Time Series Forecasting based on Parallel Neural Network

J.M. Górriz† Carlos G. Puntonet*
Moisés Salmerón * Julio Ortega * Mohammed Aldasht*

Abstract— **In this paper we show a Parallel Neural Network (Cross-over Prediction Model) for time series forecasting implemented in PVM ("Parallel Virtual Machine") and MPI ("Message Passing Interface"), in order to reduce computational time. Parallelization is achieved twofold: (a) updating autoregressive parameters using a genetic algorithm (GA) and (b) evaluating the overall prediction function via a parallel neural network. We implement the GA in two popular architectures of parallel processors (i.e hypercube and 2D-mesh) and discuss their time efficiency.**

Keywords— **Artificial Neural Networks (ANNS), Auto-Regressive Models (AR), Parallel Virtual Machine (PVM), Array and Hypercube Networks, Mask Functions, Quicksort, Genetic Algorithms.**

I. Introduction

VARIOUS techniques have been applied in order to forecast time series using data from the stock. There also exist numerous forecasting applications like those ones analyzed in [1]: signal statistical preprocessing and communications, industrial control processing, econometrics, meteorology, physics, biology, medicine, oceanography, seismology, astronomy y psychology. In this areas a great computational speed, including numerical modelling and simulation, is needed. One way of increasing it, considered for many years, is by using multiple processors operating on a single program (*parallel programming*), i.e Gill in 1958, Holland in 1959, etc. In section III we show CPM, a robust autoregressive statistical learning model for limited data set (i.e including elements of statistical learning theory), as presented in [2]. The main disadvantage of other models for exogenous data inclusion, like PCA (*Principal Components Analysis*) or ICA (*Independent Component Analysis*), was that estimators for higher order statistics were useless under these conditions and could contaminate the raw data ([14] and [4]). CPM avoided it extending and applying the Regularization Theory [5] to a set of time series assuming linear influences of the extra series in the forecast, but with high computing speed demand. In the next sections, V and IV, we will give a solution to this problem using parallel programming on a Message-Passing Multicomputer platform using SIMD (*single instruction stream - multiple data stream*) computers and a SPMD (*single program multiple data*) programming structure [6]. In section IV, we choose master-slave arrangement whereby the single master program is first executed and all others (*slaves*) are spawned from this master to get the ANNs output. In order to implement the GA in CPM we will use and compare 2D-Mesh and Hypercube interconnection networks (V). They are examples of very popular completely connected networks because of the ease of layout and expandability.

†Dep. Electronics, University of Cdiz, E.P.S. Algeciras, E-11202 Algeciras (Spain). Email: juanmanuel.gorriz@uca.es.
*Dep. Architecture and Computer Tech., University of Granada,E-18071 Granada (Spain). Email: carlos@atc.ugr.es.

II. Basis PVM and MPI

In this work we use PVM and MPI, popular software packages for workstation cluster parallel programming, to get time computing simulations. PVM is a software system that enables a collection of heterogeneous computers to be used as a coherent and flexible concurrent computational resource. The Message-Passing Interface or MPI is a library of functions and macros intended for use in programs that exploit the existence of multiple processors by message passing. The message passing model has the following characteristics:

- It is intended as a standard implementation of the "message passing" model of parallel computing.
- Each process has purely local variables, and there is no mechanism for any process to directly access the memory of another.
- Sharing of data between processes takes place by message passing, that is, by explicitly ending and receiving data between processes.
- A primary reason for the usefulness of this model is that it is extremely general. Essentially, any type of parallel computation can be cast in the message passing form.
- In addition, this model can be implemented on a wide variety of platforms, from shared-memory multiprocessors to networks of workstations and even single-processor machines.
- Generally allows more control over data location and flow within a parallel application than in, for example, the shared memory model. Thus programs can often achieve higher performance using explicit message passing. Indeed, performance is a primary reason why message passing is unlikely to ever disappear from the parallel programming world.

The primary goals addressed by message passing model languages are:

- Provide source code portability. These programs should compile and run as-is on any platform.

- Allow efficient implementations across a range of architectures.
- They also should offer a great deal of functionality including a number of different types of communication.
- Support for heterogeneous parallel architectures.
- *Standardization*, *Availability* for all architectures distributed memory, shared memory, and clusters and *Portability*.

