Designing Tissue-like P Systems for Image Segmentation on Parallel Architectures

Javier Carnero[1], Daniel Díaz-Pernil[1], Miguel A. Gutiérrez-Naranjo[2]

[1] Computational Algebraic Topology and Applied Mathematics Research Group
Department of Applied Mathematics I
University of Sevilla
Avda. Reina Mercedes s/n, 41012, Sevilla, Spain
javier@carnero.net, sbdani@us.es

[2] Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012, Sevilla, Spain
magutier@us.es

Summary. Problems associated with the treatment of digital images have several interesting features from a bio-inspired point of view. One of them is that they can be suitable for parallel processing, since the same sequential algorithm is usually applied in different regions of the image. In this paper we report a work-in-progress of a hardware implementation in *Field Programmable Gate Arrays* (FPGAs) of a family of tissue-like P systems which solves the segmentation problem in digital images.

1 Introduction

Membrane Computing is a computational paradigm inspired in the functioning of living cells and tissues. One of its characteristic features is the use of parallelism as a computation tool. In many of the models, the devices perform the computation by applying parallelization in a double sense: on the one hand, several rules can be applied simultaneously in each membrane; on the other hand, all the membranes perform the computation at the same time.

In spite of recent efforts [15], it seems that in the next future there will not be an implementation of P systems *in vivo* or *in vitro*. All the possible approaches to the theoretical model lean on the current computer architectures.

In this line, many efforts have been made for obtaining a *simulation* of the P system behavior with current computers [13, 16]. Most of these simulators are thought for running on one-processor computers. These sequential machines only perform one action per time unit and the parallelism of the membrane computing devices is lost. This bottle-neck produces a serious discrepancy between the theo-

retical efficiency of the P systems and the realistic resources needed for performing a computation.

In the last years, according with the development of new parallel architectures, new attempts have been made for approaching the computation of P systems by performing several actions in the same step. This does not mean a real implementation of the P system, but it can be considered as a new step toward a more realistic simulation.

The first parallel and distributed simulators were presented in 2003. In [12], a parallel implementation of transition P systems was presented. The program was designed for a cluster of 64 dual processor nodes and it was implemented and tested on a Linux cluster at the National University of Singapore. In [32], a purely distributive simulator of P systems was presented. It was implemented using Java's *Remote Methods Invocation* to connect a number of computers that interchange data. The class of P systems that the simulator can accept is a subset of the $NOP_2(coo, tar)$ family of systems, which have the computational power of Turing machines.

Also in 2003, Petreska and Teuscher [29] presented a parallel hardware implementation of a special class of membrane systems. The implementation was based on a universal membrane hardware component that allows efficiently run P system on a reconfigurable hardware known as *Field Programmable Gate Arrays* (FPGAs) [35]. Recently, a new research line has arisen due to a novel device architecture called CUDA^{TM}, (Compute Unified Device Architecture) [39]. It is a general purpose parallel computing architecture that allows the parallel compute engine in NVIDIA Graphic Processor Units (GPUs) to solve many complex computational problems in a more efficient way than on a CPU [5, 6, 7]. Following the research line started in [29], Van Nguyen *et al.* have proposed the use hardware implementation for membrane computing applications [22, 23, 24, 25] based on reconfigurable computing technology called Reconfig-P.

In this paper, we also explore the possibilities of the *Field Programmable Gate Arrays* (FPGAs) for building a hardware implementation of P systems. The P system model chosen for the implementation has been tissue-like P systems and as a case study we consider the *segmentation problem* in 2D images.

Segmentation in computer vision (see [31]), refers to the process of partitioning a digital image into multiple segments (sets of pixels). The goal of segmentation is to simplify and/or change the representation of an image into something that is more meaningful and easier to analyze. Image segmentation is typically used to locate objects and boundaries (lines, curves, etc.) in images. More precisely, image segmentation is the process of assigning a label to every pixel in an image such that pixels with the same label share certain visual characteristics. Technically, the process consists on assigning a label to each pixel, in such way that pixels with the same label form a meaningful region. There exist different techniques to segment an image. Some techniques are *clustering methods* [1, 36], *histogram-based methods* [34], *Watershed transformation methods* [33], image pyramids methods

[18] or *graph partitioning methods* [37, 38]. Some of the practical applications of image segmentation are medical imaging [36] or face recognition [17].

Segmentation in Digital Imagery has several features which make it suitable for techniques inspired by nature. One of them is that it can be paralleled and locally solved. Regardless how large is the picture, the segmentation process can be performed in parallel in different local areas of it. Another interesting feature is that the basic necessary information can be easily encoded by bio-inspired representations.

In the literature, one can find several attempts for bridging problems from Digital Imagery with Natural Computing as the works by K.G. Subramanian *et al.* [8, 9] or the work by Chao and Nakayama where Natural Computing and Algebraic Topology are linked by using Neural Networks [10] (extended Kohonen mapping). In this paper, we will use an information encoding and techniques borrowed from Membrane Computing. This paper is a new step in the research started at [4], where the authors present an implementation of a membrane solution of a segmentation problem using hardware programming. In this paper, we present a different family of tissue-like P systems to solve the problem and report the hardware implementation. In what follows we assume the reader is already familiar with the basic notions and the terminology underlying P systems[3].

The paper is organized as follows: firstly, we present our bio-inspired formal framework. Next, we present a family of tissue-like P systems designed to obtain an edge-based segmentation of a 2D digital image. Then, general considerations about designing hardware P systems are studied, focusing on the segmentation problem. The paper finishes with some conclusions and future work.

2 Formal Framework: Tissue-like P Systems

Tissue-like P systems were presented by Martín–Vide *et al.* in [21]. They have two biological inspirations (see [20]): intercellular communication and cooperation between neurons. The common mathematical model of these two mechanisms is a network of processors dealing with symbols and communicating these symbols along channels specified in advance.

The main features of this model, from the computational point of view, are that cells do not have polarization and the membrane structure is a general graph.

