Pergamon

0045-7906(95)00025-9

Computers Elect. Engng Vol. 22, No. 1, pp. 61–84, 1996
Copyright © 1996 Elsevier Science Ltd
Printed in Great Britain. All rights reserved
0045-7906/96 $15.00 + 0.00

PARALLEL SIMULATION OF CELLULAR NEURAL NETWORKS

L. FORTUNA, G. MANGANARO, G. MUSCATO† and G. NUNNARI

Dipartimento Elettrico, Elettronico e Sistemistico, Università di Catania, V.le A.Doria 6, 95125, Catania, Italy

(Received for publication 25 August 1995)

Abstract—In this paper a new simulator for cellular neural networks (CNN) called PSIMCNN is presented. It has been studied for a parallel general-purpose computing architecture based on transputers. The Gauss–Jacobi waveform relaxation (WR) algorithm has been adopted. It has been analytically proved that the WR algorithm is convergent for the most common CNN models. Implementation issues have been described and some experimental results comprising classical CNN and Chua circuits CNN simulations have been presented. Parallelism and performances have been investigated. Moreover, a comparison with the CNN–HAC simulator is reported.

Key words: Cellular neural networks, simulators, non-linear circuits, transputer, parallel processing.

1. INTRODUCTION

Cellular neural networks have attracted the interest of many researchers, since their first introduction in 1988 by Chua and Yang [1,2], because of their useful features. Chua and Yang's first CNN model has subsequently been generalized by a number of scientists and engineers, leading to a vast variety of alternative models [3]. Some of these generalized models have allowed strange and impressive applications that, maybe, no one thought of when the first CNN model was invented [4–7].

One of the main problems for a designer is to choose a template that allows him to solve a particular job with a CNN. Many attempts have been made in order to develop analytic or systematic methods to determine the template values [8–10]. Although they are encouraging and some of them may show the right way to cope with this problem, they only allow some particular cases to be dealt with. Moreover, they are time-consuming. Actually, the problem is often solved in a heuristic way or by trial and error.

A CNN is a high-order non-linear dynamic system, so the analytical study of its behaviour may be really hard. This means that observation of its behaviour (e.g. state trajectories) can be very indicative and useful.

These are two important cases in which it is really useful to have a CNN simulator.

Some CNN simulators already exist. The most popular one is CNN–HAC [11]. Its principal features will be summarized in Section 5. A fundamental characteristic, however, is that it needs a dedicated hardware accelerator board to perform the computations. It permits some pre-defined CNN models to be simulated but, unfortunately, it is not possible for the user to simulate other kinds of CNN models.

These two reasons lead to the need for a new simulator. This should have some advantages with respect to CNN–HAC (these are investigated in Section 5) but mainly the following features have to be implemented:

(1) it should be developed for a general-purpose computing architecture;
(2) it should be an "open" software: i.e. it should be possible for the user, with minimum modifications, to use it for his particular requirements, e.g. to simulate new CNN models.

†To whom all correspondence should be addressed.

What does a CNN simulator do? It solves the non-linear system of ordinary differential equations (ODEs) that describes the dynamic behaviour of the CNN. This is a really time-consuming activity so two ways should be followed together to cope with it efficiently:

(1) adopting parallel computing hardware;
(2) using new integration algorithms.

The problem of the numerical solution of ODE systems is not new for engineers and mathematicians and a variety of new parallel processing algorithms have been developed due to the relatively recent availability of low-cost multiprocessor machines [12]. An interesting class of integration algorithms is the family of waveform relaxation methods (WR) [13–15]. They are iterative methods that are mainly used in the simulation of VLSI systems due to their intrinsic velocity and because they can be efficiently parallelized. So this could be the answer to our problems. However, is any ODE system solvable with the waveform relaxation methods? Unfortunately, in general, the answer is "NO".

This means that if we want to use a WR method we must be sure that this method will work with our ODE system. There are some conditions that must be satisfied.

In this paper a new simulator for a general-purpose parallel architecture called PSIMCNN is proposed (Parallel SIMulator for CNN). It allows accurate simulations of several different CNN models by means of a Gauss–Jacobi waveform relaxation integration method. In particular, the case of a classical Chua and Yang CNN model [1,2] and the relatively recent case of a CNN where Chua's circuit is used as the basic cell [4,6,16] are presented.

In Section 2 some preliminary theoretical results, useful for the following sections, are reported. In Section 3, some new theoretical results are introduced; in particular, the convergence of the WR methods to the desired ODE system solution is proved (some of the proofs have been reported in the Appendix). In Section 4 the actual simulator implementation is described. In Section 5 some experimental results are reported and a comparison with CNN–HAC is made.

2. PRELIMINARIES

2.1. *CNN models*

In this paragraph only the mathematical models of the CNN that will be used in the next sections will be recalled; more exhaustive information on CNN basics can be found in [1–10].

Let us consider the equations of a single-layer CNN with non-linear and delay templates [3]:

$$C\dot{x}_{ij}(t) = -\frac{1}{R_x}x_{ij}(t) + I + \sum_{C(k,l)\in N_r(i,j)} \hat{A}_{ij;kl}(y_{kl}(t), y_{ij}(t)) + \sum_{C(k,l)\in N_r(i,j)} \hat{B}_{ij;kl}(u_{kl}(t), u_{ij}(t))$$
$$+ \sum_{C(k,l)\in N_r(i,j)} A^{\tau}_{ij;kl}y_{kl}(t-\tau) + \sum_{C(k,l)\in N_r(i,j)} B^{\tau}_{ij;kl}u_{kl}(t-\tau) \tag{1a}$$

with

$$y_{ij}(t) = g(x_{ij}(t)) \text{ with } g(x) = \tfrac{1}{2}[|x+1| - |x-1|]$$
$$1 \leqslant i, k \leqslant M$$
$$1 \leqslant j, l \leqslant N \tag{1b}$$

where x_{ij} are the state variables, u_{ij} are the inputs and y_{ij} are the outputs. The non-linear function $g(x)$ is shown in Fig. 1. A particular case of this CNN model is the classical Chua and Yang model [1,2]:

$$C\dot{x}_{ij}(t) = -\frac{1}{R_x}x_{ij}(t) + \sum_{C(k,l)\in N_r(i,j)} A(i,j;k,l)y_{kl}(t) + \sum_{C(k,l)\in N_r(i,j)} B(i,j;k,l)u_{kl} + I. \tag{2}$$

These two models are well-known to all people interested in CNNs. A less-known model (but not a less interesting one) is the CNN model described by the following equations [4]:

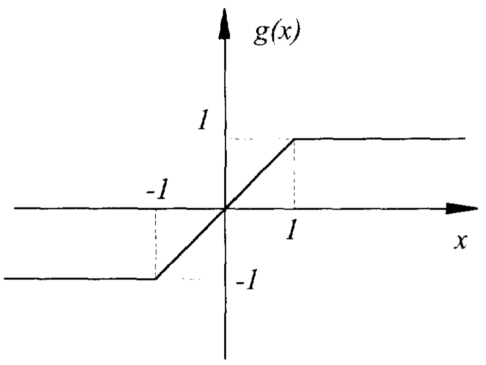


Fig. 1. Chua and Yang CNN nonlinearity.

$$\dot{x}_{i,j} = \alpha(y_{i,j} - h(x_{i,j})) + D[x_{i-1,j} + x_{i+1,j} + x_{i,j-1} + x_{i,j+1} - 4x_{i,j}]$$
$$\dot{y}_{i,j} = x_{i,j} - y_{i,j} + z_{i,j}$$
$$\dot{z}_{i,j} = -\beta y_{i,j} \tag{3a}$$

with

$$h(x) = \begin{cases} m_1 x + (m_0 - m_1) + \varepsilon & x \geqslant 1 \\ m_0 x + \varepsilon & -1 \leqslant x \leqslant 1 \\ m_1 x + (m_1 - m_0) + \varepsilon & x \leqslant -1 \end{cases}$$
$$1 \leqslant i, k \leqslant M$$
$$1 \leqslant j, l \leqslant N \tag{3b}$$

where $x_{i,j}$, $y_{i,j}$, $z_{i,j}$ are the state variables and α, $\beta \in \mathscr{R}$ some constants. The non-linear function $h(x)$ is shown in Fig. 2. These equations represent a CNN in which the classical cell has been replaced with the 3rd-order Chua circuit, so any cell contributes with three state variables. Chua's circuit is widely known for its amazing ability to exhibit a variety of bifurcation and chaotic phenomena with just a few simple circuit components. The classical Chua's circuit equation can be obtained in equation (3) simply by setting $D = 0$. The constant D, which weights the interactions with the neighbouring cells, is called the *diffusion coefficient*. This new CNN model has been studied in [4] because it can support travelling waves (called *autowaves*) propagating through the coupled cells.

In the above models we call *inner cells* those which have a complete neighbourhood. All other cells are called *boundary cells*. An important role, sometimes underestimated, is played by the *boundary conditions*. In models (1) and (2), for example, it is often assumed that the contribution of the missing cells in the array border is zero whilst with model (3) in Refs [4] and [16] the authors assumed *zero flux boundary conditions*, i.e. they assumed the same value of the boundary cell for

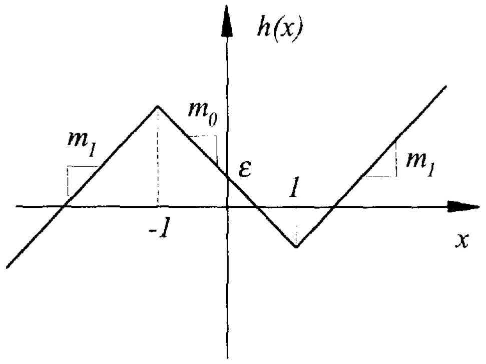


Fig. 2. Chua's circuit nonlinearity.

its missing neighbour cells (note that consistency is maintained due to its 4-connected type structure).

2.2. Gauss–Jacobi waveform relaxation method

Waveform relaxation (WR) methods are iterative methods used to solve large ODE systems [12–15]. They are used in the circuit simulation of VLSI systems. In particular, the *Gauss–Jacobi waveform relaxation method* (GJ-WR) has been considered.

Let us consider the ODE system:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x},t) \quad \mathbf{x}(t_i) = \mathbf{x}_0 \tag{4}$$

where $\mathbf{x} \in \mathscr{R}^n$ $\mathbf{f}$: $\mathscr{R}^n \times \mathscr{R}^+ \longrightarrow \mathscr{R}^n$. For $\mathbf{z}$, $\mathbf{w} \in \mathscr{R}^n$, let $\tilde{\mathbf{f}}$: $\mathscr{R}^n \times \mathscr{R}^n \times \mathscr{R}^+ \longrightarrow \mathscr{R}^n$ be defined by:

$$\tilde{f}_i(\mathbf{z}, \mathbf{w}, t) = f_i(w_1, \ldots, w_{i-1}, z_i, w_{i+1}, \ldots, w_n, t) \quad i = 1, \ldots, n. \tag{5}$$

Thus the ith components of $\tilde{\mathbf{f}}(\mathbf{z}, \mathbf{w}, t)$ is found by evaluating the ith components of $\mathbf{f}(\cdot, t)$ at a point whose components are those of $\mathbf{w}$, except the ith, whose value is z_i.

Now let us consider the norm definitions:

$$\| g \|_{[t_i,t_f]} = \max_{t \in [t_i,t_f]} |g(t)|; \quad \text{where } g:[t_i,t_f] \longrightarrow \mathscr{R} \quad (\textit{single waveform norm}) \tag{6}$$

$$\| \mathbf{x} \|_{[t_i,t_f]} = \max_{1 \leqslant i \leqslant n} (\| x_i(t) \|_{[t_i,t_f]}) \quad (\textit{waveform vector norm}). \tag{7}$$

Remark: the single waveform norm is useful because it releases us from time dependence. In Section 3, the definition interval $[t_i, t_f]$ will not be indicated, in order to avoid clutter.