III. AR models using Artificial Neural Networks.

The prediction model is shown in figure 2. We consider a data set consisting in some correlated series and try to build a forecasting function $\mathbf{P}$, for one of the set of signals $\{series_1, \ldots, series_S\}$, which allows exogenous information coming from the other series. If we consider just one series [4] the individual forecasting function can be expressed in term of RBFs as [7]:

$$\mathbf{F(x)} = \sum_{\mathbf{i=1}}^{\mathbf{N}} \mathbf{f_i(x)} = \sum_{\mathbf{i=1}}^{\mathbf{N}} \mathbf{h_i} \cdot \exp\left\{ \frac{||\mathbf{x} - \mathbf{c_i}||^2}{\mathbf{r_i^2}} \right\} \quad (1)$$

where $\mathbf{x}$ is a p-dimensional vector input at time $\mathbf{t}$, $\mathbf{N}$ is the number of neurons (RBFs) , $\mathbf{f_i}$ is the output for each neuron $i-th$, $\mathbf{c_i}$ is the centers of $i-th$ neuron which controls the situation of local space of this cell and $\mathbf{r_i}$ is the radius of the $i-th$ neuron. The global output is a linear combination of the individual output for each neuron with the weight of $\mathbf{h_i}$. Thus we are using a method for moving beyond the linearity where the core idea is to augment/replace the vector input $\mathbf{x}$ with additional variables, which are transformations of $\mathbf{x}$, and then use linear models in this new space of derived input features. RBFs are one of the most popular kernel methods for regression over the domain $\mathbf{R^n}$ and consist on fitting a different but simple model at each query point $\mathbf{c_i}$ using those observations close to this target point in order to get a smoothed function. This localization is achieved via a weighting function or kernel $\mathbf{f_i}$.

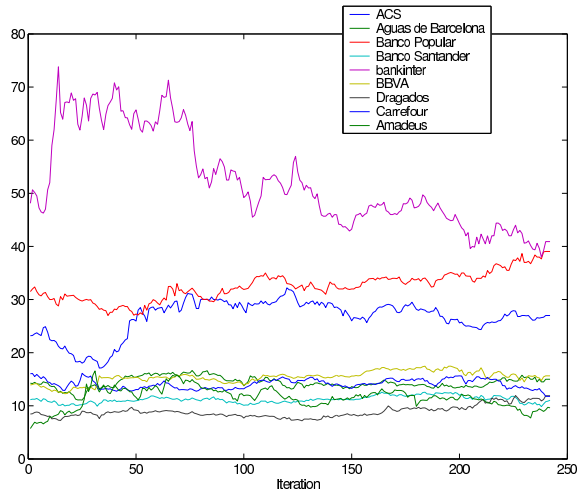

Fig. 1. Set of time Series used in Simulations.

The parameters of this endogenous model are updated using a suitable algorithm [14]. In this point we have to introduce the concept of actual risk, $R[F]$:

$$R[F] = R_{emp} + R_{reg} \quad (2)$$

where R_{emp} is the empirical risk, that is, a function of the current samples, and R_{reg} is the regularization term that penalizes large variations in the prediction function [15]. The minimization of this functional provides the parameter update.

We apply/extent this regularization concept to extra time series, i.e figure 1, including one row of neurons, equation 1, for each series and weight this values via a factor $\mathbf{b_{ij}}$ (AR model). Finally the overall smoothed prediction function for the stock $\mathbf{j}$ will be defined as:

$$\mathbf{P_j(x)} = \sum_{\mathbf{i=1}}^{\mathbf{S}} \mathbf{b_{ij} F_i(x, j)} \quad (3)$$

where $\mathbf{F_i}$ is the partial smoothed function of each series, $\mathbf{S}$ is the number of input series and $\mathbf{b_{ij}}$ are the weights for j-stock forecasting. Hence we assume *linear time dependent influence* among the set of series in the forecast. Obviously one of these weight factors must be quite relevant in this linear fit ($\mathbf{b_{jj}} \sim \mathbf{1}$, or auto weight factor).