Formally, a *tissue-like P system* with input of degree $q \geq 1$ is a tuple

$$\Pi = (\Gamma, \Sigma, \mathcal{E}, w_1, \ldots, w_q, \mathcal{R}, i_\Pi, o_\Pi),$$

where

1. Γ is a finite *alphabet*, whose symbols will be called *objects*;
2. $\Sigma (\subset \Gamma)$ is the input alphabet;

[3] We refer to [26] for basic information in this area, to [28] for a comprehensive presentation and the web site [40] for the up-to-date information.

3. $\mathcal{E} \subseteq \Gamma$ (the objects in the environment);
4. $w_1, \ldots, w_q$ are strings over Γ representing the multisets of objects associated with the cells at the initial configuration;
5. $\mathcal{R}$ is a finite set of communication rules of the following form:

$$(i, u/v, j)$$

for $i, j \in \{0, 1, 2, \ldots, q\}, i \neq j$, $u, v \in \Gamma^*$;
6. $i_\Pi \in \{1, 2, \ldots, q\}$ is the input cell;
7. $o_\Pi \in \{0, 1, 2, \ldots, q\}$ is the output cells

A tissue-like P system of degree $q \geq 1$ can be seen as a set of q cells (each one consisting of an elementary membrane) labelled by $1, 2, \ldots, q$. We will use 0 to refer to the label of the environment, i_Π denotes the input region and o_Π denotes the output region (which can be the region inside a cell or the environment).

The strings $w_1, \ldots, w_q$ describe the multisets of objects placed in the q cells of the P system. We interpret that $\mathcal{E} \subseteq \Gamma$ is the set of objects placed in the environment, each one of them available in an arbitrary large amount of copies.

The communication rule $(i, u/v, j)$ can be applied over two cells labelled by i and j such that u is contained in cell i and v is contained in cell j. The application of this rule means that the objects of the multisets represented by u and v are interchanged between the two cells. Note that if either $i = 0$ or $j = 0$ then the objects are interchanged between a cell and the environment.

Rules are used as usual in the framework of membrane computing, that is, in a maximally parallel way (a universal clock is considered). In one step, each object in a membrane can only be used for one rule (non-deterministically chosen when there are several possibilities), but any object which can participate in a rule of any form must do it, i.e., in each step we apply a maximal set of rules.

A *configuration* is an instantaneous description of the P system Π. Given a configuration, we can perform a computation step and obtain a new configuration by applying the rules in a parallel manner as it is shown above. A sequence of computation steps is called a *computation*. A configuration is *halting* when no rules can be applied to it. Then, a computation halts when the P system reaches a halting configuration.

3 Segmenting Digital Images

A *point set* is simply a topological space consisting of a collection of objects called points and a topology which provides for such notions as *nearness* of two points, the *connectivity* of a subset of the point set, the *neighborhood* of a point, *boundary points*, and *curves* and *arcs*.

The most common point sets occurring in image processing are discrete subsets of N-dimensional Euclidean space $\mathbb{R}^n$ with $n = 1, 2$ or 3 together with the discrete topology. There is no restriction on the shape of the discrete subsets of $\mathbb{R}^n$ used in applications of image algebra to solve vision problems.

For a point set X in Z, a *neighborhood function* from X in Z, is a function $N : X \to 2^Z$. For each point $x \in X$, $N(x) \subseteq Z$. The set $N(x)$ is called a *neighborhood* for x.

There are two neighborhood function on subsets of $\mathbb{Z}^2$ which are of particular importance in image processing, the *von Neumann* neighborhood and the *Moore* neighborhood. The first one $N : X \to 2^{\mathbb{Z}^2}$ is defined by $N(x) = \{y : y = (x_1 \pm j, x_2) \text{ or } y = (x_1, x_2 \pm k),\ j,k \in \{0,1\}\}$, where $x = (x_1, x_2) \in X \subset \mathbb{Z}^2$. While the Moore neighborhood $M : X \to 2^{\mathbb{Z}^2}$ is defined by $M(x) = \{y : y = (x_1 \pm j, x_2 \pm k),\ j,k \in \{0,1\}\}$, where $x = (x_1, x_2) \in X \subset \mathbb{Z}^2$. The von Neumann and Moore neighborhood are also called the *four neighborhood* (4-adjacency) and *eight neighborhood* (8-adjacency), respectively. In this paper, we work with 4-adjacency. The point sets with the usual operations has an algebra structure (see [30]).

An *Z-valued image* on X is any element of Z^X. Given an Z-valued image $I \in Z^X$, i.e. $I : X \to Z$, then Z is called the set of possible range values of I and X the spatial domain of I. The graph of an image is also referred to as the *data structure representation* of the image. Given the data structure representation $I = \{(x, I(x)) : x \in X\}$, then an element $(x, I(x))$ is called a *picture element* or *pixel*. The first coordinate x of a pixel is called the *pixel location* or *image point*, and the second coordinate $I(x)$ is called the *pixel value* of I at location x.

For example, X could be a subset of $\mathbb{Z}^2$ where $x = (i, j)$ denotes spatial location, and Z could be a subset of $\mathbb{N}$, $\mathbb{N}^3$, etc. So, given an image $I \in Z^{\mathbb{Z}^2}$, a pixel of I is the form $((i, j), I(x))$, which will be denoted by $I(x)_{ij}$. We call the *set of colors* or *alphabet of colors* to the image set of the function I with domain X and the image point of each pixel is called *associated color*. We can consider an order in this set. In this paper, we denote Z as $\mathcal{C}_I$. Usually, we consider in digital image a predefined alphabet of colors $\mathcal{C}$. We define $h = |\mathcal{C}|$ as the size (number of colors) of $\mathcal{C}$. In this paper, we work with images in grey scale, then $\mathcal{C} = \{0, \ldots, 255\}$, where 0 codify the black color and 255 the white color.

By technical reasons, we use below different ways to codify a same pixel. For example, if we take the pixel $((i, j), a)$ we could codify with the following expressions: a_{ij}, A_{ij}, a'_{ij}, $\overline{a}_{ij}$, $(a, l)_{ij}$ with $l \in \mathbb{N}$, etc.