The GJ-WR is then described in a meta-language as:

```
Algorithm (Gauss–Jacobi WR):
k ←0;
set $\mathbf{x}^{(0)}(t) = \mathbf{x}_0$ $t \in [t_i, t_f]$;
repeat{
  k ←k + 1
  forall (j in n){
    solve
      $\dot{x}_j^{(k)} = \tilde{f}_j(\mathbf{x}^{(k)}, \mathbf{x}^{(k-1)}, t)$
    for $x_j^{(k)}(t)$ $t \in [t_i, t_f]$ with the initial condition $x_j^{(k)}(t_i) = x_{0j}$
  }
}
until $\| \mathbf{x}^{(k)} - \mathbf{x}^{(k-1)} \|_{[t_i,t_f]} \leqslant \varepsilon_a$     (control condition).
```

The quantity called ε_a is used to monitor the global accuracy. Sometimes relative global accuracy is used, so the control condition is replaced with:

$$\frac{\| \mathbf{x}^{(k)} - \mathbf{x}^{(k-1)} \|_{[t_i,t_f]}}{\| \mathbf{x}^{(k)} \|_{[t_i,t_f]}} \leqslant \varepsilon_r. \tag{8}$$

It is important to distinguish between local and global accuracy. We talk about local accuracy when the error introduced during one ODE integration step is considered, while we talk about global accuracy when the whole waveform, as in equation (8), is considered.

This algorithm generates a sequence of functions $\mathbf{x}^{(k)}$ called *waveforms*. If some conditions on $\mathbf{f}(\cdot, t)$ are satisfied then this sequence converges to the solution of the ODE system. This is stated by the following theorem [12–15]:

Theorem 2.1: Suppose $\tilde{\mathbf{f}}$ is uniformly Lipschitz (i.e. $\exists K_1$, K_2 such that $\| \tilde{\mathbf{f}}(\mathbf{z}', \mathbf{w}', t) - \tilde{\mathbf{f}}(\mathbf{z}'', \mathbf{w}'', t) \| \leqslant K_1 \| \mathbf{z}' - \mathbf{z}'' \| + K_2 \| \mathbf{w}' - \mathbf{w}'' \|$)
then $\mathbf{x}^{(k)}(\cdot)$: $[t_i, t_f] \longrightarrow \mathscr{R}^n$ converges uniformly to $\mathbf{x}(\cdot)$: $[t_i, t_f] \longrightarrow \mathscr{R}^n$ and $\forall t \in [t_i, t_f]$

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), t) \quad \mathbf{x}(t_i) = \mathbf{x}_0. \tag{9}$$

■

3. NEW THEORETICAL RESULTS

In this section some innovative theoretical results, useful for the aims of simulation, are introduced. The topic of the first paragraph is the suitability of WR methods for the proposed models.

In the second paragraph a new integration method is proposed; it has been developed for the particular equation that has to be solved. The idea comes from the fact that the three proposed models have piece-wise linear nonlinearities (the two non-linear functions $g(\cdot)$, $h(\cdot)$). In particular these two functions change their slopes in ± 1 (called *breakpoints*). This means that when the equations are integrated by using a classical algorithm (e.g. the Euler method or the Runge–Kutta method), the transition of a state variable through a breakpoint can introduce a significant local integration error.

The idea consists of reducing this local error by a more accurate prediction when these circumstances occur. The same concept can be applied to several explicit integration methods. In particular, the Ralston method has been modified.

3.1. *Convergence of WR methods with CNN equations*

First of all, let us consider the following Lemmas.

Lemma 3.1: A linear function $g(x) = a \cdot x + b$, $g : \mathscr{R} \to \mathscr{R}$ is always uniformly Lipschitz.

Proof: Just choose $K \geqslant |a|$:

$$|g(x_1) - g(x_2)| = |a| \cdot |x_1 - x_2| \leqslant K|x_1 - x_2| \quad \forall x_1, x_2 \in \mathscr{R}. \tag{10}$$

■

Lemma 3.2: A piece-wise linear (PWL) function $g : \mathscr{R} \to \mathscr{R}$ with finite slope segments is always uniformly Lipschitz.

Proof: Let S be the set of slope coefficients of $g(\cdot)$. So $K \geqslant \max(S)$ can be chosen. ■

We aim to simulate the CNN models presented in Section 2 using a GJ-WR method, so the first step is to prove the convergence of the method with the proposed ODE systems. Let us start with model (1).

Theorem 3.1: *Convergence with Non-linear & Delay-type template CNN*. Given the non-linear ODE system:

$$C\dot{x}_{ij}(t) = -\frac{1}{R_x} x_{ij}(t) + I + \sum_{C(k,l) \in N_r(i,j)} \hat{A}_{ij;kl}(y_{kl}(t), y_{ij}(t)) + \sum_{C(k,l) \in N_r(i,j)} \hat{B}_{ij;kl}(u_{kl}(t), u_{ij}(t))$$

$$+ \sum_{C(k,l) \in N_r(i,j)} A^{\tau}_{ij;kl} y_{kl}(t - \tau) + \sum_{C(k,l) \in N_r(i,j)} B^{\tau}_{ij;kl} u_{kl}(t - \tau) \tag{11a}$$

$$y_{ij}(t) = g(x_{ij}(t)) \quad g(x) = \tfrac{1}{2}[|x + 1| - |x - 1|] \tag{11b}$$

$$x_{ij}(0) = x_{ij0} \quad C > 0 \;\; R_x > 0 \quad \begin{matrix} 1 \leqslant i, k \leqslant M \\ 1 \leqslant j, l \leqslant N \end{matrix} \tag{11c}$$

the WR algorithm is convergent for any initial state and any limited inputs if the non-linear functions $\hat{A}_{ij,kl}(\cdot,\cdot)$ $\hat{B}_{ij,kl}(\cdot,\cdot)$ are uniformly Lipschitz.

Proof: See the Appendix.

Remarks:

(1) This theorem proves the convergence of the WR method with model (1) so it proves the applicability of this method for this model.

(2) Model (2) is just a particular case of (1). In the Chua and Yang model the non-linear template functions have been replaced with simple constants. So the hypothesis of the uniform lipschitzianity of the templates is obviously always satisfied.

Let us now prove that model (3) is suitable for WR as well.

Theorem 3.2: *Convergence with Chua's circuit CNN*. Give the non-linear ODE system:

$$\dot{x}_{i,j} = \alpha(y_{i,j} - h(x_{i,j})) + D[x_{i-1,j} + x_{i+1,j} + x_{i,j-1} + x_{i,j+1} - 4x_{i,j}]$$
$$\dot{y}_{i,j} = x_{i,j} - y_{i,j} + z_{i,j}$$
$$\dot{z}_{i,j} = -\beta y_{i,j} \tag{12a}$$

$$h(x) = \begin{cases} m_1 x + (m_0 - m_1) + \varepsilon & x \geqslant 1 \\ m_0 x + \varepsilon & -1 \leqslant x \leqslant 1 \\ m_1 x + (m_1 - m_0) + \varepsilon & x \leqslant -1 \end{cases} \tag{12b}$$

$$x_{i,j}(0) = x_{ij0} \; y_{i,j}(0) = y_{ij0} \; z_{i,j}(0) = z_{ij0} \quad \begin{matrix} 1 \leqslant i \leqslant M \\ 1 \leqslant j \leqslant N \end{matrix}. \tag{12c}$$

The WR algorithm is convergent for any initial state.

Proof: See the Appendix.

With this theorem the applicability of the WR method for the last CNN model is proved.

3.2. *A new integration method for a particular kind of ODE*

In this paragraph a new explicit integration algorithm called *modified Ralston* is introduced. It consists of a Ralston method [17] (a particular second-order Runge–Kutta method) in which the prediction of the next instant state value is modified when a state variable crosses a breakpoint (see above). Therefore this method is suitable for any ODE containing a PWL nonlinearity with breakpoints in ± 1. Three "working zones" in the space of the considered state variable can be distinguished.

More precisely, the three working zones A, B, C are:

$$A = (-\infty, -1] \quad B =]-1, 1[\quad C = [1, +\infty). \tag{13}$$

Let us suppose that the ODE that we want solved is $\dot{x} = f(x, t)$ with $x(0) = x_0$.

Remark: this is a single equation, not a system.

Now, let us suppose, for example, that the considered non-linear function is $g(\cdot)$. When the state variable x varies within one of the three mentioned regions, this ODE is linear. It can be stated that it is a "local linear system". However, when x passes from one region to another, the nonlinearity shows its effects.

When this differential equation is integrated, the next state sample is predicted according to the current state sample. So this means that if the current sample is inside one region and the next sample is inside another one, then the prediction of the next sample value is done according to a different behavioural model (the previous one).

A new method to improve the prediction when this kind of transition occurs is now introduced.

Let us assume that the step-size h is sufficiently small so that the state variable x (we will indicate the ith sample of x with x_i) cannot "jump" between two non-adjacent working zones. When x is inside one of the working zones the conventional Ralston formula is used:

$$x_{i+1} = x_i + (\tfrac{1}{3}k_1 + \tfrac{2}{3}k_2)h$$
$$k_1 = f(x_i, t_i)$$
$$k_2 = f(x_i + \tfrac{3}{4}k_1 h, t_i + \tfrac{3}{4}h). \tag{14}$$

When x passes from x_i to x_{i+1}, with these two samples in two different regions, then it crosses the value x^*:

$$x^* = \begin{cases} 1 & \text{if} \quad (x_i \in B)\&(x_{i+1} \in C) \quad \text{or} \quad (x_i \in C)\&(x_{i+1} \in B) \\ -1 & \text{if} \quad (x_i \in A)\&(x_{i+1} \in B) \quad \text{or} \quad (x_i \in B)\&(x_{i+1} \in A) \end{cases}. \tag{15a}$$

So the simulated time t^* in which this transition happens can be obtained (see also Fig. 3):

$$\frac{t^* - t_i}{t_{i+1} - t_i} = \frac{x^* - x_i}{x_{i+1} - x_i}, \text{ i.e. } \frac{t^* - t_i}{h} = \frac{x^* - x_i}{x_{i+1} - x_i} \text{ then } t^* = t_i + h \cdot \frac{x^* - x_i}{x_{i+1} - x_i}. \tag{15b}$$

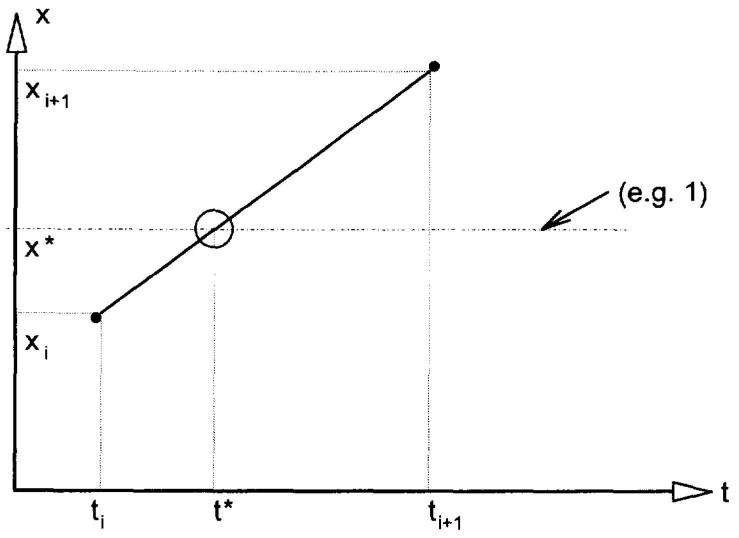


Fig. 3. Determination of the simulated instant t^*.

To be certain of this passage a (theoretically infinitesimal) quantity is defined, called δx, which can be chosen as:

$$\delta x = \frac{x_{i+1} - x^*}{L}. \tag{15c}$$

(In our simulations $L = 50$ is always chosen.) In this way a better value for x_{i+1} can be predicted with:

$$\bar{x} = x^* + \delta x \quad h^* = t_{i+1} - t^*$$

$$k_1^* = f(\bar{x}, t^*)$$

$$k_2^* = f(\bar{x} + \tfrac{3}{4}k_1 h^*, t^* + \tfrac{3}{4}h^*)$$

$$x_{i+1} = \bar{x} + (\tfrac{1}{3}k_1^* + \tfrac{2}{3}k_2^*)h^*. \tag{16}$$

After this "modified step", we return to the conventional formulae (14), until a new zone transition happens.