We can use matrix notation to include the set of forecasts in an S-dimensional vector $\mathbf{P}$ ($\mathbf{B}$ in 2):

$$\mathbf{P(x)} = \mathbf{diag(B \cdot F(x))} \quad (4)$$

where $\mathbf{F} = (\mathbf{F_1}, \ldots, \mathbf{F_S})$ is a $S \times S$ matrix with $F_i \in \mathbf{R^S}$ and B is an $S \times S$ weight matrix. The operator *diag* extract the main diagonal. In CPM controlling input space dimension and neural resources are fundamental tasks due to *curse of dimensionality* and *overfitting* problems.

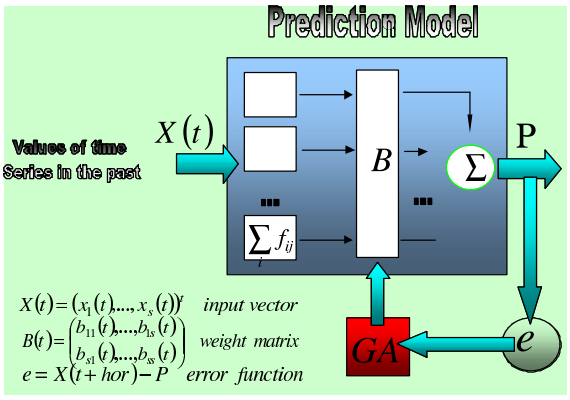

Fig. 2. Schematic representation of the Auto-regressive CPM model with adaptive radius, centers and input space ANNs. This improvement consists on neural parameters adaptation when input space increases, i.e. RBF centers and radius are statistically updated when dynamic series change takes place.

We can include equation 4 in the Generalized Additive models for regression proposed in supervised learning [8]:

$$\mathbf{E}\{\mathbf{Y}|\mathbf{X_1},\ldots,\mathbf{X_n}\} = \alpha + \mathbf{f_1}(\mathbf{X_1}) + \ldots + \mathbf{f_n}(\mathbf{X_n}) \quad (5)$$

where $\mathbf{X_i s}$ usually represent predictors and $\mathbf{Y}$ represents the system output; $\mathbf{f_j s}$ are unspecific smooth ("nonparametric") functions. Thus we can fit this model minimizing the mean square error function or other methods presented in [8].

On the other hand, CPM uses a genetic algorithm for $\mathbf{b_i}$ parameters fitting (using Least Mean Squares Error in a set of samples) although other techniques can be used such as Singular Value Decomposition. A canonical GA is constitute by operations of parameter encoding, population initialization, crossover, mutation, mate selection, population replacement etc. Our encoding parametric system consist on the codification into genes and chromosomes or individuals as string of binary digits using one's complement representation somehow there are other encoding methods also possible i.e [9], [10],[11] or [12] where the value of each parameter is a gene and an individual is encoded by a string of real numbers instead binary ones. In the Initial Population Generation step we assume that the parameters lie in a bounded region $[0, 1]$ (in the edge on this region we can reconstruct the model without exogenous data) and N individuals are generated randomly. After the initial population $\mathbf{N}$ is generated the fitness of each chromosome $\mathbf{I_i}$ is determined using the function:

$$\aleph(I_i) = \frac{1}{\mathbf{e(I_i)}} \quad (6)$$

(To amend the convergence problem in the optimal solution we add some positive constant to the denominator).

Another important question in canonical GA is defining Selection Operator. New generations for mating will be selected depending their fitness function values roulette wheel selection. Once we select the newly individuals, we apply crossover ($\mathbf{P_c}$) to generate two offspring which will be applied, in the next step, the mutation Operator ($\mathbf{P_m}$) to preserve from premature convergence. We improve speed convergence of the algorithm including some mechanisms like elitist strategy in which the best individual in the current generation always survived into the next.

The GA used in the forecasting function 3 has a error absolute value start criterion. Once it starts, it uses the values (or individual) it found optimal (elite) the last time, and apply local search around this elite individual. Thus we do an efficient search around an the elite individual (set of $\mathbf{b_i s}$) every time GA section is triggered. Results using the complete model are acceptable depending on data set, target series, etc..(see figure 3).

