Implementation on CUDA of the Smoothing Problem with Tissue-Like P Systems

Francisco Peña-Cantillana, University of Sevilla, Spain

Daniel Díaz-Pernil, University of Sevilla, Spain

Hepzibah A. Christinal, University of Sevilla, Spain, and Karunya University, India

Miguel A. Gutiérrez-Naranjo, University of Sevilla, Spain

ABSTRACT

Smoothing is often used in Digital Imagery for improving the quality of an image by reducing its level of noise. This paper presents a parallel implementation of an algorithm for smoothing 2D images in the framework of Membrane Computing. The chosen formal framework has been tissue-like P systems. The algorithm has been implemented by using a novel device architecture called CUDA^{TM} (Compute Unified Device Architecture) which allows the parallel NVIDIA Graphics Processors Units (GPUs) to solve many complex computational problems. Some examples are presented and compared; research lines for the future are also discussed.

Keywords: *Compute Unified Device Architecture (CUDA), Graphics Processors Units (GPU), Image Processing, Membrane Computing, Smoothing, Tissue-Like P Systems*

1. INTRODUCTION

The study of digital images (Shapiro & Stockman, 2001) has seen a large progress over the last decades. The aim of dealing with an image in its digital form is improving its quality, in some sense, or simply achieving some artistic effect. The physical properties of camera technology are inherently linked to different sources of noise, so the application of a smoothing algorithm is necessary for reducing such noise within an image.

In this paper we use Membrane Computing techniques for smoothing 2D images with tissue-like P systems. We refer to Păun (2002) for basic information in this area and to Păun, Rozenberg, and Salomaa (2010) for a comprehensive presentation and the web site http://ppage.psystems.eu for the up-to-date information. The algorithm has been implemented by using a novel device architecture called CUDA (Compute Unified Device Architecture, http://www.nvidia.com/object/cuda_home_new.html). CUDA^{TM} is a general purpose parallel computing architecture that allows the parallel NVIDIA Graphics Processors Units (GPUs) to

DOI: 10.4018/jncr.2011070103

solve many complex computational problems (for a good overview, the reader can refer to Owens et al., 2008) in a more efficient way than on a CPU. This architecture has been previously used in Membrane Computing (Cecilia et al., 2010a, 2010b) but, to the best of our knowledge, this is the first time that it is used for implementing a smoothing algorithm.

Dealing with Digital Images has several features which make it suitable for techniques inspired by nature. One of them is that it can be parallelized and locally solved. Regardless how large the picture is, the smoothing process can be performed in parallel in different local areas. 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 Ceterchi et al. (2003) and Ceterchi, Mutyam, Păun, and Subramanian (2003) or the work by Chao and Nakayama where Natural Computing and Algebraic Topology are linked by using Neural Networks (Chao & Nakayama, 1996). Recently, new approaches have been presented in the framework of Membrane Computing (Christinal, Díaz-Pernil, Gutiérrez-Naranjo, & Pérez-Jiménez, 2010; Díaz-Pernil, Gutiérrez-Naranjo, Molina-Abril, & Real, 2010). Christinal, Díaz-Pernil, and Real (2009a, 2009b, 2010) started a new bio-inspired research line where the power and efficiency of tissue-like P systems were applied to topological processes for 2D and 3D digital images.

The paper is organised as follows: Firstly, we recall some basics of tissue-like P systems and the foundations of Digital Imagery. Next we present our P systems family and a simple example showing different results by using different thresholds. In Section 3 we present the implementation in CUDA™ of the algorithm and show an illustrative example, including a comparison of the times obtained in the different variants. The paper finishes with some final remarks and hints for future work.

2. PRELIMINARIES

In this section we provide some basics on the used P system model, tissue-like P systems, and on the foundation of Digital Imagery.

Tissue-like P systems (Martín-Vide, Păun, Pazos, & Rodríguez-Patón, 2003) have two biological inspirations: 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.