It is obviously possible to modify any other explicit integration method in a similar way.

4. IMPLEMENTATION

In this section the actual implementation issues of the PSIMCNN simulator are discussed. A first version of PSIMCNN developed for the simulation of the Chua and Yang CNN model was called PSIMCNN 1.0 [18]. Now an improved version called PSIMCNN 1.1 has been developed. In this paper this new version is described.

The simulator has been studied for a parallel computing architecture based on an InMOS T800 Transputer network, adopting the GJ-WR method.

As shown in paragraph 2.2, the GJ-WR algorithm basically produced a sequence of waveform vectors $\mathbf{x}^{(k)} = [x_1^{(k)} x_2^{(k)} \ldots x_n^{(k)}]^T$. Any solved equation is a first-order ODE in a single unknown (see the algorithm in paragraph 2.2). Moreover, any waveform of this vector is obtained independently from each other so they can be generated in parallel by different processors.

This form of parallelism is known as the *multiple barrier approach* [12,14]. When the set of n waveforms has been obtained, the processors will use the computer waveforms to perform the new relaxation iteration. These iterations will continue until convergence is reached in the sense explained in paragraph 2.2.

4.1. Hardware

For the sake of clarity, some basic characteristics of transputers should be recalled. They are general-purpose processors that were created for parallel processing. Any transputer has its own private RAM to store programs and data, and four communication channels that can be used concurrently. Therefore it can be considered as a small independent computer. For our implementation InMOS T800 transputers were used.

As already states the ODEs (or from another point of view, it is possible to talk about cells instead of equations) have to be distributed among the available processors. At the end of any iteration, in order to perform the next one, the generic processor P that simulates the cell $C(i, j)$, will need all the waveforms corresponding to the variables that appear in the cell state equation, i.e. its neighbour cells. So, if some of these cells are simulated by a different processor, then processor P will need to receive these waveforms.

Communications among processors take a certain time that should, obviously, be minimized to improve the efficiency of the simulator. Therefore, in order to decide what cells/equations have to be assigned to a particular processor, the following criterion should be adopted:

(1) if a cell is assigned to P, its neighbour cells, if possible, have to be assigned to P as well;
(2) if some neighbour cell $C1$ cannot be assigned to P, a communication between P and the processor that simulates $C1$ will be necessary, so it is better to assign $C1$ to a processor that can directly communicate with P.

With this basis, a good choice is to use a hardware architecture that mirrors the CNN architecture. The transputer network shown in Fig. 4 has been adopted; it consists of a pipeline of T transputers (InMOS T800 with 20 MHz clock and 1 MB RAM each). The first one, called P0, can communication with the user by a simple PC (the host).

A PC 486 was used but many other host types can be used as well because it is just used as a terminal (no computing features were used). The actual simulation is performed by the transputers.

The four communication channels (marked 0–3) have been configured in such a way that any transputer can exchange information with its two neighbouring processors.

With this architecture it is very simple to add a new processor (in order to improve the computing capabilities). An alternative array architecture has been considered in Ref. [18] but the high flexibility of the architecture of Fig. 4 leads us to reject other alternatives.

4.2. Partitioning, scheduling and communications

Bearing in mind the above criterion about assignment of cells to the available processors it has been chosen to divide the CNN "by columns" and to assign an equal number of columns to any processor (see Fig. 5). If the number of columns (N) is not exactly divisible by the number of processors (T), there will be some processors loaded more than the others. More precisely, it will be assigned:

$$(N \text{ DIV } T) + 1 \text{ columns to the } (N \text{ REM } T) \text{ first processors, and}$$

$$(N \text{ DIV } T) \text{ columns to the remaining processors.}$$

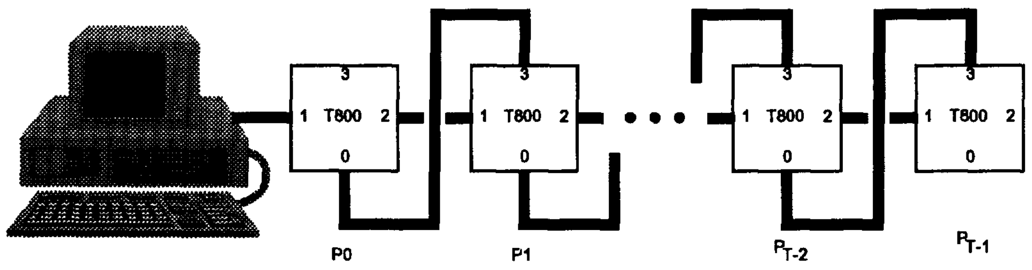


Fig. 4. The transputer architecture used. T = number of transputers.

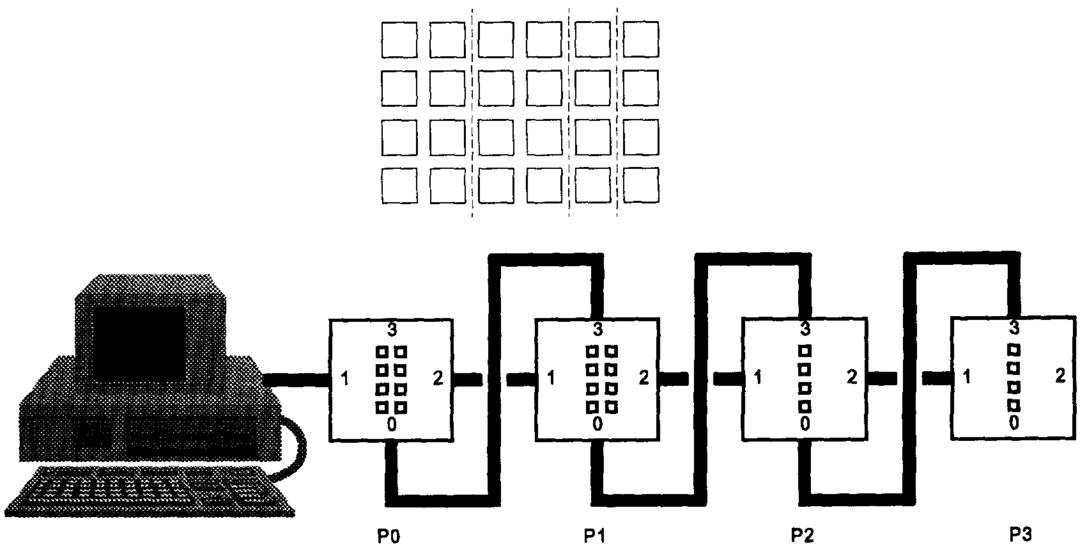


Fig. 5. Partition of a 4 × 6 CNN on a 4-transputer network.

Obviously the best computing load distribution is obtained when all the processors have an equal number of columns to simulate. It is clear as well that, as the CNN dimensions grow (more exactly as N grows) then possible differences in the computing load are less important. Suppose, for example, that the transputer network is composed by four transputers. If a CNN with six columns is simulated then there are two processors with two columns and two processors with one column. If a CNN with 38 columns is simulated then there are two processors with 10 columns and two processors with nine columns. It is obvious that in the latter case the computing load distribution is better although the first two processors have a bigger number of columns.

Now consider the generic processor P with the generic cell $C(i, j)$. When the ODE of $C(i, j)$ is solved, P needs the waveforms of the neighbouring cells. With this placement, these cells are assigned to P itself or to one of the two processors linked with P, so communications are minimized. Moreover, any processor communicates with a neighbouring one with two concurrent channels, therefore communication is even faster.

So at the end of any relaxation sweep, any processor will send some waves to its two neighbours and will receive some other waves from these two processors. Figure 6, for example, shows the case in which a CNN with a neighbour radius $r = 1$ is simulated. If the processors P2 is considered, it needs to receive the waves of the cells marked in black in Fig. 6(a) from the two processors P1 and P3. Analogously, it must send to P1 the marked cells to the left of Fig. 6(b) and it must send to P3 the marked cells shown to the right.

This is accomplished with the method shown in Fig. 7; i.e. concurrently, any transputer will use:

(1) channel 0 to send the "right-hand side columns" to the "right-hand side processor";
(2) channel 1 to send the "left-hand side columns" to the "left-hand side processor";
(3) channel 2 to receive the "right-hand side columns" from the "right-hand side processor";
(4) channel 3 to receive the "left-hand side columns" from the "left-hand side processor".

It is interesting that if the processor computing load is well distributed, although the processors work in an asynchronous way, they will finish their relaxation sweeps in almost the same time, so they will exchange their waves almost all together at the same instant.

Another interesting issue is that with this topology, the communication time becomes slower than the integration time as the CNN dimensions grow.

The previous issues herald high speed-up and parallelism efficiency. Experimental confirmation will be given in the following section.

As we saw, during the actual simulation (the integration of the ODE system), all the available processors do the same task. This means that it is possible to design a unique general program that

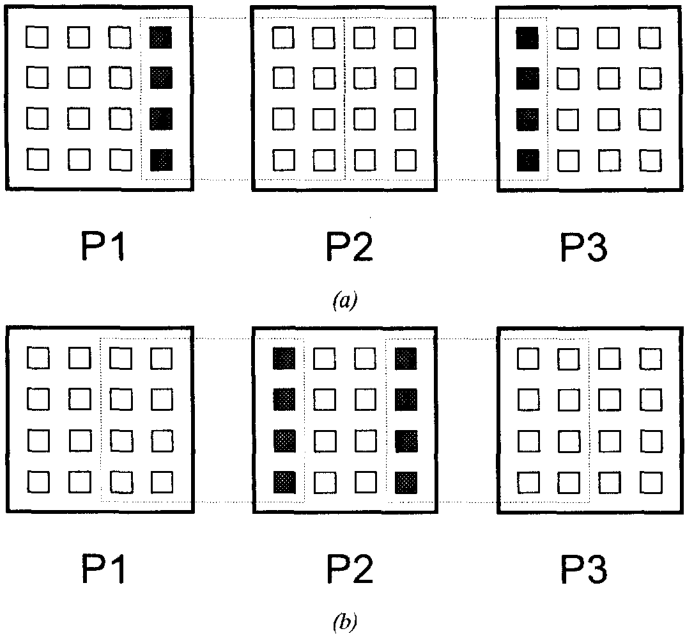


Fig. 6. (a) Black cells must be received by P2; (b) black cells must be sent by P2.

will run on any processor in the network. So implementation on an arbitrary number of processor is straightforward.

4.3. *Algorithm implementation: practical issues*

4.3.1. The memory problem: sections. As discussed in paragraph 2.2, in any relaxation iteration, we must have enough memory to store the entire waves of the previous and current relaxation sweeps. This is not a trivial matter. Let us consider, in fact, the following points:

(1) a lot of samples may be needed for the waves;
(2) any sample is represented with a double-precision 64-bit floating point number.

Obviously any processor will store just the waves it uses, so the needed memory is distributed among the available processors. However, this cannot be the final solution.

This problem has been solved in the following way. The samples are divided into groups called

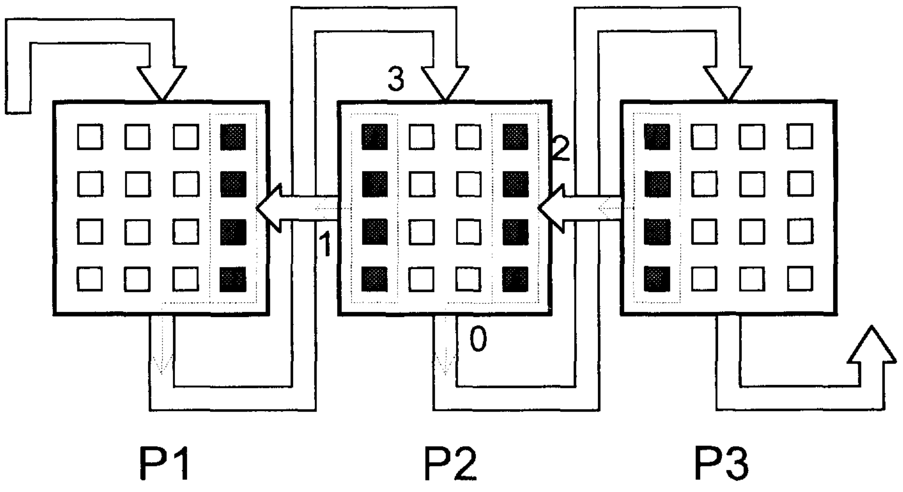


Fig. 7. Concurrent communication between adjacent transputers.

sections. The number of successive samples in any section depends on the quantity of memory available on any processor.