Computational time in GA depends on the encoding length, number of individuals and genes. Because of the probabilistic nature of the GA-based method, the proposed method almost converges to a global optimal solution on average. In our simulation we didn't find any nonconvergent case.

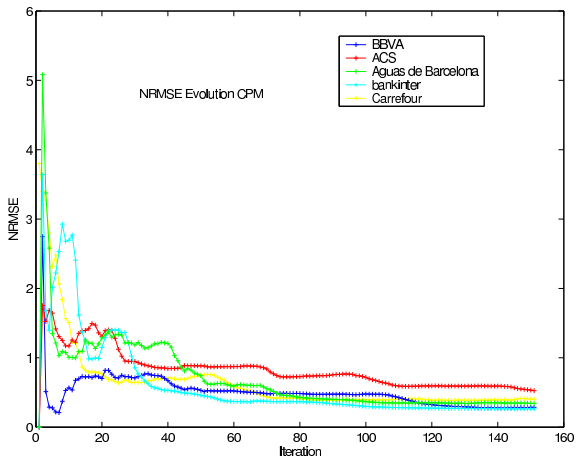

Fig. 3. Error Evolution of selected Series from figure 1.

IV. Master-Slave Configuration for ANN system.

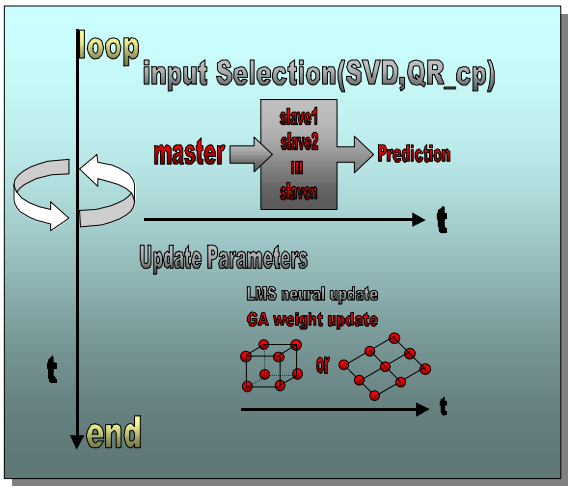

Fig. 4. Schematic representation of Parallel Neural Network.

In order to implement the model presented in section III (see fig 4)we choose M-S configuration to get the overall prediction. Each slave uses parameters (input space, neural and weight parameters) corresponding to each series, so the number of signals fixes the number of processors spawned by PVM. The master launches all the tasks (number of input signals) and received the partial prediction from the slaves ($\mathbf{P_j} = \sum_{\mathbf{i=1}}^{\mathbf{N(j)}} \mathbf{f_i}$, where $N(j)$ is the number of neurons of layer j) in each prediction tentative.Once every layer is done the master compute the overall prediction ($\mathbf{P} = \sum_{\mathbf{j=1}}^{\mathbf{S}} \mathbf{b_j} \cdot \mathbf{P(j)}$, where $\mathbf{S}$ is the number of signals) The parallel neural network (PNN) implemented gives good results as shown in [2]. In this section we care about computational speed i.e in the figure 7 we plot computational time versus number of processors and problem dimension in sequential model and parallel model. Obviously results improve sequential computing and computational time (assuming each slave works the same time) does not depend on number of slaves spawned and is proportional to the dimension of vectors used.

V. Hypercube and 2D-Mesh Configuration for GA implementation.

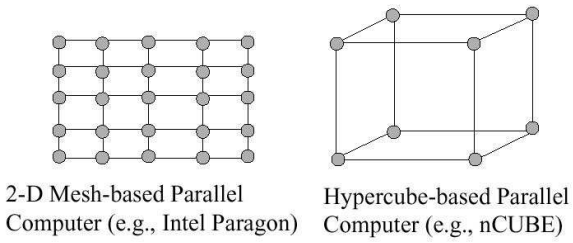

Fig. 5. Hypercube and 2D-Mesh Configuration

2D-Mesh can be created having each node connected to all its four nearest neighbors as shown in figure 5. The diameter of a $m \times m$ mesh, where m is the number of nodes, is $2 \cdot m - 1$ since to reach one corner from the opposite one requires a path across $m-1$ nodes and down $m-1$ nodes. If the free end of a mesh connects the opposite side we have a torus, in our case it is useless. Each processor or node has the communication functions displayed in table I.

