FOP2 is the best criterium and the best parameter combination found has been PUN=5 and PHS=1000 (4 phases). The best RED have been 2 or 2.5.

- GMF2. In Fig. 8b the same experiment results are shown, but considering GMF2. The conclusions are similar to those found with GMF1. Then, there is not any difference for using GMF1 or GMF2, because the only appreciable differences are produced when FOP criterium is changed.

For verifying the validity of these conclusions, we carry out the same experiments with other benchmarks. From the results found we can conclude there are not common politics for tuning the parameters in such a way that always the best results will be found. The experience indicates the each system nature offers better behaviours for different values of PERLS parameters.

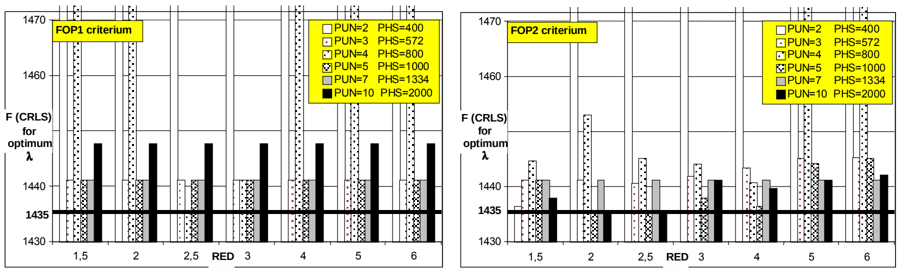


Fig. 8 (a,b). Experiments with benchmark *ball*.

4. Performance

We establish fixed values for PERLS parameters, in order to define an unique algorithm applicable to any system. This has the advantage of an easier PERLS synthesis on a digital architecture (our project in progress). This architecture will quickly be applicable for SI without the previous task of tuning parameters. There is the possibility of same parameter values will give good results in many systems but worse in other ones. Besides this inconvenient, we believe it is useful to build a PERLS general structure with fixed values of its parameters. This structure (Fig. 9) will be subject to a proof versus an CRLS classic SI with an unique λ value. In Fig. 10 the dual results of PERLS performance, using tuned parameters, are shown.

Parameter	**Value**
PUN	5
RED	2
PHN	4
PHS	TSN / 4
GMF	GMF2
FOP	FOP2
ARMAX size	na=3 nb=2 nc=1 nk=1

Fig. 9. Tuned parameters for the portable PERLS algorithm.

First of all, we apply CRLS to the benchmarks using λ values of 0.95 and 0.98, and we compute the corresponding F values. Then, we apply PERLS for computing the optimal λ value. With this value we compute the F corresponding to an CRLS identification with this optimal λ value.

				standard PERLS		CRLS	diference	PERLS	CRLS	diference	PERLS
bench	**TSN**	**PHS**	**PHN**	**λ**	**F**	**F (λ=0.95)**	**of F (%)**	**is better**	**F (λ=0.98)**	**of F (%)**	**is better**
ball	1000	250	4	0.975	1.42563	1.47218	3.16	YES	1.41907	0.46	
boxjen	296	74	4	0.96875	134.728	134.871	0.11	YES	134.644	0.06	
cstr1	7500	1875	4	0.925	0.49858	0.491946	1.35		0.515113	3.21	YES
cstr2	7500	1875	4	0.975	521.277	521.867	0.11	YES	521.13	0.03	
dryer	1000	250	4	1.025	74.8104	81.7031	8.44	YES	77.3192	3.24	YES
exch	4000	1000	4	1	1441.11	1533.51	6.03	YES	1474.31	2.25	YES
fluttr	1024	256	4	0.9	2.88424	4.00232	27.94	YES	6.09777	52.70	YES
heat1	801	201	3.985	0.9	320.306	317.017	1.04		319.71	0.19	
robarm	1024	256	4	0.95	11.181	11.181	0.00	YES	11.1709	0.09	
gen0	2000	500	4	0.95	0.305919	0.305919	0.00	YES	0.301233	1.56	
gen1	2000	500	4	1.05	44.1167	60.174	26.68	YES	58.6136	24.73	YES

Fig. 10. Benchmark results of PERLS. The columns "PERLS is better" inform when the F corresponding to λ found by PERLS is lesser (and, by that, better) than the found by CRLS.

5. Conclusions

For λ=0.95, PERLS finds better results in all cases, except in the benchmarks *heat1* and *cstr1*. For *cstr1*, the difference of corresponding F values does not achieve the 0.5 %. Only in one case PERLS is not better than CRLS with λ=0,95.
For λ=0,98, PERLS finds better results in the 50% of cases. However, and this is important, for the other 50% of benchmarks, in all cases the difference of F oscillates between 0.06 % and 1.56 %. That is, PERLS improves or holds the results found with CRLS using λ=0.98.
We can say the parallel evolutionary methodology PERLS offers a good performance, and this encourages us to follow this research and in a near future to achieve a PERLS synthesis on reconfigurable hardware systems.

References

1. Söderström et al., T: "System Identification". Prentice-Hall, London (1989)
2. Ljung, L: System Identification – Theory for the User. Prentice-Hall, London (1999)
3. Ljung, L: System Identification Toolbox. The Math Works Inc. (1991)
4. "A Database for Identification of Systems". http://www.esat.kuleuven.ac.be /sista / daisy /
5. D.E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, Addison Wesley, Reading MA, 1989.
6. I. Rechenberg, Evolutionsstrategie: Optimierung technischer systeme nach prinzipien der biolgischen evolution. Stutgart: Formmann-Holzboog Verlag, 1973.