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

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

where

1. Γ is a finite *alphabet*, whose symbols will be called *objects*;
2. $\Sigma (\subset \Gamma)$ is the input alphabet;
3. $E \subseteq \Gamma$ is the alphabet of 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. R is a finite set of communication rules of the 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 cell.

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 cell and o_Π denotes the output cell (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 cells of the P system. We interpret that $E \subseteq \Gamma$ 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 *computation* is a sequence of computation steps such that either it is infinite or it is finite and the last step yields a halting configuration (i.e., no rules can be applied to it). Then, a computation halts when the system reaches a halting configuration.

Next we recall some basics on Digital Imagery (we refer the interested reader to Ritter, Wilson, & Davidson, 1990 for a detailed introduction). A *point set* is simply a topological space consisting of a collection of objects called points and a topology which provides notions as *nearness* of two points, the *connectivity* of a subset of the point set, the *neighbourhood* of a point, *boundary points*, and *curves* and *arcs*.

For a point set $X \subseteq Z$, a *neighbourhood 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 *neighbourhood* for x.

There are two neighbourhood function on subsets of Z^2 which are of particular importance in image processing, the *von Neumann* neighbourhood and the *Moore* neighbourhood. The first one $N: X \to 2^{Z^2}$ 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 $x=(x_1, x_2) \in X \subset Z^2$. While the Moore neighbourhood $M: X \to 2^{Z^2}$ is defined by $M(x)=\{ y=(x_1 \pm j, x_2 \pm k),\ j,k \in \{0,1\} \}$, where $x=(x_1, x_2) \in X \subset Z^2$. The von Neumann and Moore neighbourhood are also called the *four neighbourhood* (4-adjacency) and *eight neighbourhood* (8-adjacency), respectively.

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. Usually, we consider an ordered set $C \subseteq Z$ called the *set of colours*.

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 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.

The purpose of image enhancement is to improve the visual appearance of an image, or to transform an image into a form that is better suited for human interpretation or machine analysis. There exists a multitude of image enhancement techniques, as are averaging of multiple images, local averaging, Gaussian smoothing, max-min sharpening transform, etc.

One of the forms to enhancement an image could be eliminate non important regions of an image, i.e., remove regions which do not provide relevant information. This technique is known as smoothing.

2.1. A Family of Tissue-Like P Systems

Given an image with n^2 pixels ($n \in N$) and r a threshold ($r \in N$) which represents the upper bound of the distance between related colours, we define a tissue-like P system whose input is given by the pixels of the image encoded by the objects a_{ij}, where $1 \leq i,j \leq n$ and $a \in C$. Next, we shall give some outlines how to prove that our *smoothing problem* can be solved in a logarithmic number of steps using a family of tissue-like P systems Π.

We define a family of tissue-like P systems to do a smoothing of a 2D image. For each image of size n^2 with $n \in N$, we consider the tissue-like P system with input of degree 1:

$\prod(\mathrm{r,n}) = (\Gamma, \Sigma, E, \mathrm{w}_1, R, i_{\Pi}, o_{\Pi})$

where

- $\Gamma = \Sigma \cup E$,
- $\Sigma = \{\mathrm{a}_{\mathrm{ij}} : \mathrm{a} \in C, 1 \leq i,j \leq n\}$,
- $E = \{\mathrm{a}_{\mathrm{ij}} : \mathrm{a} \in C, 1 \leq i,j \leq n\}$,
- $w_1 = \varnothing$
- R is the following set of communication rules:
- $(1, \mathrm{a}_{\mathrm{ij}}\ \mathrm{b}_{\mathrm{kl}} / \mathrm{a}_{\mathrm{ij}}\ \mathrm{a}_{\mathrm{kl}}, 0)$,
- for $1 \leq i,j \leq n$, $a,b \in C$, $a<b$ and $d(a,b) \leq r$,

These rules are used to simplify the image. If we have two colours whose distance in the set of colours of the image is lower than a given threshold r, then the colour with a higher value is replaced by a lower one. Of this manner, we change the regions structure.