TABLE I
Communication Routines in 2D Mesh

Communication Routines

`int upnode (int mytid) {`
`  int node=mytid -WIDTH;`
`  if (node<0)`
`  nodo=(HEIGHT-1)*WIDTH+mytid;`
`return node;}`

`int downnode (int mytid) {` `return (mytid+WIDTH) Mod NPROC;}`

`int rightnode (int mytid) {`
`  if (me Mod WIDTH==WIDTH-1)`
`  return me-(ANCHURA-1);`
`  else`
`  return me+1;}`

`int leftnode (int mytid) {`
`  if (me Mod WIDTH==0)`
`  return me+(WIDTH-1);`
`  else`
`  return me-1;}`

In a Hypercube each node is connected to other in each dimension of the network. In the figure 5 we present 3D hypercube.A notable advantage of the hypercube is that the diameter is given by $log_2 m$ which grows slower than the mesh with increasing m. The communication function in the hypercube is implemented by the exclusive-OR function between node position and 2^i, where i is each dimension. We avoided the unnecessary communications between nodes using a mask function. The hypercube networks became popular for constructing message-passing multicomputers after the pioneering research system called Cosmic Cube was constructed at Caltech in 1985 (Seitz).

Both Networks will are feeded with N possible solutions coming from the main GA routine. These solutions are distributed in the nodes, sharing the computational effort and communicating themselves using standard PVM and MPI routines. The processors compute GA standard operations and sort solutions according to their fitness function values using *Quicksort* [1] method [13] and send the elite components to the neighbor. *Quicksort* is one of the fastest and simplest sorting algorithms originally published in [16]. It works recursively by a divide and conquer strategy and has a time complexity of $\emptyset(n \cdot log(n))$ on average and $\emptyset(n^2)$ in the (unlikely) worst case. Other sorting has better time complexity in worst case but not on average, i.e heapsort and mergesort. In the table II we compare efficiency of different sorting methods depending on number of comparisons C (first row) and movements M necessary, where n is the number of elements in sorting. We compute the maximum, minimum and mean of the functions C and M in the $n!$ combinations.

Finally, after all communications between nodes, we get in node 0 elite solutions sorted with respect to its fitness function. In this process nodes reduce time complexity on average to:

$$\emptyset(\frac{n}{m} n \cdot log(\frac{n}{m}) + \delta(n,m)) \tag{7}$$

where $\delta << \emptyset(\log n)$ is computational time in pivot selection, vectors broadcast, data split, data communication an data merge of m couples of sorted vectors; and $\emptyset(\log n)$ is the optimal parallel time complexity (Leighton, 1984). Our method, using hypercube networked processors is similar to *Hyperquicksort* (Wagar, 1987)[6], follows the stages:

1. Each processor sorts its list sequentially.
2. The processors in the "lower" subelement (in 2D-mesh or hypercube) send the sorted vectors to neighbors in the "upper" subelement (in 2D-mesh or hypercube) in each step dimension.
3. Each processor merges the list received with its list to obtain a sorted list, calling not used nodes in recursive sorting process.

With this algorithm we reduce the number of communications in hypercube to $2^{\log m} - 1$. In *Hyperquicksort* the number of communications between nodes is $log(m) \cdot 2^{\log(m)}$ (like a common hypercube). Anyway time complexity is similar (but lower![2]) to *Hyperquicksort* because we call not used nodes in "lower" subelements from nodes in "upper" subelements when sorting merged lists.

VI. Results and Conclusions.

The master-slave configuration implements neural system for prediction. Obviously this arrangement improves the computational speed of sequential CPM, due to parallel neural work, for each series, is now performed concurrently. This fact is proved in figure 7 where MPI libraries were used

[1] this improves the efficiency of the GA, mainly in mate selection to offspring generation and elitist strategies.

[2] Our algorithm avoids communications related with pivoting element propagation.