The simulator will integrate the system in a section; when the section is completed the requested waves are discarded to the hard disk of the host (so the memory can be re-used); the final state of the CNN is assumed as an initial state and the simulation is continued. This does not imply a relevant increase of time because when the processors have finished the communication of their waveforms to the host, they are free to continue the simulation, while the host saves the data to the hard-disk.

4.3.2. Accelerating convergence: windows and partial convergence. In order to accelerate convergence several authors have proposed a technique based on *time windows* [12–14]. It consists of the following concept. The simulated time interval $[t_i, t_f]$ (corresponding to a section) can be divided into m sub-intervals called *windows* $[T_0, T_1]$, $[T_1, T_2]$, . . . $[T_{m-1}, T_m]$ (with $t_i = T_0$ and $t_f = T_m$). The relaxation algorithm will not apply over the entire interval $[t_i, t_f]$ but on the single windows. It will begin on $[T_0, T_1]$; eventually the final values $\mathbf{x}(T_1)$ will be used as initial values for the successive window $[T_1, T_2]$ and so on. It has been theoretically proved that this approach can increase the convergence speed [14]. Moreover, convergence speed depends on the window dimensions and the system dynamics, so choosing the most useful window dimension is not easy.

The best thing is to determine it, for our particular case, is by rule of thumb. Obviously when a window is so small that it contains just one sample per wave, a classical direct algorithm is obtained and the advantages of the WR method are lost. On the other hand, it can be proved [14] that the relaxation in a window containing the entire simulation interval can be slower than using $m > 1$ windows. So a trade-off should be found.

In summary, samples are divided into successive sections and each section is simulated dividing it into windows.

There is another topic that can help us to improve the simulator efficiency. If the WR algorithm is applied to very large systems, it is often the case that some state waveforms will converge much more rapidly than others. This phenomenon is called *partial waveform convergence*, and can be exploited [14]. It is in fact clear that if a particular wave converges before the others, we can avoid re-computing it in the next relaxation sweep, with a gain in terms of simulation time. Therefore, as the relaxation sweep proceeds, the number of computed waveforms is reduced. It is clear, nevertheless, that if some processors have slow-converging waves, as relaxation proceeds, they will work more than the others. In this case the parallelism efficiency is reduced but the overall speed is improved with respect to the case in which partial convergence is not exploited.

4.3.3. Exploiting dynamic behaviour. As we saw above, our problem has been reduced to the integration of single unknown first-order non-linear ODEs. In the first version of PSIMCNN [18] this was obtained by fixed step-size explicit integration algorithms. In PSIMCNN version 1.1 this fundamental step in the simulation has been improved by means of *adaptive step-size methods* [17–19]. In particular, the user can choose between four integration methods: explicit Euler, modified Ralston, fourth-order Runge–Kutta, implicit Euler. In any one of these, the step-size is dynamically determined by an adaptive step-size method.

As anyone interested in numerical methods knows, the adaptive step-size technique consists of dynamically changing the integration step-size h during the integration process, trying to use the widest possible step-size permitted by the desired local accuracy. This implies that the step-size will be reduced when the integrated variable has a "fast dynamic" whilst it will be enlarged when the variable has a "slow dynamic" (Fig. 8).

This technique implies an additional computing load due to the problem of dynamically estimating the local integration error and resizing the step-size h. However, this is plentifully compensated by a lot of advantages. The first one is that variables with a "slow dynamic" are faster simulated (whilst with a fixed step-size method the smallest step-size for the entire set of variables must be used) because just a few "wide" steps are necessary. The second one is that we have a better guarantee that the desired local accuracy is respected.

However, after the wave has been obtained, it must be compared with its previous iteration estimation. Obviously, in general, the samples describing these two waves will not be the same. In particular they will differ in the number of samples, the corresponding simulated time instants and, obviously, in their value (Fig. 8). So, in order to compare these two waves a unique representation

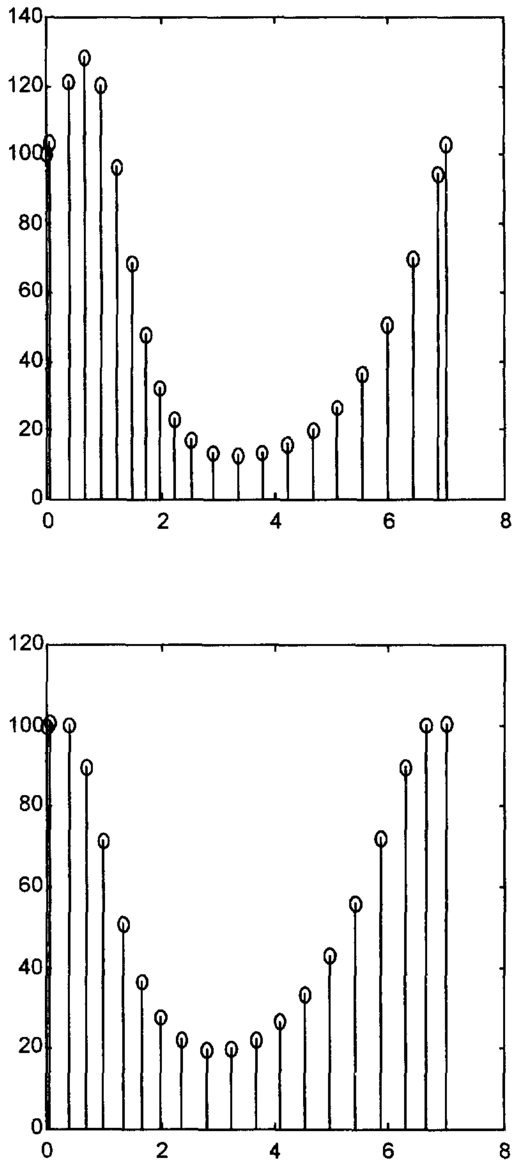


Fig. 8. Two successive relaxation sweeps of a waveform. Note the adaptive step-size sampling and the different number and value of samples.

form must be used. The simplest solution is to re-sample the computed waves with a fixed sampling time.

The drawback of this approach are that:

(1) the re-sampling process increases the processing time;
(2) if the fixed step-size is too wide, information about the wave dynamic is lost.

However, these two disadvantages are not so serious. Re-sampling is, in fact, very fast as compared to integration time. Moreover, the second problem can be solved with an opportune step-size: it is clear that if good local accuracy is desired we cannot require just a few samples.

There is an alternative to the re-sampling approach. It consists of storing the computed samples and the corresponding instants as well. However, this solution was rejected because:

(1) it increases the needed memory; and, as we said before, one of the disadvantages of WR methods is the large memory required, so it is not suitable to worsen this situation;

(2) the comparison procedure becomes highly complicated and time-consuming; let us recall that this kind of comparison is very often executed during the simulation process.

So the re-sampling approach was chosen.

If the value of a waveform is needed at an instant that is intermediate between two available samples an estimate of this wave value must be chosen. In these cases, some authors choose the nearest previous sample value, while, in our case, the desired value has been obtained by linear interpolation between the two available samples.

The adaptive step-six approach reveals its advantages in terms of computing time in many CNN simulations because, in a significant number of cases, just a few cells are really "active" at the same time, so the other relatively " inactive" variables can be integrated faster. In Ref. [4], for example just the cells involved in the transition from P_- to P_+ are really active (whilst the others are always in the proximity of an equilibrium point or completely at rest).

On the other hand, if a processor has a lot of inactive cells it is under-exploited as compared with another one that has more active cells. This implies a waste of resources and thus lower speed-up and parallelism efficiency. However, as we just said, there are some reasons to prefer this method to the fixed step-size.

4.4. *Overall view of the simulator*

Now the main implementation issues have been examined and an overall view of the simulator can be obtained looking at the flow-chart in Fig. 9.

The differences between versions 1.0 and 1.1 of PSIMCNN are reported in Fig. 17.

5. EXPERIMENTAL RESULTS

In this section the experimental results obtained with PSIMCNN 1.0 and PSIMCNN 1.1 are reported. Moreover, a comparison with the popular CNN simulator CNN–HAC is done.

In particular some simulations of Chua and Yang CNNs executed with PSIMCNN 1.0 and of Chua's circuit CNNs executed with PSIMCNN 1.1 are reported. This choice is dictated by a simple reason: in Ref. [20] it is shown that acceptable simulations of some simple Chua and Yang CNNs can be obtained using fixed step-six algorithms if the simulated CNN converges to its final state in just a few simulations steps. This is not the case of a CNN built with Chua's circuits. In fact, this system, because of its dynamic, needs to be simulated with more accuracy and for longer simulated intervals. So it absolutely requires PSIMCNN 1.1.

5.1. *Parallelism*

The parallelism of our implementation [12,21] it is now considered; in particular, the speed-up and efficiency are evaluated.

Speed-up and efficiency are defined as:

$$S_P = \frac{\tau_1}{\tau_P} \quad \text{(speed-up)} \quad E = \frac{S_P}{T} \quad \text{(efficiency)} \tag{17}$$

where τ_1 is the running time for a single processor implementation, τ_P is the running time for a parallel implementation, and T is the number of processors. An important (not desired but unavoidable) contribution to τ_P is the communication time.

Three configurations were considered: with one, four and six transputers. The actual simulation time was evaluated using the internal timers of the transputers.

5.1.1. Chua and Yang CNN simulations. Let us consider some Chua and Yang CNN simulations executed with PSIMCNN 1.0.

The parameters of the reported simulations are:

$t_i = 0$ s $t_f = 5$ μs (initial and final simulated time)

$R_x = 1$ kΩ $C = 1$ nF (cell parameters)

$\varepsilon = 0.001$ (i.e. global relative accuracy: 0.1%)

5 windows 150 samples for any waveform

Explicit integration with modified Ralston.

Because simulation time depends on the templates and on the initial values, two different sets of templates were used.