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

Each P system works as follows: We take pairs of adjacent pixels and change the col or of the pixel with lower col or. We do it in a parallel way with all the possible pairs of pixels. In the next step, we will repeat the previous process, but the colours of the pixels may have been changed. So, in the worst case, a linear number of steps are necessary to do all the possible changes to obtain a smoothing of our image.

2.2. A Simple Example

In this section, we show the results obtained by the application of our method. Our input image (of size 30×30) can be seen in Figure 1. In this case, 0 represents the white colour and 30 represents the black colour.

Working with different thresholds provides different results as we can observe in Figure 2. If we take r=5, then we get the first image, and when the threshold is r=10 the second image is obtained. By using this method, it is clear that the structure of regions is (more or less) conserved with a threshold of r=5 and we need to a high number to obtain a more simplified image. Moreover, we can observe in both cases the col or of regions is similar to the regions of input image in this method. Therefore, we can use this technique for smoothing images, and clarify the structure of a region without eliminate important information.

Bearing in mind the size of the input data is $O(n^2)$, representing by h the number of the elements of the set C and considering r as the threshold used with both membrane solutions, the amount of necessary resources for defining the P systems of our family and the complexity of our problem is determined in Table 1.

3. PARALLEL IMPLEMENTATION

GPUs constitute nowadays a solid alternative for high performance computing, and the advent of CUDA™ allows programmers a friendly model to accelerate a broad range of applications. GPUs are especially well-suited to address problems that can be expressed as data-parallel computations. GPUs can support several thousand of concurrent threads providing a massively parallel environment. This parallel

Figure 1. An example

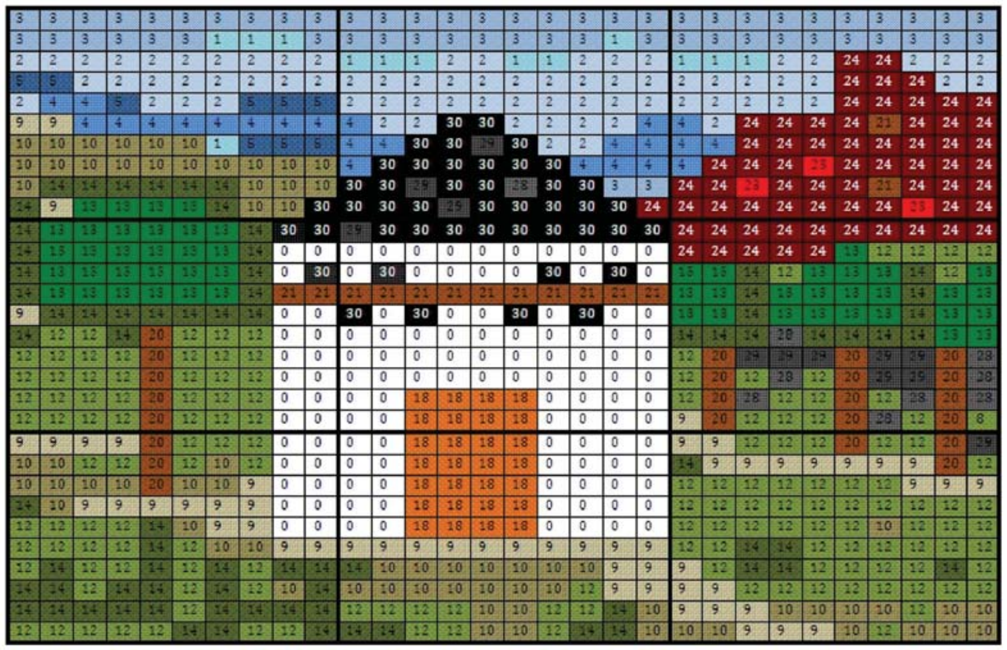
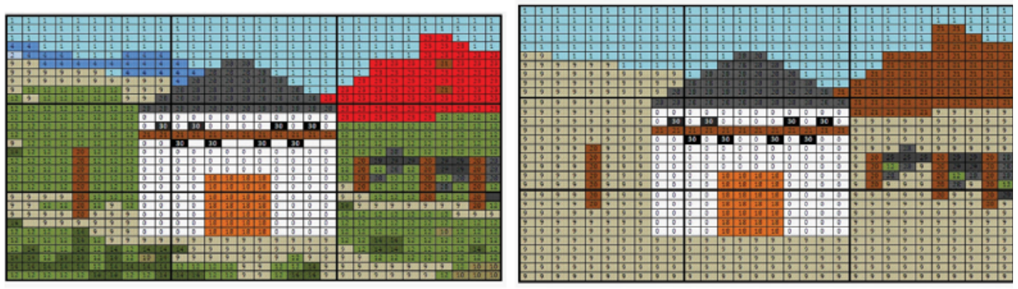