A *region* could be defined by a subset of the domain of I whose points are all mapped to the same (or similar) pixel value by I. So, we can consider the region R_i as the set $\{x \in X : I(x) = i\}$ but this kind of regions has not to be connected. We prefer to consider a region r as a maximal connected subset of a set like R_i. We say two regions r_1, r_2 are adjacent when at less a pair of pixel $x_1 \in r_1$ and $x_2 \in r_2$ are adjacent. We say x_1 and x_2 are *border pixels*. If $I(x_1) < I(x_2)$ we say x_1 is an *edge pixel*. The set of connected edge pixels with the same pixel value is called a *boundary* between two regions.

From a general point of view, segmentation refers to the process of partitioning a digital image into multiple regions. Thresholding is a method of image segmentation whose basic aim is to obtain a binary image from a colour one. The idea is to split the set of pixels into two sets (black and white) depending on its bright and a fixed valued, the *threshold*. If the bright of the pixel is greater than the threshold,

then the pixel is labelled as *object*. Otherwise, it is labelled as *background*. After labelling, a new binary image is created by colouring each pixel white or black, depending on the label.

The basic thresholding method can be generalized in a natural way. Instead of getting a binary image by labelling the original set of pixels by $\{0, 1\}$, we can consider a larger set of labels, $\{1, \ldots, k\}$ so we obtain a final image with k levels. Another natural generalization is to replace the colour information by another scale on the features of the pixel (bright, intensity, gray scale, etc.).

Edge detection is an important operation in a large number of image processing applications, such as image segmentation, character recognition and scene analysis.

In this paper we work with the first one, the *edge-based segmentation of 2D digital images problem (2D-ES problem)*, which is described as follows: *Given a digital 2D image with pixels of (possibly) different colors, obtain the boundaries of regions in that image.*

In order to provide a logarithmic-time uniform solution to our problem, we design a family of tissue-like P systems, Π. Given an image I of size n^2, we take the P system $\Pi(n, k)$ of the family to work with I. The input data (image I) is codified by a set of objects a'_{ij}, with $a \in \mathcal{C}$ and $1 \leq i, j \leq n$ and k is referred to the number of processing cells. So, when we work with a parallel architecture we do not have to know previously an exact number of processors to work. Then, we introduce the parameter k to solve this problem.

The functioning of a P system of the family consists of the following stages:

- First of all, the P system generates 8 auxiliary copies of the input data. Then, we have 9 codifications of the input image, but one of them is distinguished of the rest. So, we can work with each pixel without taking into account what happens with the rest of the image.
- Second, the P system applies a *basic noise filter* in order to eliminate some pickle noise that could affect the segmentation process. The P system will apply the largely used average filter because of its simplicity and good results. For each pixel, the process consists of calculating the average average of its adjacent pixels. If the distance between the pixel and its average is greater than a threshold ρ, the pixel will be considered as noise and it will be replaced by its average colour.
- Next, the P system performs a thresholding of the image to solve the problem of degradation of colours of pixels in the boundary of adjacent regions with different colours.
- Once this process is finished, the P system applies a translation of rules defined in [11] obtaining an edge-based segmentation of the image took of the previous stage.

The family $\mathbf{\Pi} = \{\Pi(n, k) : n, k \in \mathbb{N}\}$ of tissue-like P systems of degree $k + 1$ is defined as follows:

For each $n, k \in \mathbb{N}$,

$$\Pi(n, k) = (\Gamma, \Sigma, \mathcal{E}, w_1, \ldots, w_{k+1}, \mathcal{R}, i_\Pi, o_\Pi),$$

defined as follows:

- $\Gamma = \Sigma \cup \{a_{ij}, a''_{ij}, \bar{a}_{ij}, A_{ij}, A'_{ij}, A''_{ij}, \overline{A}_{ij}, \overline{\overline{A}}_{ij}, (a,1)_{ij}, (a,2)_{ij}, (a,3)_{ij} \ : \ 1 \leq i,j \leq n,\ a \in \mathcal{C}\}$ is the working alphabet;
- the input alphabet is $\Sigma = \{a'_{ij} : 1 \leq i,j \leq n,\ a \in \mathcal{C}, I(i,j) = a\}$;
- the environment alphabet is $\mathcal{E} = \Gamma \setminus \Sigma$;
- the multisets of the cells are $w_1 = \{\{\nu^3_{ij}, \nu^3_{ji} \ : \ i = 0, n+1, 0 \leq j \leq n+1\}\}$, $w_2 = \cdots = w_{k+1} = T^{\lceil n^2/k \rceil}$, respectively. We call to the last k cells as *processing cells*;
- R is the following set of communication rules:

1. $(1, a'_{ij}/a^8_{ij} A_{ij}, 0)$
   for $1 \leq i,j \leq n$.
   These rules are used to generate new elements, so the P system can work in parallel with each pixel and forget what happen with the rest of the image. The P system first uses these elements to work with the noise of our image.

2. $$\left(\begin{array}{lll} & c_{i-1j-1} & d_{i-1j} \ e_{i-1j+1} \\ 1, & b_{ij-1} \quad A_{ij} & f_{ij+1} \end{array} \ / \ T, t \right)$$
   $$\begin{array}{lll} & & \\ & l_{i+1j-1} & h_{i+1j} \ g_{i+1j+1} \end{array}$$
   for
   - $1 \leq i,j \leq n$,
   - $a,b,c,d,e,f,g,h,l \in \mathcal{C} \cup \{\nu\}$.

   This type of rules are used to translate each object A_{ij} and one copy of their neighbours (objects) to a processing cell. We are sure that all the pixels not go to the same cell, because our P system has n^2 or n^2+1 objects T spread over processing cells, each one with a similar number of copies of T.

3. $$\left(\begin{array}{lll} & c_{i-1j-1} & d_{i-1j} \ e_{i-1j+1} \\ t, & b_{ij-1} \quad A_{ij} & f_{ij+1} \end{array} \ / \ \alpha'_{ij}, 0 \right)$$
   $$\begin{array}{lll} & & \\ & l_{i+1j-1} & h_{i+1j} \ g_{i+1j+1} \end{array}$$
   for
   - $1 \leq i,j \leq n$,
   - $a,b,c,d,e,f,g,h,l \in \mathcal{C} \cup \{\nu\}$,
   - We take μ as the number of pixels with colors in $\mathcal{C}$ and $\nu = 0$. Then, $av(a) = (b+c+d+e+f+g+h+i)/\mu$,
   - α is the nearest colour in $\mathcal{C}$ to the average colour $av(a)$ with $|\alpha - av(a)| > \rho$, with $\rho \in \mathbb{R}$.