TABLE II
DIFFERENT SORTING METHODS. 1^{st} COMPARISONS 2^{sd} MOVEMENTS

Method	Min.	Mean	Max.
Direct	$n-1$	$\frac{n^2+n-2}{4}$	$\frac{n^2-n}{2}-1$
Inser.	$2(n-1)$	$\frac{n^2-9n-10}{4}$	$\frac{n^2+3n-4}{2}$
Direct	$\frac{n^2-n}{2}$	$\frac{n^2-n}{2}$	$\frac{n^2-n}{2}$
Selec.	$3(n-1)$	$n(\ln n+0.57)$	$\frac{n^2}{4}+3(n-1)$
Quick	$\frac{n^2-n}{2}$	$\frac{n^2-n}{2}$	$\frac{n^2-n}{2}$
Sort	0	$(n^2-n)\cdot 0.75$	$(n^2-n)\cdot 1.5$

to get results. In this simulation (8 nodes Cluster Pentium II 332MHz 512Kb Cache) we fixed a high number of neurons (1000) in layers and input space dimension(500) (worst case). The master spawns n_{proc} tasks to slaves and they compute set of partial predictions $\sum_{i=1}^{\mathbf{N(i)}} \mathbf{b_i}\cdot\mathbf{F_i}$ where $\mathbf{N(i)} \in \{\mathbf{mod}(\frac{\mathbf{num_{series}}}{\mathbf{n_{proc}}}), \mathbf{mod}(\frac{\mathbf{num_{series}}}{\mathbf{n_{proc}}})+\mathbf{1}\}$ and replay them to the master that compute the overall prediction value.

The sequential response of our prediction system is defined by the curve peaks corresponding to one processor configuration, approximately. In figure 6

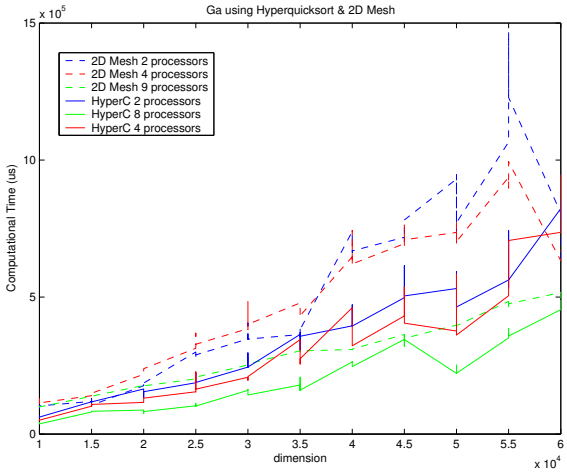

Fig. 6. Time Processing in GA using Parallel Hypercube & 2DMesh configuration.

we show time results obtained from the GA implementation, using PVM software, on the cluster using Hypercube and 2D-mesh Networked processors. The presence of 8 nodes contributes to divide-conquer strategy of *Quicksort*, reducing computational time.

If we arrange the proposed method in section V on a pentium 166MHz using PVM we get the results in figure 8. The number of processors are the number of time series in this simulation (9). The presence of just one processor gives trivial results (PVM spawns n virtual processors in Hypercube or 2D-Mesh configuration on just one real processor!) as we expect. Theres an increasing computational time with increasing problem dimension(larger amount of data) and with increasing number of processors (larger communications between nodes).

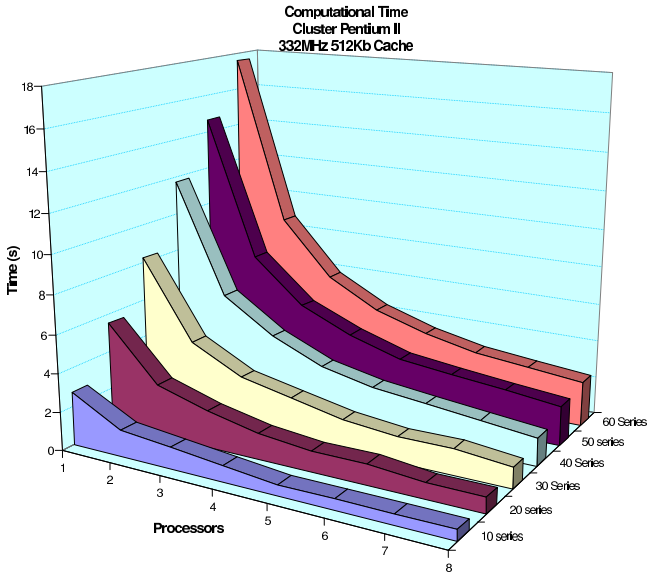

Fig. 7. Time Processing in Parallel master-slave configuration for ANN system.