Figure 2. Theoretical smoothness of Figure 1



technology is suitable for parallel computational paradigms by providing an efficient framework for real parallel implementations.

In this paper, we present a parallel software tool which implements our membrane approach for smoothing images. It has been developed by using Microsoft Visual Studio 2008 Professional Edition (C++) with the plugging Parallel Nsight (CUDA^{TM}) under Microsoft Windows 7 Professional with 32 bits.

To implement the P systems, CUDA^{TM} C, an extension of C for implementations of executable kernels in parallel with graphical cards NVIDIA has been used. It has been necessary the *nvcc compiler* of CUDA^{TM} Toolkit. Moreover, we use libraries from openCV to the treatment of input and output images. Microsoft Visual Studio 2008 is responsible for calling to the compilers to build the objects, and to link them with the final program. This allows us to

Table 1. Complexity and resources

Smoothing Problem	
Complexity	Dynamical
Number of steps of a computation	$O(n)$
Necessary Resources	
Size of the alphabet	$n^2 h$
Initial number of cells	1
Initial number of objects	0
Number of rules	$O(n^2 h)$
Upper bound for the length of the rules	4

deal with images stored in .BMP, .JPG, .PNG, or .TIF formats among others.

The experiments have been performed on a computer with a CPU Intel Pentium 4 650, with support for HT technology which allows to work like two CPUs of 32 bits to 3412 MHz. It has 2 MB of L2 cache memory and 1 GB DDR SDRAM of main memory with 64 bits bus wide to 200 MHz. Moreover, it has a hard disc of 160 GB SATA2 with a transfer rate of 300 Mbps in a 8 MB buffer. The graphical card (GPU) is an NVIDIA Geforce 8600 GT composed by 4 *Stream Processors* with a total of 32 cores to 1300 MHz and executes 512 threads per block as maximum. It has a 512 MB DDR2 main memory, but 499 MB could be used by processing in a 128 bits bus to 700 MHz. So, the transfer rate obtained is by 22.4 Gbps. For constant memory used 64 KB and for shared memory 16 KB. Its Compute Capability is 1.1 (from 1.0 to 2.1), then we can obtain a lot of improvements in the efficiency of the algorithms.

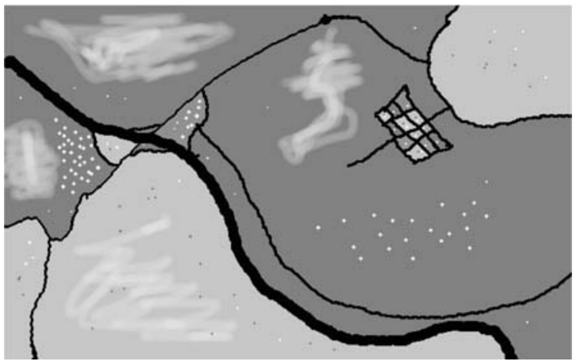
We have developed two applications of our P systems. In this case, we consider the natural order in the gray-scale set of colours $C = \{0,\ldots,255\}$. In the first one, we have considered a deterministic implementation, where the Moore neighbourhood is considered. The system checks if the rules can be applied for eight adjacent pixels. In the second one, we have considered a random selection of an adjacent pixel to work. In this case, the system checks only one possibility randomly chosen. This way, we simulate the characteristic non determinism of P systems using randomness. Moreover, we have decided to stop the system before the halting configuration, because more than an appropriate number of parallel steps of processing could make too much uniform the output image. In fact, in the deterministic version, the process could finish before the pre-fixed number of steps. So, the system needs some time to check this possibility. In the second one, it is not necessary to look at this question.