The first set is called AVERAGE and consists of:

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 0 \end{bmatrix} \cdot 10^{-3}\,\Omega^{-1} \quad B = 0\,\Omega^{-1} \quad I = 0A. \tag{18}$$

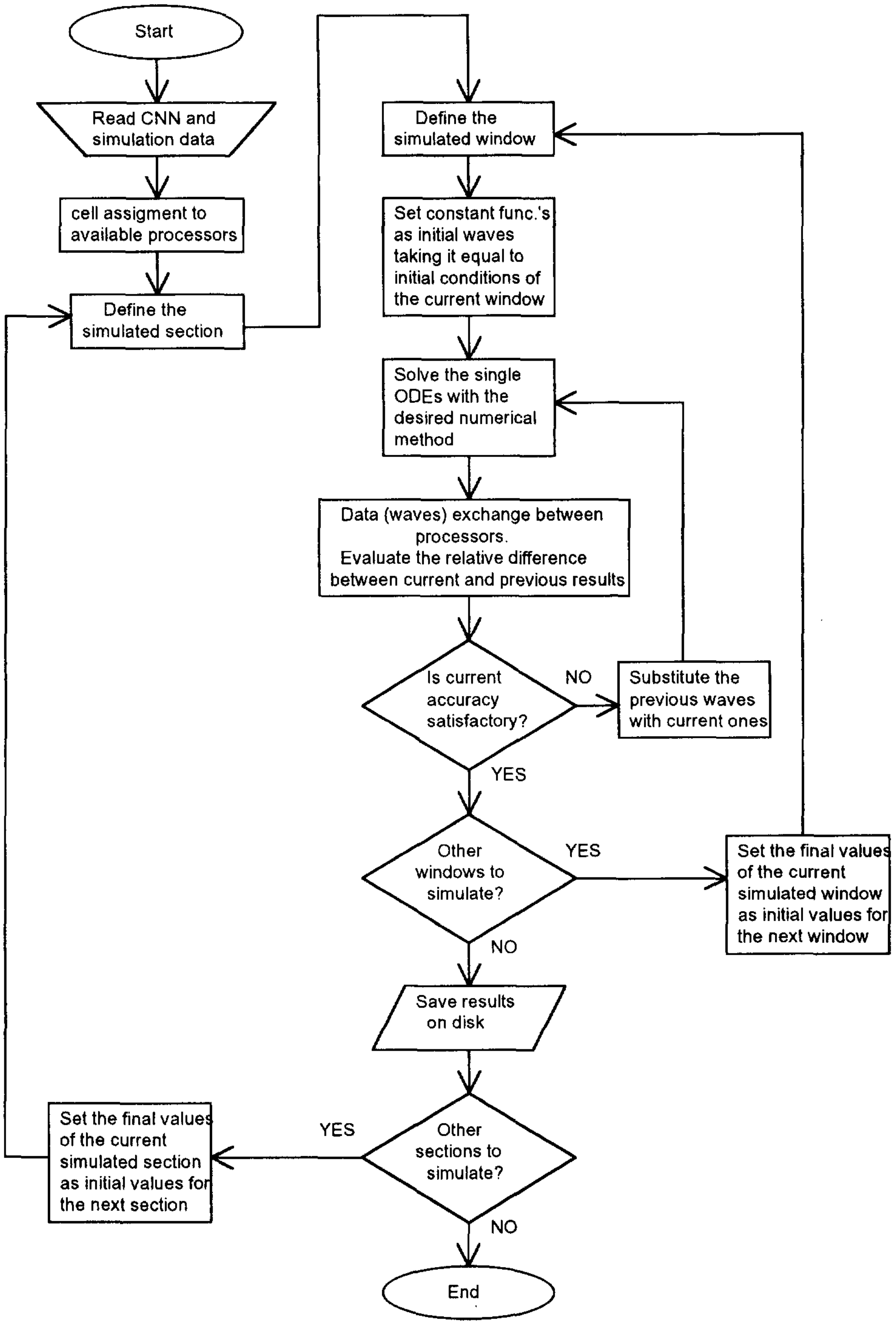

Start
Read CNN and simulation data
cell assigment to available processors
Define the simulated section
Define the simulated window
Set constant func.'s as initial waves taking it equal to initial conditions of the current window
Solve the single ODEs with the desired numerical method
Data (waves) exchange between processors. Evaluate the relative difference between current and previous results
Is current accuracy satisfactory?
NO
Substitute the previous waves with current ones
YES
Other windows to simulate?
YES
Set the final values of the current simulated window as initial values for the next window
NO
Save results on disk
Other sections to simulate?
YES
Set the final values of the current simulated section as initial values for the next section
NO
End

Fig. 9. The overall simulator flow-chart.

CNN			Running time (ms)			Speed-up		Efficiency	
Rows	Columns	Area	T=1	T=4	T=6	T=4	T=6	T=4	T=6
6	6	36	9119	3625	2399	2,515586	3,801167	0,628897	0,633528
8	8	64	15999	4736	4736	3,378167	3,378167	0,844542	0,563028
10	10	100	26631	9214	6249	2,890276	4,261642	0,722569	0,710274
12	12	144	44013	12611	10785	3,490048	4,080946	0,872512	0,680158
14	14	196	74600	24054	18193	3,101355	4,100478	0,775339	0,683413
16	16	256	70612	19934	15063	3,54229	4,687778	0,885572	0,781296
18	18	324	105442	32771	25131	3,21754	4,195695	0,804385	0,699282
20	20	400	174143	48625	38998	3,581347	4,465434	0,895337	0,744239
22	22	484	240693	72856	48978	3,303681	4,914308	0,82592	0,819051
24	24	576	315107	87423	73735	3,604395	4,273506	0,901099	0,712251
26	26	676	444967	132384	96564	3,361184	4,608001	0,840296	0,768
28	28	784	564499	156540	113826	3,606101	4,959315	0,901525	0,826553

Fig. 10. Simulation results for the set AVERAGE.

The second one is called BORDER and is:

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} \cdot 10^{-3}\,\Omega^{-1}$$

$$B = \begin{bmatrix} -0.25 & -0.25 & -0.25 \\ -0.25 & 2 & -0.25 \\ -0.25 & -0.25 & -0.25 \end{bmatrix} \cdot 10^{-3}\,\Omega^{-1} \quad I = -1.5\,\mathrm{mA}. \tag{19}$$

So several CNNs with increasing dimensions ($6 \times 6, 8 \times 8, \ldots, 28 \times 28$) have been simulated with the set AVERAGE taking random numbers (between -1 and 1) as initial values ($B = 0$ so the inputs are not important). The simulation results are given in Figs 10–12.

In a similar way, using the set BORDER several CNNs have been simulated, with initial conditions equal to the inputs, each one equal to 1, except for the border cells, that are equal to -1. The corresponding results are given in Figs 13–15.

5.1.2. Chua's circuit CNN simulations. Now let us consider some Chua's circuit CNN simulations with PSIMCNN 1.0. The simulations of the autowave phenomenon discussed in Ref. [4] has been presented. The parameters of the simulations are:
CNN cell parameters:

$$\alpha = 9 \quad \beta = 30 \quad m_0 = -1/7 \quad m_1 = 2/7 \quad \varepsilon = -1/14 \quad D = 0.65.$$

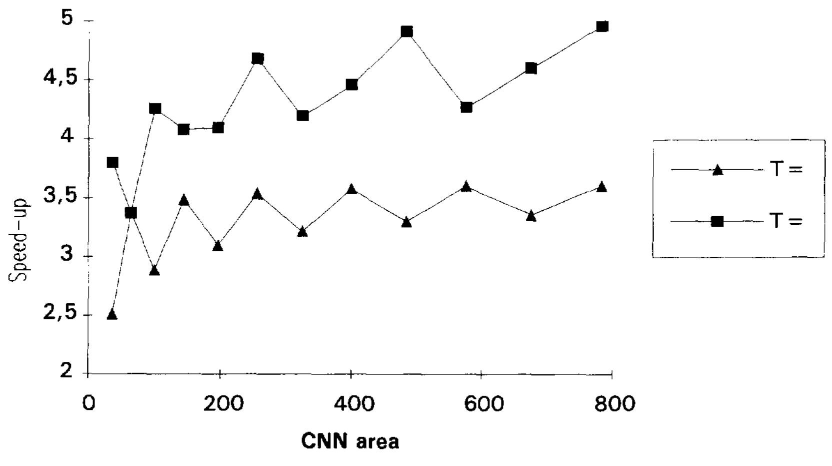


Fig. 11. Speed-up vs CNN area for the set AVERAGE.

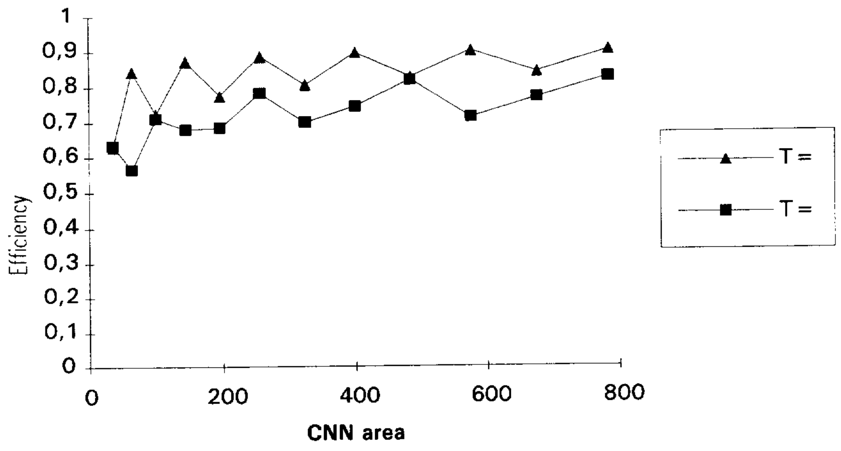


Fig. 12. Efficiency vs CNN area for the set AVERAGE.

Simulation parameters:

$t_i = 0$ s $t_i = 30$ s (simulated interval)

6000 samples for each waveform, divided into 60 sections of 100 samples

5 windows for any section

$\varepsilon_g = 0.001$ (global accuracy 0.1%)

$\varepsilon_l = 0.0005$ (local accuracy 0.05%)

Adaptive step-size fourth-order Runge–Kutta.

7 waveforms requested on disk.

5.2. *Results analysis*

In Figs 10–15 the experimental results concerning parallelism are shown. The CNN area is obviously the row × columns product.

Let us look at Figs 11 and 14. It can be seen that speed-up is high, even with a relatively small CNN and can be noted that it grows globally with the CNN area. This can be explained because the actual processing time grows faster than the communication time. Moreover, some small "oscillations" are superimposed. This is due to the fact that the number of CNN columns is not always exactly divisible by T (see also paragraph 4.2).

Let us now look at Figs 12 and 15. These results confirm the high degree of parallelism of the implementation.

It seems that the four-transputer implementation is more efficient than the six-transputer one.

CNN			Running time (ms)			Speed-up		Efficiency	
Rows	Columns	Area	1 TP	4 TPS	6 TPS	4 TPS	6 TPS	4 TPS	6 TPS
6	6	36	4066	1621	1077	2,508328	3,775302	0,627082	0,629217
8	8	64	7405	2213	2210	3,346136	3,350679	0,836534	0,558446
10	10	100	11823	4150	2803	2,848916	4,217981	0,712229	0,702997
12	12	144	17278	5005	4287	3,452148	4,030324	0,863037	0,671721
14	14	196	31304	8919	6747	3,509811	4,639692	0,877453	0,773282
16	16	256	39836	12510	9623	3,184333	4,139665	0,796083	0,689944
18	18	324	49270	13955	11209	3,530634	4,395575	0,882659	0,732596
20	20	400	59705	18381	12376	3,248191	4,824257	0,812048	0,804043
22	22	484	71141	20095	16668	3,540234	4,268119	0,885058	0,711353
24	24	576	83579	25374	18232	3,293884	4,584193	0,823471	0,764032
26	26	676	97018	27373	19679	3,544295	4,930027	0,886074	0,821671
28	28	784	111460	33487	25825	3,328456	4,315973	0,832114	0,719329
30	30	900	126903	35773	27587	3,547452	4,600101	0,886863	0,766684

Fig. 13. Simulation results for the set BORDER.

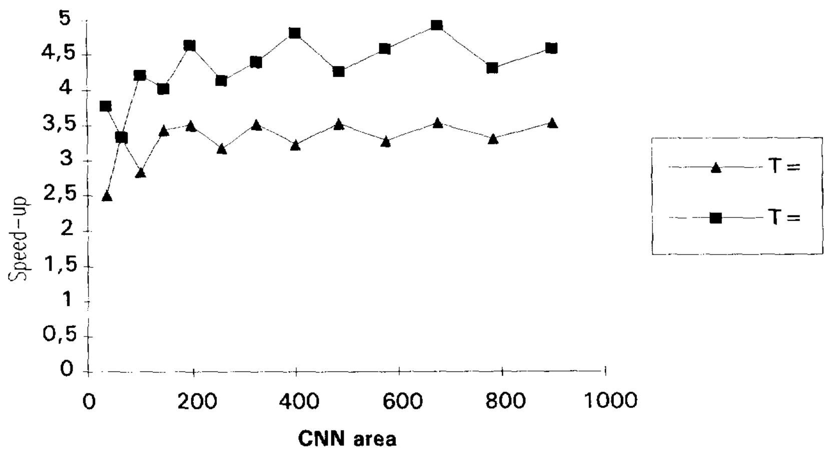


Fig. 14. Speed-up vs CNN area for the set BORDER.

This is just an illusion because if a coherent comparison has to be done then the efficiency with the same number of cells per processor should be considered.