4. $$\left(\begin{array}{lll} & c_{i-1j-1} & d_{i-1j} \ e_{i-1j+1} \\ t, & b_{ij-1} \quad A_{ij} & f_{ij+1} \end{array} \ / \ a'_{ij}, 0 \right)$$
   $$\begin{array}{lll} & & \\ & i_{i+1j-1} & h_{i+1j} \ g_{i+1j+1} \end{array}$$
   for
   - $1 \leq i,j \leq n$,
   - $a,b,c,d,e,f,g,h,i \in \mathcal{C} \cup \{\nu\}$,

- We take μ as the number of pixels with colors in $\mathcal{C}$ and $\nu = 0$. Then, $av(a) = (b + c + d + e + f + g + h + i)/\mu$,
- $|a - av(a)| \leq \rho$, where $\rho_1 \in \mathbb{R}$.

This set of rules is used to detect the noise and correct it with the average colour of its adjacent pixels. We find here a local thresholding (with respect to the colors) with predefined threshold ρ. In fact, we are simulating one of the more typical algorithms to remove noise. The P system changes the notation of the objects which are codifying pixels and they adopt the form a'_{ij}, with $a \in \mathcal{C}$.

5. $(t, b'_{ij}/A'_{ij}, 0)$
   for
   - $1 \leq i, j \leq n$,
   - $\tau = (|\mathcal{C}|/\rho_2)$, $l = 0, 1, 2, \ldots, \rho_2$,
   - If $b \in \mathcal{C}$ then $a \in \mathcal{C}$ $(a < b \leq a + (\tau - 1)$ and $a = \tau \cdot l)$ or $(b = a = \tau \cdot l)$,
   - If $b = \nu$ then $A = \nu$.

   These rules are used to discretize the colors dividing the set of colors in ρ_2 subsets of length ν. We find here a general thresholding (with respect to the colors) with predefined threshold ν.

6. $(t, A'_{ij}/T, 1)$
   for
   - $0 \leq i, j \leq n + 1$, $2 \leq t \leq k + 1$,
   - $a \in \mathcal{C}$.

   This set of rules are used to send our transformed image to the cell 1. Now, the objects A'_{ij} encode the pixels of our image.

7. $(1, A'_{ij}/A''_{ij}\overline{A}_{ij}\overline{a}^8_{ij}, 0)$
   for
   - $0 \leq i, j \leq n + 1$,
   - $a \in \mathcal{C} \cup \{\nu\}$.

   The P system uses these rules to generate enough copies of our image to perform the segmentation process in the cells $2, \ldots, k$ and $k+1$. The objects A''_{ij} are used in the second part of the segmentation. The rest of the objects are used in the first part of the segmentation.

8. $$\left(t, \begin{array}{ccc} \overline{c}_{i-1j-1} & \overline{d}_{i-1j} & \overline{e}_{i-1j+1} \\ \overline{b}_{ij-1} & \overline{A}_{ij} & \overline{f}_{ij+1} \\ \overline{i}_{i+1j-1} & \overline{h}_{i+1j} & \overline{g}_{i+1j+1} \end{array} / \ T, t \right)$$
   for
   - $1 \leq i, j \leq n$,
   - and $a, b, c, d, e, f, g, h, i \in \mathcal{C} \cup \{\nu\}$.

   These rules are defined to send new objects to the processing cells to do the first part of the segmentation. We look for edge pixels.

9. $(t, \overline{A}_{ij}\overline{b}_{kl} / \overline{\overline{A}}_{ij}\overline{b}_{kl}, 0)$,
for
   - $1 \leq i, j, k, l \leq n$, $(i, j), (k, l)$ adjacent pixels,
   - $a, b \in \mathcal{C}$ and $a < b$.

   These rules are used to mark edge pixels. In fact, the the P system brings from the environment an object of the form $\overline{\overline{A}}_{ij}$ for each edge pixel. Our problem is the edge pixels not always are adjacent. So, we do not have an only one set of connected edge pixel forming a boundary. Then, we should add the necessary pixel to connect all the edge pixels of a boundary.
10. $(t, \overline{\overline{A}}_{ij} / T, 1)$
for
   - $1 \leq i, j \leq n$,
   - $a \in \mathcal{C}$.

   These rules send the edge pixels to the cell 1.
11. $(1, \overline{\overline{A}}_{ij} / (a, 1)^2_{ij}, 0)$
for
   - $0 \leq i, j \leq n + 1$,
   - $a \in \mathcal{C} \cup \{\nu\}$.

   The P system uses these rules to generate two copies of our edge pixels to perform the second part of the segmentation in processing cells.
12. $(1, A''_{ij} / (a, 2)^2_{ij}, 0)$
for
   - $0 \leq i, j \leq n + 1$,
   - $a \in \mathcal{C} \cup \{\nu\}$.

   The P system uses these rules to generate enough copies of our image to perform the second part of the segmentation in processing cells.
13. $\left(1, \begin{matrix}(a,1)_{i-1j-1} & (a,2)_{i-1j} \\ (b,2)_{ij-1} & (a,1)_{ij}\end{matrix} \;/\; T, t\right) \left(1, \begin{matrix}(b,2)_{i-1j-1} & (a,1)_{i-1j} \\ (a,1)_{ij-1} & (a,2)_{ij}\end{matrix} \;/\; T, t\right)$
$\left(1, \begin{matrix}(a,1)_{i-1j-1} & (b,2)_{i-1j} \\ (a,2)_{ij-1} & (a,1)_{ij}\end{matrix} \;/\; T, t\right) \left(1, \begin{matrix}(a,2)_{i-1j-1} & (a,1)_{i-1j} \\ (a,1)_{ij-1} & (b,2)_{ij}\end{matrix} \;/\; T, t\right)$
for
   - $1 \leq i, j \leq n$,
   - $a, b \in \mathcal{C}$.