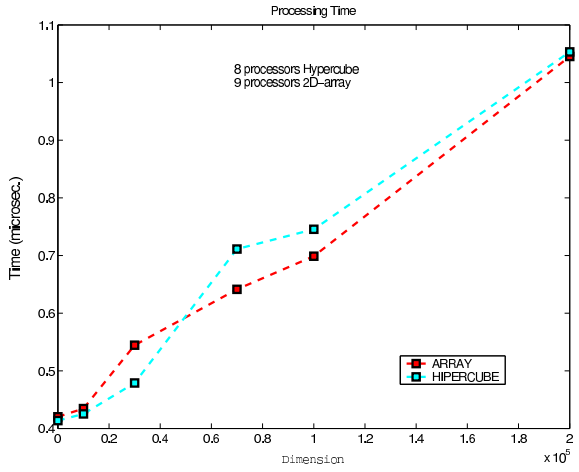

Fig. 8. Time Processing in sequential GA configuration.

REFERENCES

[1] D.S.G. Pollock, *A handbook of time series analysis, signal processing and dynamics*, Academic Press,1999.

[2] J.M. Górriz, Carlos G. Puntonet , J.J de la Rosa, Moisés Salmerón, *New Model For Time-Series Forecasting using RBFS and Exogenous Data*, ISDA 2003, Tulsa, USA, August 2003.

[3] J.M. Górriz-Sáez, *Predicción con Redes Neuronales y Técnicas de Separación de Senales*, University of Cádiz ,Departamento de Ing. de Sistemas y Aut. Tec. Electrónica y Electrónica, 2003.

[4] M. Salmerón-Campos, *Predicción de Series Temporales con Redes Neuronales de Funciones Radiales y Técnicas de Descomposición Matricial*, University of Granada, Departamento de Arquitectura y Tecnología de Computadores, 2001.

[5] T.A. Tikhonov , V.Y. Arsenin, *Solutions of Ill-Posed Problems*, Winston, Washington D.C., USA, 1977.

[6] B. Wilkinson, Michael Allen, *Parallel Programming*, Prentice Hall, New Jersey, USA, 1999.

[7] J. Moody, C. J. Darken, *Fast Learning in Networks of Locally-tuned processing units*, Neural Computation, volume 1,pages 284-294, 1989.

[8] T. Hastie, R. Tibshirani, J. Friedman, The elements of Statistical Learning, Springer, Berlin, 2000.

[9] Z. Michalewicz, Genetic Algorithms + Data Structures = Evolution Programs, Springer-Verlag, Berlin 1992.
[10] S. Matwin, T. Szapiro, K. Haigh, Genetic Algorithms Approach to a Negotiation Support System, IEEE Trans. Syst., Man. Cybern, volume 21, pages 102-114, 1991.
[11] S. Chen, Y. Wu, Genetic Algorithm optimization for blind channel identification with higher order cumulant fitting, IEEE Trans. Evol. Comput.,volume 1, pages 259-264, 1997.
[12] L. Chao, W. Sethares, Non linear parameter estimation via the genetic algorithm, IEEE Transactions on Signal Processing, volume 42, pages 927-935, 1994.
[13] N. Wirth, Algorithms + Data Structures = Programs, Prentice Hall, New Jersey, USA, 1999.
[14] J.M. Grriz, Algoritmos Hbridos para la Modelizacin de Series Temporales usando Tcnicas AR-ICA, In Press, Ph Thesis, 2003.
[15] V. Vapnik, The nature of Statistical Learning Theory, Springer, 1999.
[16] C.A.R. Hoare, *Quicksort*, Computer Journal, Vol. 5, 1, 10-15, 1962.