We consider the image of size 640×400 in Figure 3. When we take the deterministic application of our software, we can check that if we use an threshold r=50, our software smooths the original image (Figure 4) using 44 parallel steps. Nonetheless, when we work with a higher threshold, new important regions are changed, and the output image is different (Figure 5).

When we consider the random version of our software, we can check that if we use an threshold r=50 we need 300 steps, but the differences with the resulting image with 150 or 200 steps are minimum, as we can see in Figure 6. When we take higher thresholds, as in the Figure 7, we can check that new regions have different colours.

Table 2 shows the running times of our software for both cases with different thresholds, shown in the examples. We can observe that the deterministic version of our software needs less time with respect to the random version.

Figure 3. Original image



In the first one, it applies eight rules for each pixels while, in the second one, it applies only one rule for each pixel. Moreover, we need an additional running time to implement the random in each step for each pixel.

Finally, we have done some experiments with our software to know what happened if we work with images of different size. We have checked our software with images until size 512×512 in both versions. The deterministic version needs much time with bigger images, and the random version does not work with those images. This is a physical problem with our graphical card, because the shared memory is small.

Figure 4. Deterministic version. Threshold 50: 0, 5, 10, 20, 32 and 44 steps, respectively.

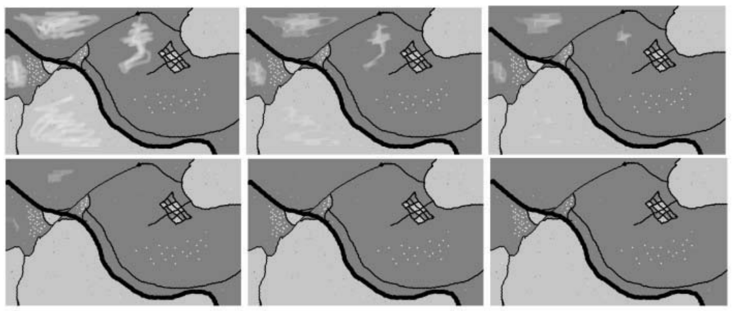


Figure 5. Deterministic version. Left image: Threshold 75, step 193. Right image: Threshold 125, step 193.

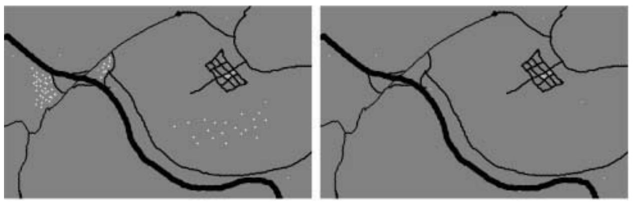


Figure 6. Random version. Threshold 50: 0, 50, 100, 150, 200 and 300 steps, respectively.

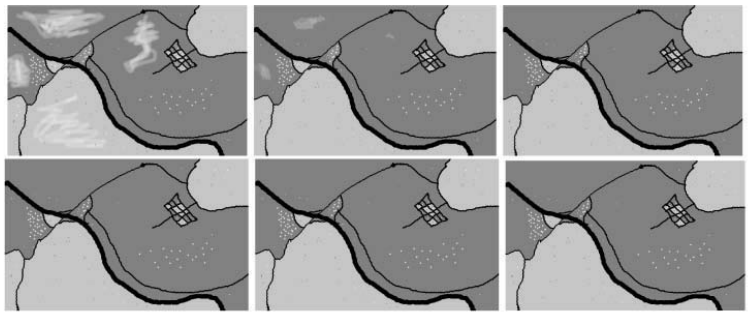


Figure 7. Random version. Left image: Threshold 75, step 1000. Right image: Threshold 125, step 800.