   These rules are defined to send new objects to the processing cells to do the second part of the segmentation. We look for new edge pixels.
14. $\left(1, \begin{matrix}(a,1)_{i-1j-1} & (a,2)_{i-1j} \\ (b,2)_{ij-1} & (a,1)_{ij}\end{matrix} \;/\; \begin{matrix}(a,3)_{i-1j-1} & (a,3)_{i-1j} \\ (b,2)_{ij-1} & (a,3)_{ij}\end{matrix}, t\right)$
$\left(1, \begin{matrix}(b,2)_{i-1j-1} & (a,1)_{i-1j} \\ (a,1)_{ij-1} & (a,2)_{ij}\end{matrix} \;/\; \begin{matrix}(b,2)_{i-1j-1} & (a,3)_{i-1j} \\ (a,3)_{ij-1} & (a,3)_{ij}\end{matrix}, t\right)$

$$\left(1, \begin{matrix}(a,1)_{i-1j-1} & (b,2)_{i-1j} \\ (a,2)_{ij-1} & (a,1)_{ij}\end{matrix} / \begin{matrix}(a,3)_{i-1j-1} & (b,2)_{i-1j} \\ (a,3)_{ij-1} & (a,3)_{ij}\end{matrix}, t\right)$$
$$\left(1, \begin{matrix}(a,2)_{i-1j-1} & (a,1)_{i-1j} \\ (a,1)_{ij-1} & (b,2)_{ij}\end{matrix} / \begin{matrix}(a,3)_{i-1j-1} & (a,3)_{i-1j} \\ (a,3)_{ij-1} & (b,2)_{ij}\end{matrix}, t\right)$$

for
- $1 \leq i, j \leq n$,
- $a, b \in \mathcal{C}$.

These rules are used to complete the set of edge pixels of our image.

15. $(t, (a,3)_{ij}/\lambda, 1)$
    for
    - $1 \leq i, j \leq n$,
    - $a \in \mathcal{C}$.

    These rules send to the cell 1 the edge pixels.

16. We can find more than one copy of an specific edge pixel, so if we wish only one copy of each edge pixel we can add a new type of rules:
    $(t, (a,3)_{ij}(a,3)_{ij}/(a,3)_{ij}, 1)$
    for
    - $1 \leq i, j \leq n$,
    - $a \in \mathcal{C}$.

- $i_\Pi = o_\Pi = 1$.

4 The Hardware Design

In [11], some preliminary segmentation results were obtained using the *tissue simulator* developed in [3]. Such a *tissue simulator* follows one of the common features of the first generation of simulators of P systems (see [13]): the lack of efficiency in favor of expressiveness. Therefore, experiments performed using this tool were extremely slow, and it could only use synthetic images of at most 30×30 pixels. Recently, a new sequential software was presented in [14], implementing ideas borrowed from [11].

In order to make the hardware design of a tissue-like P system there are several considerations that must be considered:

1. On a tissue-like P system, not only each cell evolve in a parallel manner. Every rule in every cell must be executed as many times as possible at each step. Thus, if we want that the hardware system to work exactly like the theoretical model, the system has to implement as many *minimal computation units* as the maximum number of rules in all the cells that could be executed in the same step in order to be fully parallel. If we are designing a general tissue-like P system which we want to use to configure different tissue-like P systems that solve specific problems, this is probably the main problem, as this number is defined by each P system configuration. In this case, the only way to do this is

to design the *minimal computation units* as small as possible in terms of chip area, in order to have the maximum number of them. Then, if this number is not enough to solve our problem, the system can be designed in order to do the following:

- Separate each conflicting step, into two or more sub-steps. So, in the first sub-step the system executes all the possible rules, using a piece of memory to save the results, and then it continues executing the rest of the rules that could not be executed before due to insufficient *minimal computation units* .
- Connect with other clone system(s) to solve the hole problem using more computation capacity. This options is not always possible and, in general, it is more difficult to design that the first one, but it can be the best choice dealing with hard computation problems.

On the other hand, if we want to design a specific P system, usually the best choice to deal with this issue is finding sets of rules that are mutually exclusive, that is, rules that we know that if the executing condition is true for one of them, then we know that the other ones in the same set cannot be executed. Thus, in fact we have only to design one *minimal computation unit* for each set of rules, optimizing the system area. This is the case of the described segmentation problem, in which we know that we can define only one set of rules mutually exclusive for each pixel, so in fact the system will have as many *minimal computation units* as the biggest segmentation. So the computational order will be constant, and the spatial order will be lineal.

2. The copy rules are necessary in the theoretical tissue-like P system, but in the hardware design it is not necessary in general to implement them as rules like the other ones, since they can be seen as parallel readings of some information.

   So usually those rules can be ignored in the design, seeing them as multi-lectures of the data that is trying to copy the rule. Also is easily to transform those rules into asynchronous rules. That is the segmentation case that we present, where *main rules* are synchronized by the clock system representing the synchronous P system, and the *copy rules* are asynchronous and are implicit in the interconnection circuit of the design.
3. Depending on the variant of tissue-like P systems we work with, cells could create or remove other cells during the execution in order to solve the problem. This is one of the biggest problems when simulating tissue-like P systems in software in a efficient way, but could not be the case in hardware. The FPGAs can be reconfigured while the system is still working. This feature help us to design the addition or removal cell rules as partial *on air* reconfigurations of the system *on air* easily. The only thing that we have to worry about is that those operations are not fast in terms of time, so the steps with those rules will be slower than the other ones. Because of that, trying to avoid those kind of rules while defining the P system is a good practice. The segmentation problem described has no rule of this kind.

4. The halting condition can be redefined in order to save some final computation step. This is the case of rule 9 on the segmentation problem. In most of synchronous P systems, we can know that the system has finished without make an explicit operation. For example, in our design we know that the system stops three clock cycles after the beginning. Another simple option can be found by observing the system behavior.

5. The system has to be always running while there are instances of the problem that does not have been solved yet. In fact that is a consideration that have to be done every time a hardware design is made, besides designing the P system it is important that it can return the results whereas the system is starting with a new problem. So, perhaps this is a consideration that has to be only considered not only in the design step, but in the previous theoretical P system definition before.