For example, let us look at the table in Fig. 13. Let us consider the case of a 20 × 20 CNN with four transputers. In this case there are 100 cells per processor (5 columns × 20 rows). Now let us look at the case of a 26 × 26 CNN with six transputers; in this one there are 112.6667 cells per processor (an average value). The corresponding efficiencies are 0.812048 and 0.821671, so they are almost equal.

Now let us consider the results concerning Chua's circuit CNN simulation with PSIMCNN 1.1. shown in Fig. 16.

For these simulations the actual integration time and the total simulation time were evaluated. The former is the time spent just on actual simulation whilst the latter comprises the time spent writing the requested waveforms on the hard disk. It can be observed that the time spent on hard-disk operations is always between 2 and 5% of the total computing time. This confirms the statements of paragraph 4.3.1.

The speed-up is around 3.625 with six transputers (60% parallelism efficiency). As anticipated in paragraphs 4.3.2 and 4.3.3, it is lower than the fixed step-size/no partial convergence case. However, as it has been said in these paragraphs, there are a lot of good reasons for preferring these characteristics. Let us recall, once again, that other kinds of simulations may be unreliable with Chua's circuit arrays. Moreover, it is worth noting that an 8 × 15 Chua's circuit CNN is a non-linear dynamical system represented by $3 \times 8 \times 15 = 360$ state equations that have been simulated for 6000 time instants with a very high accuracy.

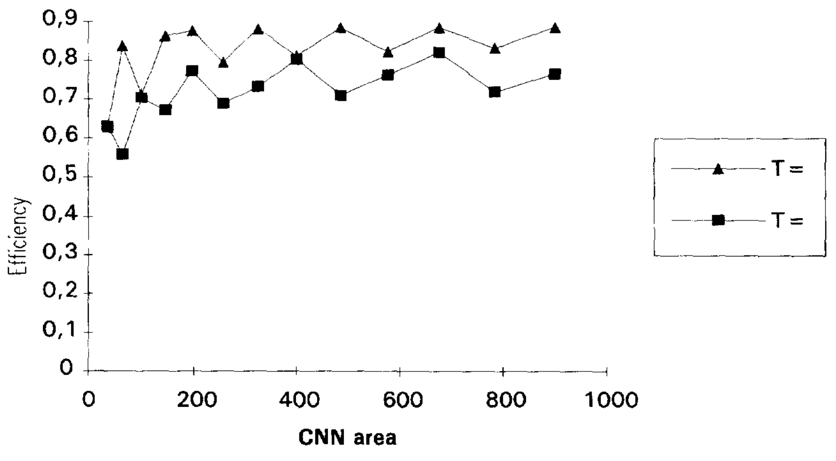


Fig. 15. Efficiency vs CNN area for the set BORDER.

CNN dimensions	Integration time with 1 TP	Total time with 1 TP	Integration time with 6 TPS	Total time with 6 TPS
2x15	1819 s (almost 30 min)	1856 s	490 s (almost 8 min)	512 s
2x12	1508 s (almost 25 min)	1536 s	407 s (almost 7 min)	424 s
8x15			1966 s (almost 33 min)	2048 s

Fig. 16. Simulation results for a Chua's circuit CNN with a propagating autowave.

5.3. The CNN–HAC simulator

The CNN–HAC is a CNN simulator that needs a hardware accelerator card [11]. This is a PC add-on board with four Texas TMS320C25/26 digital signal processors with 40 MHz clock frequency. The main points of this simulator are the following:

(1) The time-discretization of equation (2) with the explicit Euler method; so that (2) is transformed into the state transition equation:

$$x_{ij}(k+1) = (1-h)\cdot x_{ij}(k) + \mathrm{h}\cdot \sum_{C(k,l)\in N_r(i,j)} A(i,j;k,l)y_{kl}(k) + h\cdot \sum_{C(k,l)\in N_r(i,j)} B(i,j;k,l)u_{kl} + h\cdot I \tag{20}$$

in which the operations are done in fixed-point math. This equation can be thought of as the description of the operations done by a virtual digital processor. This simulator can simulate CNN models of types (1) and (2) only.

(2) All these virtual digital processors are mapped onto a realisable set of physical processors [the four DSPs that can perform the multiplications and sums on the right-hand side of equation (20)].

(3) Algorithm (20) is iterated by the physical processors, exchanging the information about the common data (cell mapping is similar to our simulator) before any time-step.

(4) The algorithm is implemented using the TMS320C2x Assembly Language.

5.4. A comparison with CNN–HAC

In this paragraph a comparison between PSIMCNN 1.0 and CNN–HAC is made. It has to be recalled that CNN–HAC cannot simulate the Chua's circuit array (3).

First of all, two different CNNs on the two simulators will be simulated. In this comparison a four-transputer implementation for PSIMCNN is used. The parameters of the first reported simulation are:

$t_i = 0$ s $t_f = 20$ s (initial and final simulated time)
$Rx = 1\,\Omega$ $C = 1$ F (cell parameters)
$\varepsilon = 0.001$ (i.e. global relative accuracy: 0.1%)
5 windows 50 samples for any waveform
Explicit integration with Euler method

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} \Omega^{-1} \quad B = \begin{bmatrix} -0.25 & -0.25 & -0.25 \\ -0.25 & 2 & -0.25 \\ -0.25 & -0.25 & -0.25 \end{bmatrix} \Omega^{-1} \quad I = -1.5A. \tag{21}$$

With a 16×16 CNN using the set of initial condition/inputs known as CHINEESE which is basically a set of ± 1, PSIMCNN ran for 1156 ms, whilst CNN–HAC spent 35 s.

The parameters of the second simulation are:

$t_i = 0$ s $t_f = 30$ s (initial and final simulated time)
$Rx = 1\,\Omega$ $C = 1$ F (cell parameters)
$\varepsilon = 0.001$ (i.e. global relative accuracy: 0.1%)
7 windows 70 samples for any waveform
Explicit integration with Euler method

	PSIMCNN 1.0	**PSIMCNN 1.1**	**CNN-HAC**
Hardware	InMOS T800 Transputer network (general purpose)	InMOS T800 Transputer network (general purpose)	CNNHAC dedicated accelerator board
Software	OCCAM program (almost 1500 lines for P0+ 800 lines for any other transputer)	OCCAM program (almost 2000 lines for P0+ 1300 lines for any other transputer)	Assembly program
Numerical methods	Gauss-Jacobi Waveform Relaxation + 3 Fixed step-size methods: Explicit Euler / Modified Ralston / Fourth-order R-K	Gauss-Jacobi Waveform Relaxation + 4 Adaptive step-size methods: Explicit Euler / Modified Ralston / Fourth-order R-K / Implicit Euler	Fixed step-size Explicit Euler Method
Accuracy	Global accuracy monitoring	Global and Local accuracy monitoring	Local accuracy fixed by step-size
	Local accuracy fixed by step-size		
Simulation results	pixels map representing the state/output of the cells + numerical values (on file) of any desired state variable	pixels map representing the state/output of the cells + numerical values (on file) of any desired state variable	pixels map representing the state/output of the cells
Flexibility	It is simple to modify the software to simulate new CNN models. Moreover it can be implemented on an arbitrary number of transputers	It is simple to modify the software to simulate new CNN models. Moreover it can be implemented on an arbitrary number of transputers	Only supported CNN models can be simulated
Memory	We must keep all the samples of the relaxed waveforms	Only section samples need be maintained	We need to keep just the current simulated time samples

Fig. 17. The features of PSIMCNN 1.0 and 1.1 vs CNN–HAC.

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 0 \end{bmatrix} \Omega^{-1} \quad B = 0\,\Omega^{-1} \quad I = 0A. \tag{22}$$

With a 44×25 CNN using the set of initial condition/inputs known as CONWAY that is basically another set of ± 1, PSIMCNN ran for 13 s, whilst CNN–HAC spent 55 s.

In summary, PSIMCNN can achieve a faster simulation speed than CNN–HAC.

There are many other very important issues to compare. They are reported in the table in Fig. 17.

6. CONCLUSIONS

In this paper a new CNN simulator for a general-purpose parallel computing architecture based on a T800 Transputer network has been presented. It adopts a Gauss–Jacobi waveform relaxation method and allows to choose between four ODE integration methods for the single equation integrations. The convergence of the WR method with the most common CNN models has been proved. A new integration method for a particular kind of non-linear ODE has been introduced. Implementation issues of the simulator have been discussed. Experimental results show that good parallelism efficiency can be obtained. Parallelism efficiency decreases when, with PSIMCNN 1.1, a CNN with just local "dynamic activity" is simulated. This is the case of the Chua's array with the autowave phenomenon. However, although efficiency is not comparable with other presented cases, some other advantages are obtained, e.g. higher accuracy (due to the dynamic local error control).

The simulator also allows us to introduce the Chua circuit as a cell in the network. CNN–HAC is not capable of simulating this array of circuits so a more flexible simulator is needed; moreover, this kind of simulation is very time-consuming [4]. The proposed solution can be an effective answer to this need.

The improvement with respect to classical simulators, mainly consists of the reconfigurability of the system, referring to the number of available parallel hardware units.

High speed computations levels could be reached for very high-order CNN circuits with a slight increase in the number of transputers.

A possible improvement on the presented simulator would be a dynamic partitioning of CNN in which just some cells have a real dynamic activity. In this way, better efficiency could be obtained. However, this issue should be seriously investigated because the implementation of a dynamic partitioner and scheduler may overload the system itself, leading to small improvements or worse.

REFERENCES

1. L. O. Chua and L. Yang, Cellular neural networks: theory. *IEEE Trans. Circuits Syst.* **35,** 1257–1272 (1988).
2. L. O. Chua and L. Yang, Cellular neural networks: applications. *IEEE Trans. Circuits Syst.* **35,** 1273–1290 (1988).
3. L. O. Chua and T. Roska, The CNN paradigm. *IEEE Trans. Circuits Syst.—I: Fund. Theory and Appl.* **40,** 147–156 (1993).
4. V. Pérez-Muñuzuri, V. Pérez-Villar and L. O. Chua, Autowaves for image processing on a two-dimensional CNN array of excitable nonlinear circuits: flat and wrinkled labyrinths. *IEEE Trans. Circuits Syst.—I: Fund. Theory and Appl.* **40,** 174–181 (1993).
5. S. Jankowski, C. Mazur and R. Wanczuk, Some problems of molecular physics solved by CNN. *Proc. 1993 Int. Symp. Nonlinear Theory and Its Application (NOLTA '93)*, pp. 17–22 (1993).
6. A. Pérez-Muñuzuri, V. Pérez-Muñuzuri, V. Pérez-Villar and L. O. Chua, Spiral waves on a 2-D array of nonlinear circuits. *IEEE Trans. Circuits Syst.—I: Fund. Theory and Appl.* **40,** 872–877 (1993).
7. T. Roska, J. Hámori, E. Lábos, K. Lotz, L. Orzó, J. Takács, P. L. Venetianer, Z. Vidnyánsky and A. Zarándy, The use of CNN models in the subcortical visual pathway. *IEEE Trans. Circuits Syst.—I: Fund. Theory and Appl.* **40,** 182–195 (1993).
8. L. O Chua and P. Thiran, An analytic method for designing simple cellular neural networks. *IEEE Trans. Circuits Syst.* **38,** 1332–1341 (1991).
9. T. Kozek, T. Roska and L. O. Chua, Genetic algorithm for CNN template learning. *IEEE Trans. Circuits Syst.—I: Fund. Theory and Appl.* **40,** 392–402 (1993).
10. J. A. Nossek, H. Magnussen, P. Nachbar and A. J. Schuler, Learning algorithms for cellular neural networks. *Proc. 1993 Int. Symp. Nonlinear Theory and Its Application (NOLTA '93)*, pp. 11–16 (1993).
11. T. Roska *et al.*, A digital multiprocessor hardware accelerator board for cellular neural networks: CNN–HAC. *Proc. IEEE Int. Workshop on Cellular Neural Networks and their Applications*, pp. 160–168 (1990).
12. R. A. Saleh, K. A. Gallivan, M. C. Chang, I. N. Hajj, D. Smart and T. N. Trick, Parallel circuit simulation on supercomputers. *Proc. IEEE* **77,** 1915–1931 (1989).
13. A. R. Newton and A. L. Sangiovanni-Vincentelli, Relaxation-based electrical simulation. *IEEE Trans. Electron Dev.* **30,** 1184–1207 (1983).
14. J. K. White and A. L. Sangiovanni-Vincentelli, *Relaxation Techniques for the Simulation of VLSI Circuits.* Kluwer, New York (1987).
15. R. J. Kaye and A. L. Sangiovanni-Vincentelli, Solution of piecewise-linear ordinary differential equation using waveform relaxation and laplace transforms. *IEEE Trans. Circuits Syst.* **30,** 353–357 (1983).
16. V. Pérez-Muñuzuri, V. Pérez-Villar and L. O. Chua, Travelling wave front and its failure in a one-dimensional array of Chua's circuits. *J. Circuits Syst. Comp.* **3,** 215–229 (1993).
17. S. C. Chapra and R. P. Canale, *Numerical Methods for Engineers with Personal Computer Applications.* McGraw–Hill, New York (1985).
18. L. Fortuna, G. Manganaro, G. Muscato and G. Nunnari, PSIMCNN: an innovative simulator for cellular neural networks. *Transputer Applications and Systems '94* (Edited by A. De Gloria *et al.*), pp. 302–317 (1994).
19. W. H. Press, B. P. Flannery, S. A. Tenkolsky and W. T. Vetterling, *Numerical Recipes: the Art of Scientific Computing.* Cambridge Univ. Press, Boston, Mass. (1987).
20. H. Harrer, A. Schuler and E. Amelunxen, Comparison of different numerical integration methods for simulating cellular neural networks. *Proc. IEEE Workshop on Cellular Neural Networks and their Applications*, pp. 151–159 (1990).
21. D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computations, Numerical Methods.* Prentice–Hall, New York (1989).
22. F. Zou and J. A. Nossek, Bifurcation and chaos in cellular neural networks. *IEEE Trans. Circuits Syst.—I: Fund. Theory and Appl.* **40,** 166–173 (1993).