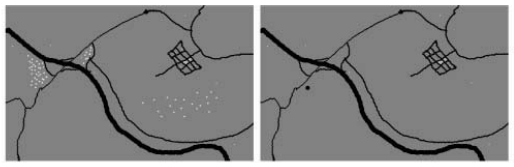


Table 2. Running times of the experiments

Version ∖ Thresholds	**Computation steps**	**Running Time**
Determ. ∖ 50	44	536.522 ms
Determ. ∖ 75	193	1582.098 ms
Determ. ∖ 125	193	1563.660 ms
Random ∖ 50	300	6823.683 ms
Random ∖ 75	1000	21537.701 ms
Random ∖ 100	800	17332.891 ms

4. CONCLUSION AND FUTURE WORK

In this paper, three emergent research fields are put together. Firstly, as pointed in Christinal et al. (2009a) and Christinal, Díaz-Pernil, and Real (2010), Membrane Computing has features as the encapsulation of the information, a simple representation of the knowledge and parallelism, which are appropriate with dealing with digital images. Nonetheless, the use of the intrinsic parallelism of Membrane Computing techniques cannot be implemented in current one-processor computers, so the potential advantages of the theoretical design are lost.

In this paper we show that the drawback of using one-processor computers for implementing Membrane Computing designs can be avoided by using the parallel architecture CUDA^{TM}. This new technology provides the hardware needed for a real parallel implementation of Membrane Computing algorithms.

Considering this paper as a starting point, several research lines are open: From Digital

Imagery, new parallel algorithms can be proposed or adapted to the new technology, from the Membrane Computing side, new design or different P system models can be explored. From the hardware point of view, the advances in the new technology CUDA™ open new possibilities for going on with the research.

ACKNOWLEDGMENTS

DDP and MAGN acknowledge the support of the projects TIN2008-04487-E and TIN-2009-13192 of the Ministerio de Ciencia e Innovación of Spain and the support of the Project of Excellence with *Investigador de Reconocida Valía* of the Junta de Andalucía, grant P08-TIC-04200. HC acknowledges the support of the project MTM2009-12716 of the Ministerio de Educación y Ciencia of Spain and the project PO6-TIC-02268 of Excellence of Junta de Andalucìa, and the "CATAM" PAICYT research group FQM-296.

REFERENCES

Cecilia, J. M., García, J. M., Guerrero, G. D., Martínez-del-Amor, M. A., Pérez-Hurtado, I., & Pérez-Jiménez, M. J. (2010a). Simulating a P system based efficient solution to SAT by using GPUs. *Journal of Logic and Algebraic Programming*, *79*(6), 317–325. doi:10.1016/j.jlap.2010.03.008

Cecilia, J. M., García, J. M., Guerrero, G. D., Martìnez-del-Amor, M. A., Pérez-Hurtado, I., & Pérez-Jiménez, M. J. (2010b). Simulation of P systems with active membranes on CUDA. *Briefings in Bioinformatics*, *11*(3), 313–322. doi:10.1093/bib/bbp064

Ceterchi, R., Gramatovici, R., Jonoska, N., & Subramanian, K. G. (2003). Tissue-like P systems with active membranes for picture generation. *Fundamenta Informaticae*, *56*(4), 311–328.

Ceterchi, R., Mutyam, M., Păun, G., & Subramanian, K. G. (2003). Array-rewriting P systems. *Natural Computing*, *2*(3), 229–249. doi:10.1023/A:1025497107681

Chao, J., & Nakayama, J. (1996). Cubical singular simplex model for 3D objects and fast computation of homology groups. In *Proceedings of the 13th International Conference on Pattern Recognition* (Vol. 4, pp. 190-194).

Christinal, H. A., Díaz-Pernil, D., Gutiérrez-Naranjo, M. A., & Pérez-Jiménez, M. J. (2010). Thresholding of 2D images with cell-like P systems. *Romanian Journal of Information Science and Technology*, *13*(2), 131–140.