4.1 Segmentation Problem Design Based on FPGA

Following the segmentation example, an formal hardware system design based on the tissue-like P system described above is shown in Figures 1, 2, 3 and 4. It has been done by following the previous considerations. The system consists on *processing units* capable of dealing with 4×4 images. These units can be combined like a puzzle in order to process $n \times m$ images.

Each pixel in the image is codified with 56 bits in order to represent the theoretical objects (it contains color information, original color information, and type of object). Using this codification, a 4×4 section of the initial image is passed through the image port of each *processing unit*. Also additional information about the neighborhood of the 4×4 is required in order to work correctly, using the $blec, blrec, trec, tlec$, and bb, rb, tb, lb buses for that. If the neighborhood (or a part of it), does not exist those inputs will be at high impedance (*ghost pixel*).

The t and k ports specify the maximum distance between the pixel and its average (noise filter), and the number of different levels for the thresholding respectively. The different system steps are controlled by the clock signal (CLK).

As it is shown in figure 1, inside the *processing unit* are 16 *pixel processing units* capable of execute any rule for each pixel and each step (using the techniques described before in the first point.). The signal *change* is used to feed this units with the input data (the original image), or feed them with its own output (representing in that way the copy rules as described before in item two).

These *pixel processing units* shown in figure 2 receives a pixel and its neighborhood, and the t, k and CLK signals, and send this information to four units that implements the four sets of rules mutually exclusive mentioned before. The results are collected and processed as output. In general a fixed group of *pixel processing units* will represent a fixed group of cells in the theoretical model. But looking at the described tissue-like P, we have that each cell except cells zero and one is representing the computation of one pixel, so there is exactly one *pixel processing*

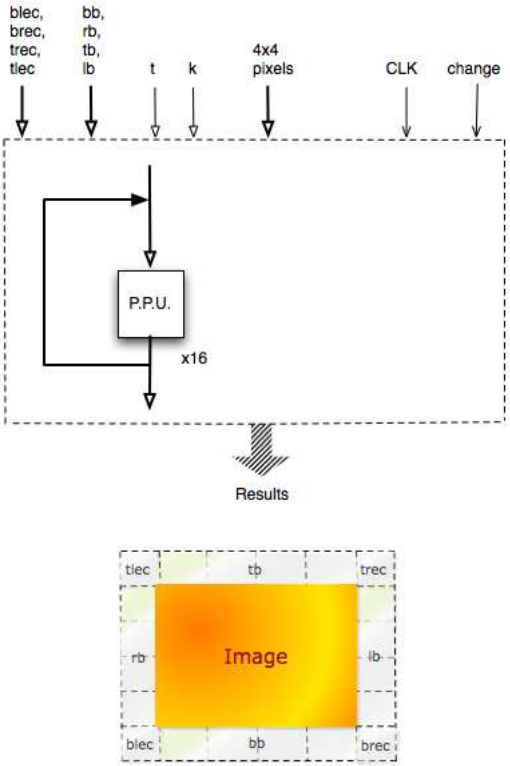

Fig. 1. Processing unit, and neighborhood of an image

unit for each cell. Then we can say that these units represents each cell in the theoretical model except zero and one.

The four units that implement the four sets of rules mutually exclusive are shown in figure 3: removing noise rules (types 3 and 4), thresholding rules (type 5), rules of the first part of the segmentation (type 9) and rules of the second part of the segmentation (type 14). The rest of rules of the theoretical family of systems are rules of coping and sending objects. These units detect automatically if the input data is a corner, an edge, or an interior pixel. Finally, in order to deal with bigger images, we can use the $blec, blrec, trec, tlec$, and bb, rb, tb, lb buses to interconnect as many as *processing units* we need (figure 4). A very simple interconnection circuit is necessary in order to give the input data to the different processing units.

The implementation of this hardware tool allows the system to apply the maximum number of rules at each moment, using the *pixel units* to solve the whole problem. Therefore, the system works exactly like the theoretical model in terms of complexity, time, concurrency and results. As said before, the implementation

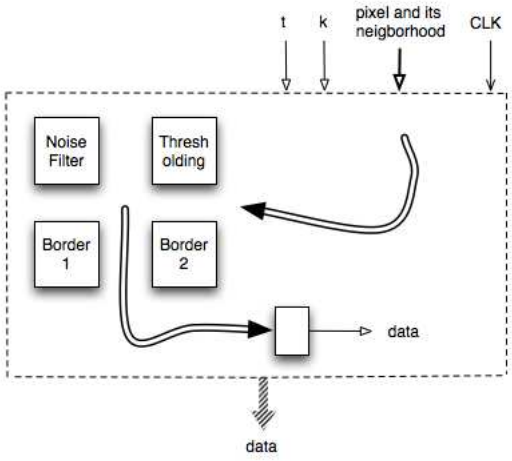

Fig. 2. Pixel Processing Unit

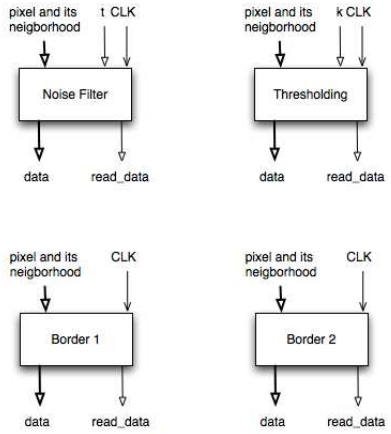

Fig. 3. Chips that implements the sets of rules mutually exclusive

of this design reveals that in fact the system is able to process any image of size $n \times m$ by using at most four clock cycles.

In figure 5, it is shown a simulation of the code following the described design that deals with 16×16 images, and some simple results using a SP605 Xilinx board with a Spartan 6 XC6SLX45T FPGA chip.