APPENDIX

In this appendix the proofs of Theorems 3.1 and 3.2 are reported.

Proof of Theorem 3.1

The ODE system (11) can be written as:

$$\dot{x}_{ij}(t) = -a\cdot x_{ij}(t) + J + \sum_{C(k,l)\in N_r(i,j)} \hat{a}_{ij;kl}(y_{kl}(t), y_{ij}(t)) + \sum_{C(k,l)\in N_r(i,j)} \hat{b}_{ij;kl}(u_{kl}(t), u_{ij}(t))$$
$$+ \sum_{C(k,l)\in N_r(i,j)} a^{\tau}_{ij;kl} y_{kl}(t-\tau) + \sum_{C(k,l)\in N_r(i,j)} b^{\tau}_{ij;kl} u_{kl}(t-\tau) \tag{A1}$$

where:

$$a = \frac{1}{R_x C};\quad \hat{a}_{ij;kl}(\cdot,\cdot) = C^{-1}\hat{A}_{ij;kl}(\cdot,\cdot);\quad \hat{b}_{ij;kl}(\cdot,\cdot) = C^{-1}\hat{B}_{ij;kl}(\cdot,\cdot);\quad J = C^{-1}I;$$
$$a^{\tau}_{ij;kl} = C^{-1}A^{\tau}_{ij;kl};\quad b^{\tau}_{ij;kl} = C^{-1}B^{\tau}_{ij;kl}; \tag{A2}$$

substituting (11b):

$$\dot{x}_{ij}(t) = -a\cdot x_{ij}(t) + J + \sum_{C(k,l)\in N_r(i,j)} \hat{a}_{ij;kl}(g(x_{kl}(t)), g(x_{ij}(t))) + \sum_{C(k,l)\in N_r(i,j)} \hat{b}_{ij;kl}(u_{kl}(t), u_{ij}(t))$$
$$+ \sum_{C(k,l)\in N_r(i,j)} a^{\tau}_{ij;kl} g(x_{kl}(t-\tau)) + \sum_{C(k,l)\in N_r(i,j)} b^{\tau}_{ij;kl} u_{kl}(t-\tau). \tag{A3}$$

For the sake of convenience, we introduce the well-known *lexicographical order* [22]:

$$s = (i-1)\cdot N + j \quad 1 \leqslant s \leqslant n \quad n = M\cdot N \tag{A4}$$

so we can re-write the system, numbering equations and variables with a single index as:

$$\dot{x}_s(t) = -a\cdot x_s(t) + J + \sum_{C(d)\in N_r(s)} \hat{a}_{s;d}(g(x_d(t)), g(x_s(t))) + \sum_{C(d)\in N_r(s)} \hat{b}_{s;d}(u_d(t), u_s(t))$$
$$+ \sum_{C(d)\in N_r(s)} a^{\tau}_{s;d} g(x_d(t-\tau)) + \sum_{C(d)\in N_r(s)} b^{\tau}_{s;d} u_d(t-\tau). \quad \begin{matrix} s \equiv (i,j) \\ d \equiv (k,l) \end{matrix} \tag{A5}$$

In this way it is clear that our system is form (5). In particular:

$$f_s(\mathbf{x}, t) = -a\cdot x_s(t) + J + \sum_{C(d)\in N_r(s)} \hat{a}_{s;d}(g(x_d(t)), g(x_s(t))) + \sum_{C(d)\in N_r(s)} \hat{b}_{s;d}(u_d(t), u_s(t))$$
$$+ \sum_{C(d)\in N_r(s)} a^{\tau}_{s;d} g(x_d(t-\tau)) + \sum_{C(d)\in N_r(s)} b^{\tau}_{s;d} u_d(t-\tau) \qquad \mathbf{x}\in\mathscr{R}^n \tag{A6}$$

and, defining $N_r'(s) = N_r(s) - \{C(s)\}$:

$$\tilde{f}_s(\mathbf{z}, \mathbf{w}, t) = -a\cdot z_s(t) + \hat{a}_{s;s}(g(z_s(t)), g(z_s(t))) + \sum_{C(d)\in N_r'(s)} \hat{a}_{s;d}(g(w_d(t)), g(z_s(t)))$$
$$+ \sum_{C(d)\in N_r(s)} \hat{b}_{s;d}(u_d(t), u_s(t)) + a^{\tau}_{s;s} g(z_s(t-\tau)) + \sum_{C(d)\in N_r'(s)} a^{\tau}_{s;d} g(w_d(t-\tau))$$
$$+ \sum_{C(d)\in N_r(s)} b^{\tau}_{s;d} u_d(t-\tau) + J \qquad \mathbf{z},\mathbf{w}\in\mathscr{R}^n \tag{A7}$$

so we have:

$$\|\tilde{\mathbf{f}}(\mathbf{z}',\mathbf{w}', t) - \tilde{\mathbf{f}}(\mathbf{z}'',\mathbf{w}'', t)\| = \max_{1\leqslant s\leqslant n} (\|\tilde{f}_s(\mathbf{z}',\mathbf{w}', t) - \tilde{f}_s(\mathbf{z}'',\mathbf{w}'', t)\|). \tag{A8}$$

In order to prove that $\tilde{\mathbf{f}}$ is uniformly Lipschitz (UL), consider the norm:

$$\|\tilde{f}_s(\mathbf{z}',\mathbf{w}', t) - \tilde{f}_s(\mathbf{z}'',\mathbf{w}'', t)\| = \| -a\cdot(z_s'(t) - z_s''(t)) + \{\hat{a}_{s;s}(g(z_s'(t)), g(z_s'(t))) - \hat{a}_{s;s}(g(z_s''(t)), g(z_s''(t)))\}$$
$$+ \sum_{C(d)\in N_r'(s)} \{\hat{a}_{s;d}(g(w_d'(t)), g(z_s'(t))) - \hat{a}_{s;d}(g(w_d''(t)), g(z_s''(t)))\}$$
$$+ \{a^{\tau}_{s;s}[g(z_s'(t-\tau)) - g(z_s''(t-\tau))]\} + \sum_{C(d)\in N_r'(s)} a^{\tau}_{s;d}[g(w_d'(t-\tau)) - g(w_d''(t-\tau))] \| \tag{A9}$$

then we have:

$$\|\tilde{f}_s(\mathbf{z}',\mathbf{w}', t) - \tilde{f}_s(\mathbf{z}'',\mathbf{w}'', t)\|$$
$$\leqslant |a|\cdot\|z_s'(t) - z_s''(t)\| + \|\hat{a}_{s;s}(g(z_s'(t)), g(z_s'(t))) - \hat{a}_{s;s}(g(z_s''(t)), g(z_s''(t)))\|$$
$$+ |a^{\tau}_{s;s}|\cdot\|g(z_s'(t-\tau)) - g(z_s''(t-\tau))\|$$

$$+\sum_{C(d)\in N'_r(s)} \|\hat{a}_{s;d}(g(w'_d(t)),g(z'_s(t)))-\hat{a}_{s;d}(g(w''_d(t)),g(z''_s(t)))\|$$

$$+\sum_{C(d)\in N'_r(s)} |a^{\tau}_{s;d}|\cdot\|g(w'_d(t-\tau))-g(w''_d(t-\tau))\|. \tag{A10}$$

Now let us look at the right hand part of this inequality. First of all, let us recall that we supposed that the non-linear functions $\hat{A}_{ij,kl}(\cdot,\cdot)$ $\hat{B}_{ij,kl}(\cdot,\cdot)$ are UL, therefore:

$\exists \hat{K}'_{s,d}$, $\hat{K}''_s>0$ so that

$$\|\hat{a}_{s;d}(g(w'_d(t)),g(z'_s(t)))-\hat{a}_{s;d}(g(w''_d(t)),g(z''_s(t)))\|\leqslant \hat{K}'_{s,d}\|g(w'_d(t))-g(w''_d(t))\|$$
$$+\hat{K}''_s\|g(z'_s(t))-g(z''_s(t))\|. \tag{A11}$$

Recalling that the set $N'_r(s)$ has $[(2\cdot r+1)^2-1]$ elements:

$$\sum_{C(d)\in N'_r(s)}\|\hat{a}_{s;d}(g(w'_d(t)),g(z'_s(t)))-\hat{a}_{s;d}(g(w''_d(t)),g(z''_s(t)))\|\leqslant \sum_{C(d)\in N'_r(s)}\hat{K}'_{s,d}\|g(w'_d(t))-g(w''_d(t))\|$$
$$+\hat{K}''_s\cdot[(2r+1)^2-1]\cdot\|g(z'_s(t))-g(z''_s(t))\|. \tag{A12}$$

So, taking $\hat{K}'_s=\max_{C(d)\in N'_r(s)}\hat{K}'_{s,d}$ and choosing $c\in N'_r(s)$ such that $\|g(w'_d(t))-g(w''_d(t))'_s$ we have:

$$\leqslant \hat{K}'_s\cdot[(2r+1)^2-1]\cdot\|g(w'_c(t))-g(w''_c(t))\|+\hat{K}''_s\cdot[(2r+1)^2-]1\cdot\|g(z'_s(t))-g(z''_s(t))\| \tag{A13}$$

but $g(x)$ is piece-wise linear, so applying lemma 3.2:

$$\leqslant \hat{K}'_s\cdot[(2r+1)^2-1][\cdot\{K_g\cdot\|w'_c(t)-w''_c(t)\|\}+\hat{K}''_s\cdot[(2r+1)^2-1]\cdot\{K_g[\text{msde}]\|z'_s(t)-z''_s(t)\|\}. \tag{A14}$$

So briefly, calling $\bar{K}'_s=\hat{K}'_s\cdot[(2r+1)^2-1]\cdot K_g$ and $\bar{K}''_s=\hat{K}''_s\cdot[(2r+1)^2-1]\cdot K_g$:

$$=\bar{K}'_s\cdot\|w'_c-w''_c\|+\bar{K}''_s\cdot\|z'_s-z''_s\|. \tag{A15}$$

Now consider the term $\|\hat{a}_{s;s}(g(z'_s(t)),g(z'_s(t)))-\hat{a}_{s;s}(g(z''_s(t)),g(z''_s(t)))\|$; analogously to the above procedure:

$\exists K'_s>0$ such that

$$\|\hat{a}_{s;s}(g(z'_s(t)),g(z'_s(t)))-\hat{a}_{s;s}(g(z''_s(t)),g(z''_s(t)))\|\leqslant K'_s\cdot\|z'_s-z''_s\|. \tag{A16}$$

The other terms contain linear functions of the state variables so they are particular cases of the above one. We can therefore write:

$\exists K^{\tau}_s>0$ such that

$$|a^{\tau}_{s;s}|\cdot\|g(z'_s(t-\tau))-g(z''_s(t-\tau))\|\leqslant K^{\tau}_s\cdot\|z'_s-z''_s\| \tag{A17}$$

$\exists K^{\tau w}_s>0$ such that

$$\sum_{C(d)\in N'_r(s)}|a^{\tau}_{s;d}|\cdot\|g(w'_d(t-\tau))-g(w''_d(t-\tau))\|\leqslant K^{\tau w}_s\cdot\|w'_e-w''_e\| \tag{A18}$$

where $e\in N'_r(s)$ is such that $\|g(w'_e)-g(w''_e)\|=\max_{C(d)\in N'_r(s)}\|g(w'_d)-g(w''_d)\|$. In summary we have: $\exists K'_s, K''_s, K'''_s>0$ such that:

$$|a|\cdot\|z'_s(t)-z''_s(t)\|+\|\hat{a}_{s;s}(g(z'_s(t)),g(z'_s(t)))-\hat{a}_{s;s}(g(z''_s(t)),g(z''_s(t)))\|$$
$$+|a^{\tau}_{s;s}|\cdot\|g(z'_s(t-\tau))-g(z''_s(t-\tau))\|$$
$$+\sum_{C(d)\in N'_r(s)}\|\hat{a}_{s;d}(g(w'_d(t)),g(z'_s(t)))-\hat{a}_{s;d}(g(w''_d(t)),g(z''_s(t)))\|$$
$$+\sum_{C(d)\in N'_r(s)}|a^{\tau}_{s;d}|\cdot\|g(w'_d(t-\tau))-g(w''_d(t-\tau))\|$$
$$\leqslant K'_s\cdot\|z'_s-z''_s\|+K''_s\cdot\|w'_c-w''_c\|+K'''_s\cdot\|w'_e-w''_e\|. \tag{A19}$$

From this relationship and (30) and (31) it follows that:

$$\|\bar{f}_s(\mathbf{z}',\mathbf{w}',t)-\bar{f}_s(\mathbf{z}'',\mathbf{w}'',t)\|\leqslant K'_s\cdot\|z'_s-z''_s\|+K''_s\cdot\|w'_c-w''_c\|+K'''_s\cdot\|w'_e-w''_e\|. \tag{A20}$$

Now taking $K_1=\max_s\{K'_s\}$ and $K_2=\max_s\{K''_s,K'''_s\}$ and considering the waveform vector norms:

$$\|\bar{\mathbf{f}}(\mathbf{z}',\mathbf{w}',t)-\bar{\mathbf{f}}(\mathbf{z}'',\mathbf{w}'',t)\|\leqslant K_1\|\mathbf{z}'-\mathbf{z}''\|+K_2\|\mathbf{w}'-\mathbf{w}''\|. \tag{A21}$$

This proves that $\bar{\mathbf{f}}$ is uniformly Lipschitz and from theorem 2.1, proves the thesis. ■

Proof of Theorem 3.2

Let us define the new state variables:

$$\xi_{i,j,s}=\begin{cases}x_{i,j}\text{ if } s=0\\ y_{i,j}\text{ if } s=1\\ z_{i,j}\text{ if } s=2\end{cases}\quad 0\leqslant s\leqslant 2. \tag{A22}$$

Moreover, let us consider an *extended lexicographical order*:

$$k = (i-1)\cdot N + j + s\cdot N\cdot M \Rightarrow 1 \leqslant k \leqslant L \text{ where } L = 3\cdot N\cdot M. \tag{A23}$$

With this index, let us introduce the other variables:

$$\lambda_k = \xi_{i,j,s} \tag{A24}$$

i.e.:

$$\lambda_k = \begin{cases} x_{i,j} \text{ if } 1 \leqslant k \leqslant N\cdot M \\ y_{i,j} \text{ if } N\cdot M + 1 \leqslant k \leqslant 2\cdot N\cdot M \\ z_{i,j} \text{ if } 2\cdot N\cdot M + 1 \leqslant k \leqslant L \end{cases} \tag{A25}$$

so the L equations of (12) are re-written as:

$$\begin{array}{lll} \dot{\lambda}_k = \alpha(\lambda_{(k+MN)} - h(\lambda_k)) + D[\lambda_{(k-N)} + \lambda_{(k+N)} + \lambda_{(k-1)} + \lambda_{(k+1)} - 4\lambda_k] & \text{if} & 1 \leqslant k \leqslant MN \\ \dot{\lambda}_k = \lambda_{(k-MN)} - \lambda_k + \lambda_{(k+MN)} & \text{if} & MN+1 \leqslant k \leqslant 2MN \\ \dot{\lambda}_k = -\beta\lambda_{(k-MN)} & \text{if} & 2MN+1 \leqslant k \leqslant L \end{array} \tag{A26}$$

Our system (A26) is now in the form $\dot{\lambda}_k = f_s(\lambda, t)$ where $\lambda \in \mathscr{R}^L$. We can also write in the form:

$$\tilde{f}_k(\eta, \lambda, t) = \begin{cases} \alpha(\lambda_{(k+MN)} - h(\eta)) + D[\lambda_{(k-N)} + \lambda_{(k+N)} + \lambda_{(k-1)} + \lambda_{(k+1)} - 4\eta_k] & \text{if} \quad 1 \leqslant k \leqslant MN \\ \lambda_{(k-MN)} - \eta_k + \lambda_{(k+MN)} & \text{if} \quad MN+1 \leqslant k \leqslant 2MN. \\ -\beta\lambda_{(k-MN)} & \text{if} \quad 2MN+1 \leqslant k \leqslant L \end{cases} \tag{A27}$$

In this proof an approach similar to the one used in the previous theorem will be followed; so let us consider the norm:

$$\|\tilde{\mathbf{f}}(\mathbf{z}', \mathbf{w}', t) - \tilde{\mathbf{f}}(\mathbf{z}'', \mathbf{w}'', t)\| = \max_{1 \leqslant k \leqslant L} (\|\tilde{f}_k(\mathbf{z}', \mathbf{w}', t) - \tilde{f}_k(\mathbf{z}'', \mathbf{w}'', t)\|). \tag{A28}$$

In order to prove that $\tilde{\mathbf{f}}$ is uniformly Lipschitz (UL), consider:

$$\|\tilde{f}_k(\mathbf{z}', \mathbf{w}', t) - \tilde{f}_k(\mathbf{z}'', \mathbf{w}'', t)\|. \tag{A29}$$

Let us begin by examining just the first MN equations (i.e. $1 \leqslant k \leqslant MN$):

$$\begin{aligned} \|\tilde{f}_k(\mathbf{z}', \mathbf{w}', t) - \tilde{f}_k(\mathbf{z}'', \mathbf{w}'', t)\| &\leqslant |\alpha|\cdot\|h(z'_k) - h(z''_k)\| + 4|D|\cdot\|z'_k - z''_k\| \\ &+ |\alpha|\cdot\|w'_{k+MN} - w''_{k+MN}\| + |D|\cdot[\|w'_{k-N} - w''_{k-N}\| + \|w'_{k+N} - w''_{k+N}\| \\ &+ \|w'_{k-1} - w''_{k-1}\| + \|_{k+1}' - w''_{k+1}\|]. \end{aligned} \tag{A30}$$

The non-linear function $h(\cdot)$ is a piece-wise linear function so it is UL. Therefore choosing $K'_k = 2\cdot\max(|\alpha m_0|, |\alpha m_1|, 4|D|)$:

$$\begin{aligned} &\leqslant K'_k\cdot\|z'_k - z''_k\| + |\alpha|\cdot\|w'_{k+MN} - w''_{k+MN}\| \\ &+ |D|\cdot[\|w'_{k-N} - w''_{k-N}\| + \|w'_{k+N} - w''_{k+N}\| \\ &+ \|w'_{k-N} - w''_{k-1}\| + \|w'_{k+1} - w''_{k+1}\|]. \end{aligned} \tag{A31}$$

Now let us consider the equations for $MN + 1 \leqslant k \leqslant 2MN$:

$$\begin{aligned} \|\tilde{f}_k(\mathbf{z}', \mathbf{w}', t) - \tilde{f}_k(\mathbf{z}'', \mathbf{w}'', t)\| &= \|(w'_{k-MN} - w''_{k-MN}) - (z'_k - z''_k) + (w'_{k+MN} - w''_{k+MN})\| \\ &\leqslant \|w'_{k-MN} - w''_{k-MN}\| + \|z'_k - z''_k\| + \|w'_{k+MN} - w''_{k+MN}\|. \end{aligned} \tag{A32}$$

Finally, let us consider the equations for $2MN + 1 \leqslant k \leqslant L$:

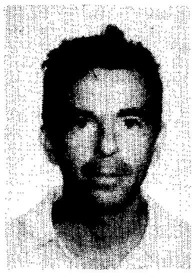
$$\|\tilde{f}_k(\mathbf{z}', \mathbf{w}', t) - \tilde{f}(\mathbf{z}'', \mathbf{w}'', t)\| = |\beta|\cdot\|w'_{k-MN} - w''_{k-MN}\|. \tag{A33}$$

In summary, if we choose $K'_1 = \max_{1 \leqslant k \leqslant MN} K'_k$, $K_1 = \max(1, K'_1)$ and $K'_2 = 5\cdot\max(|\alpha|, |D|)$, $K_2 = \max(K'_2, 2, |\beta|)$ then, recalling the waveform vector norm definition:

$$\|\tilde{\mathbf{f}}(\mathbf{z}', \mathbf{w}', t) - \tilde{\mathbf{f}}(\mathbf{z}'', \mathbf{w}'', t)\| \leqslant K_1\|\mathbf{z}' - \mathbf{z}''\| + K_2\|\mathbf{w}' - \mathbf{w}''\|. \tag{A34}$$

This proves that $\tilde{\mathbf{f}}$ is uniformly Lipschitz and, for theorem 2.1, proves the thesis. ■

AUTHORS' BIOGRAPHIES



Luigi Fortuna was born in Siracusa, Italy, in 1953. He received the electrical engineering degree from the University of Catania in 1977. He is presently Full Professor of System Theory at the University of Catania. His scientific interests currently include model order reduction, identification, control theory, neural networks and nonlinear systems.

Gabriele Manganaro was born in Catania, Italy, on 31st August 1969. He received the Dr.Eng. degree in electronic engineering from the University of Catania on 28th January 1994. He is currently a Ph.D. student in electrical engineering at the Dipartimento Elettrico, Elettronico e Sistemistico (DEES) of the University of Catania. His research interests include the cellular neural networks, nonlinear dynamic circuits and systems, transputer-based parallel architectures, electronic design and simulation.

Giovanni Muscato was born in Catania, Italy, in 1965. He received the electrical engineering degree from the University of Catania, in 1988. Following graduation he worked with "Centro di Studi sui Sistemi" in Turin. Since 1990 he joined the Dipartimento Elettrico, Elettronico e Sistemistico of the University of Catania, where he presently holds the position of Assistant Professor of Automatic Control. His principal research interest include robust control, model reduction and the use of neural networks in the modeling and control of dynamical systems.

Giuseppe Nunnari was born in Catania, Italy, in 1955. He received the electrical engineering degree from the University of Catania in 1979. Following graduation, he worked as software engineer with Olivetti (Computer Division) until 1983. Since 1983, he has been a researcher of the Italian National Research Council. Since 1992, he is an associate professor of Automatic Control at the University of Catania. His scientific interest include modeling, model reduction, control applications, fuzzy logic and parallel architectures.