Christinal, H. A., Díaz-Pernil, D., & Real, P. (2009a, November 15-18). Segmentation in 2D and 3D image using tissue-like P system. In E. Bayro-Corrochano & J. O. Eklundh (Eds.), *Proceedings of the 14th Ibero-American Conference on Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*, Guadalajara, Jalisco, Mexico (LNCS 5856, pp. 169-176).

Christinal, H. A., Díaz-Pernil, D., & Real, P. (2009b, November 24-27). Using membrane computing for obtaining homology groups of binary 2D digital images. In P. Wiederhold & R. P. Barneva (Eds.), *Proceedings of the 13th International Workshop on Combinatorial Image Analysis*, Playa del Carmen, Mexico (LNCS 5852, pp. 383-396).

Christinal, H. A., Díaz-Pernil, D., & Real, P. (2010). P systems and computational algebraic topology. *Journal of Mathematical and Computer Modelling*, *52*(11-12), 1982–1996. doi:10.1016/j.mcm.2010.06.001

Díaz-Pernil, D., Gutiérrez-Naranjo, M. A., Molina-Abril, H., & Real, P. (2010). A bio-inspired software for segmenting digital images. In A. K. Nagar, R. Thamburaj, K. Li, Z. Tang, & R. Li (Eds.), *Proceedings of the IEEE Fifth International Conference on Bio-inspired Computing: Theories and Applications*, Beijing, China (Vol. 2, p. 1377-1381).

Martín-Vide, C., Păun, G., Pazos, J., & Rodríguez-Patón, A. (2003). Tissue P systems. *Theoretical Computer Science*, *296*(2), 295–326. doi:10.1016/S0304-3975(02)00659-X

Owens, J. D., Houston, M., Luebke, D., Green, S., Stone, J. E., & Phillips, J. C. (2008). GPU computing. *Proceedings of the IEEE*, *96*(5), 879–899. doi:10.1109/JPROC.2008.917757

Păun, G. (2002). *Membrane computing: An introduction*. Berlin, Germany: Springer-Verlag.

Păun, G., Rozenberg, G., & Salomaa, A. (Eds.). (2010). *The Oxford handbook of membrane computing*. Oxford, UK: Oxford University Press.

Ritter, G. X., Wilson, J. N., & Davidson, J. L. (1990). Image algebra: An overview. *Computer Vision Graphics and Image Processing*, *49*(3), 297–331. doi:10.1016/0734-189X(90)90106-6

Shapiro, L. G., & Stockman, G. C. (2001). *Computer vision*. Upper Saddle River, NJ: Prentice Hall.

Francisco Peña-Cantillana studies Computer Engineering at the University of Seville, Spain. He is also a member of the Research Group on Natural Computing of the University of Seville. Currently, he is making his Final Project on the fields of Natural Computing, Digital Image and GPGPU using CUDA.

Daniel Diaz-Pernil obtained his PhD in Mathematics in 2008. Currently, he is a professor in the Applied Mathematic I Department at the University of Seville, Spain. He is also a member of the Research Group on Computational Algebraic Topology and Applied Mathematics of the University of Seville. His research interest includes topics related to Algebraic Topologya, Digital Image and Natural Computing.

Hepzibah A. Christinal is doing her PhD in Mathematics in the Applied Mathematics Department at the University of Seville, Spain. She is currently working as Assistant Professor (SG) in Karunya University, Coimbatore, Tamilnadu, India. She is also a member of the Research Group on Computational Algebraic Topology and Applied Mathematics of the University of Seville. Her research interest includes Algebraic Topology, Digital image, Natural Computing and Optimization Techniques.

Miguel A. Gutiérrez-Naranjo. He obtained his PhD in Mathematics in 2002. Currently, he is a professor in the Computer Science and Artificial Intelligence Department at the University of Seville, Spain. He is also a member of the Research Group on Natural Computing of the University of Seville. His research interest includes topics related to Artificial Intelligence and Natural Computing. He has coauthored more than 30 scientific papers in these areas.