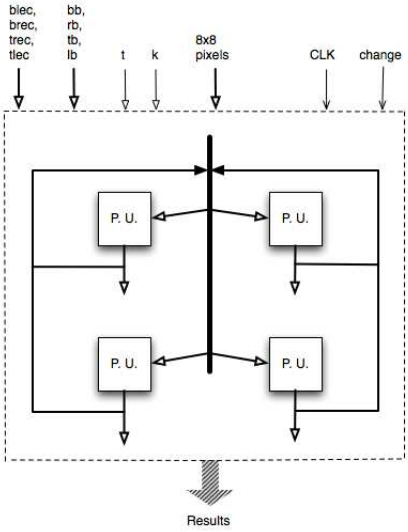

Fig. 4. Processing Units Interconnection

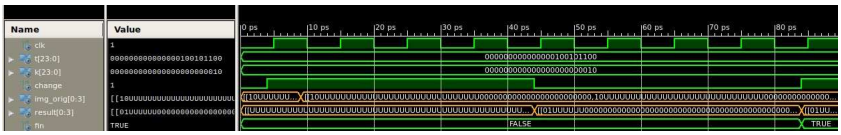

Fig. 5. Hardware design

Fig. 6. 16x16 black and white image segmentation

Fig. 7. 16x16 black and white image segmentation

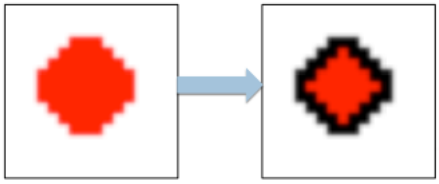

Fig. 8. 16x16 color image segmentation

5 Conclusions and Future Works

Problems associated with the treatment of Digital Images have several interesting features from a bio-inspired point of view. One of them is that they can be suitable for parallel processing, since the same sequential algorithm is usually applied in different regions of the image.

In this paper, we study the advantages and drawbacks of considering a hardware implementation of tissue-like P systems solving the segmentation problem on a hardware programming tool (FPGA). The theoretical study has been made via the language programming VHDL [2] and currently we are in the process of the real hardware implementation.

In addition, although the segmentation example showed here is a synchronous tissue-like P system, we want in the next future to work with asynchronous tissue-like P systems in order to optimize performance.

Many questions remains open as future work. One of them is the treatment of the noise in images with Membrane Computing techniques, or the parallelization and automatization of the choice of the threshold by artificial intelligence techniques.

Acknowledgements

DDP and MAGN acknowledge the support of the projects TIN-2009-13192 of the Ministerio de Ciencia e Innovación of Spain and the support of the Project of Excellence of the Junta de Andalucía, grant P08-TIC-04200. JC acknowledges the support of the project MTM2009-12716 of the Ministerio español de Educación y Ciencia, the project PO6-TIC-02268 of Excellence of Junta de Andalucía, and the *Computational Topology and Applied Mathematics* PAICYT research group FQM-296.

References

1. Abdala, D.D., Jiang, X.: Fiber segmentation using constrained clustering. In: Zhang, D., Sonka, M. (eds.) ICMB. Lecture Notes in Computer Science, vol. 6165, pp. 1–10. Springer (2010)

2. Ashenden, P.J.: The Designer's Guide to VHDL. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2nd edn. (2001)
3. Borrego-Ropero, R., Díaz-Pernil, D., Pérez-Jiménez, M.J.: Tissue simulator: A graphical tool for tissue P systems. In: Vaszil, G. (ed.) Proceedings of the International Workshop Automata for Cellular and Molecular Computing. pp. 23–34. MTA SZTAKI, Budapest, Hungary (August 2007), satellite of the 16th International Symposium on Fundamentals of Computational Theory
4. Carnero, J., Díaz-Pernil, D., Molina-Abril, H., Real, P.: Image segmentation inspired by cellular models using hardware programming. Image-A 1(3), 143–150 (2010)
5. Cecilia, J.M., García, J.M., Guerrero, G.D., Martínez-del-Amor, M.A., Pérez-Hurtado, I., Pérez-Jiménez, M.J.: Implementing P systems parallelism by means of GPUs. In: Păun et al. [27], pp. 227–241
6. Cecilia, J.M., García, J.M., Guerrero, G.D., Martínez-del-Amor, M.A., Pérez-Hurtado, I., Pérez-Jiménez, M.J.: Simulating a P system based efficient solution to SAT by using GPUs. Journal of Logic and Algebraic Programming 79(6), 317–325 (2010)
7. Cecilia, J.M., García, J.M., Guerrero, G.D., Matínez-de-Amor, M.A., Pérez-Hurtado, I., Pérez-Jiménez, M.J.: Simulation of P systems with active membranes on CUDA. Briefings in Bioinformatics 11(3), 313–322 (2010)
8. Ceterchi, R., Gramatovici, R., Jonoska, N., Subramanian, K.G.: Tissue-like P systems with active membranes for picture generation. Fundamenta Informaticae 56(4), 311–328 (2003)
9. Ceterchi, R., Mutyam, M., Păun, Gh., Subramanian, K.G.: Array-rewriting P systems. Natural Computing 2(3), 229–249 (2003)
10. Chao, J., Nakayama, J.: Cubical singular simplex model for 3D objects and fast computation of homology groups. In: 13th International Conference on Pattern Recognition (ICPR'96). vol. IV, pp. 190–194. IEEE Computer Society, IEEE Computer Society, Los Alamitos, CA, USA (1996)
11. Christinal, H.A., Díaz-Pernil, D., Real, P.: Segmentation in 2D and 3D image using tissue-like P system. In: Bayro-Corrochano, E., Eklundh, J.O. (eds.) CIARP. Lecture Notes in Computer Science, vol. 5856, pp. 169–176. Springer (2009)
12. Ciobanu, G., Wenyuan, G.: P systems running on a cluster of computers. In: Martín-Vide et al. [19], pp. 123–139
13. Díaz-Pernil, D., Graciani, C., Gutiérrez-Naranjo, M.A., Pérez-Hurtado, I., Mario J. Pérez-Jiménez, M.: Software for P systems. In: Păun et al. [28], pp. 437–454
14. Díaz-Pernil, D., Gutiérrez-Naranjo, M.A., Molina-Abril, H., Real, P.: A bio-inspired software for segmenting digital images. In: Nagar, A.K., Thamburaj, R., Li, K., Tang, Z., Li, R. (eds.) Proceedings of the 2010 IEEE Fifth International Conference on Bio-Inspired Computing: Theories and Applications BIC-TA. vol. 2, pp. 1377 – 1381. IEEE Computer Society (2010)
15. Gershoni, R., Keinan, E., Păun, Gh., Piran, R., Ratner, T., Shoshani, S.: Research topics arising from the (planned) P systems implementation experiment in Technion. In: Díaz-Pernil, D., Graciani, C., Gutiérrez-Naranjo, M.A., Păun, Gh., Pérez-Hurtado, I., Riscos-Núñez, A. (eds.) Sixth Brainstorming Week on Membrane Computing. pp. 183–192. Fénix Editora, Sevilla, Spain (2008)
16. Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J., Riscos-Núñez, A.: Available membrane computing software. In: Ciobanu, G., Pérez-Jiménez, M.J., Păun, Gh. (eds.) Applications of Membrane Computing, pp. 411–436. Natural Computing Series, Springer (2006)

17. Kim, S.H., Kim, H.G., Tchah, K.H.: Object oriented face detection using colour transformation and range segmentation. Electronics Letters, IEEE 34, 979–980 (1998)
18. Kropatsch, W.G., Haxhimusa, Y., Ion, A.: Multiresolution image segmentations in graph pyramids. In: Kandel, A., Bunke, H., Last, M. (eds.) Applied Graph Theory in Computer Vision and Pattern Recognition, Studies in Computational Intelligence, vol. 52, pp. 3–41. Springer (2007)
19. Martín-Vide, C., Mauri, G., Păun, Gh., Rozenberg, G., Salomaa, A. (eds.): Membrane Computing, International Workshop, WMC 2003, Tarragona, Spain, July 17-22, 2003, Revised Papers, Lecture Notes in Computer Science, vol. 2933. Springer (2004)
20. Martín-Vide, C., Păun, Gh., Pazos, J., Rodríguez-Patón, A.: Tissue P systems. Theoretical Computer Science 296(2), 295–326 (2003)
21. Martín-Vide, C., Pazos, J., Păun, Gh., Rodríguez-Patón, A.: A new class of symbolic abstract neural nets: Tissue P systems. In: Ibarra, O.H., Zhang, L. (eds.) COCOON. Lecture Notes in Computer Science, vol. 2387, pp. 290–299. Springer (2002)
22. Nguyen, V., Kearney, D., Gioiosa, G.: Balancing performance, flexibility, and scalability in a parallel computing platform for membrane computing applications. In: Eleftherakis, G., Kefalas, P., Păun, Gh., Rozenberg, G., Salomaa, A. (eds.) Workshop on Membrane Computing. Lecture Notes in Computer Science, vol. 4860, pp. 385–413. Springer (2007)
23. Nguyen, V., Kearney, D., Gioiosa, G.: An algorithm for non-deterministic object distribution in p systems and its implementation in hardware. In: Corne, D.W., Frisco, P., Păun, Gh., Rozenberg, G., Salomaa, A. (eds.) Workshop on Membrane Computing. Lecture Notes in Computer Science, vol. 5391, pp. 325–354. Springer (2008)
24. Nguyen, V., Kearney, D., Gioiosa, G.: A region-oriented hardware implementation for membrane computing applications. In: Păun et al. [27], pp. 385–409
25. Nguyen, V., Kearney, D., Gioiosa, G.: An extensible, maintainable and elegant approach to hardware source code generation in reconfig-P. Journal of Logic and Algebraic Programming 79(6), 383–396 (2010)
26. Păun, Gh.: Membrane Computing. An Introduction. Springer-Verlag, Berlin, Germany (2002)
27. Păun, Gh., Pérez-Jiménez, M.J., Riscos-Núñez, A., Rozenberg, G., Salomaa, A. (eds.): Membrane Computing, 10th International Workshop, WMC 2009, Curtea de Arges, Romania, August 24-27, 2009. Revised Selected and Invited Papers, Lecture Notes in Computer Science, vol. 5957. Springer (2010)
28. Păun, Gh., Rozenberg, G., Salomaa, A. (eds.): The Oxford Handbook of Membrane Computing. Oxford University Press (2010)
29. Petreska, B., Teuscher, C.: A reconfigurable hardware membrane system. In: Martín-Vide et al. [19], pp. 269–285
30. Ritter, G.X., Wilson, J.N., Davidson, J.L.: Image algebra: An overview. Computer Vision, Graphics, and Image Processing 49(3), 297–331 (1990)
31. Shapiro, L.G., Stockman, G.C.: Computer Vision. Prentice Hall PTR, Upper Saddle River, NJ, USA (2001)
32. Syropoulos, A., Mamatas, L., Allilomes, P.C., Sotiriades, K.T.: A distributed simulation of transition P systems. In: Martín-Vide et al. [19], pp. 357–368
33. Tarabalka, Y., Chanussot, J., Benediktsson, J.A.: Segmentation and classification of hyperspectral images using Watershed transformation. Pattern Recognition 43(7), 2367–2379 (2010)

34. Tobias, O.J., Seara, R.: Image segmentation by histogram thresholding using fuzzy sets. IEEE Transactions on Image Processing 11(12), 1457–1465 (2002)
35. Trimberger, S.M.: Field-Programmable Gate Array Technology. Kluwer Academic Publishers, Norwell, MA, USA (1994)
36. Wang, D., Lu, H., Zhang, J., Liang, J.Z.: A knowledge-based fuzzy clustering method with adaptation penalty for bone segmentation of ct images. In: Proceedings of the 2005 IEEE Engineering in Medicine and Biology 27th Annual Conference. pp. 6488–6491 (2005)
37. Yazid, H., Arof, H.: Image segmentation using watershed transformation for facial expression recognition. In: IFMBE Proceedings, 4th Kuala Lumpur International Conference on Biomedical Engineering. pp. 575–578 (2008)
38. Yuan, X., Situ, N., Zouridakis, G.: A narrow band graph partitioning method for skin lesion segmentation. Pattern Recognition 42(6), 1017–1028 (2009)
39. NVIDIA Corporation. NVIDIA CUDA^{TM} Programming Guide. `http://www.nvidia.com/object/cuda_home_new.html`
40. P system web page. http://ppage.psystems.eu