A Parallel Implementation of the Thresholding Problem by Using Tissue-Like P Systems

Francisco Peña-Cantillana[1], Daniel Díaz-Pernil[2],
Ainhoa Berciano[2,3], and Miguel Angel Gutiérrez-Naranjo[1]

[1] Research Group on Natural Computing - Dept. of Computer Science and AI
University of Seville, Spain
[2] CATAM Research Group - Dept. of Applied Mathematics I
University of Seville, Spain
[3] Departament of Didactic of Mathematics and Experimental Sciences
University of the Basque Country, Spain
frapencan@gmail.com, {sbdani,magutier}@us.es, ainhoa.berciano@ehu.es

Abstract. In this paper we present a parallel algorithm to solve the thresholding problem by using Membrane Computing techniques. This bio-inspired algorithm has been implemented in a novel device architecture called CUDA™, (Compute Unified Device Architecture). We present some examples, compare the obtained time and present some research lines for the future.

1 Introduction

In Computer Vision [13], segmentation is the process of splitting a digital image into sets of pixels in order to make it simpler and easier to analyze. One of its main uses is the localization of objects and boundaries. Technically, the process consists of assigning a label to each pixel, in such way the pixels with the same label form a meaningful region. *Thresholding* is one of the simplest and most widely used image segmentation techniques.

In this paper, we present a bio-inspired algorithm to solve the thresholding problem. The treatment of digital images has several features which make it suitable for techniques inspired by nature. One of them is that the treatment of the image can be parallelized and locally solved. Another interesting feature is that the basic necessary information can be easily encoded by bio-inspired representations. Throughout this paper, we use techniques taken from Membrane Computing. This is a theoretical model of computation inspired by the structure and functioning of cells as living organisms able to process and generate information. The computational devices in Membrane Computing are called *P systems* [12,15]. Roughly speaking, a P system consists of a membrane structure, in whose compartments one places multisets of objects which evolve according to given rules which are usually applied in a synchronous non-deterministic maximally parallel manner. In particular, we consider antiport rules, which were introduced as communication rules for P systems in [11]. Such rules are inspired

on the communication among cells. In the antiport rules, objects residing at both sides of the membrane cross it simultaneously in opposite directions.

In the literature, one can find several attempts for bridging problems from Digital Imagery with Natural Computing as the work by K.G. Subramanian *et al.* [1] or the work by Chao and Nakayama where Natural Computing and Algebraic Topology are linked by using Neural Networks [2] (extended Kohonen mapping). Recently, new approaches have been presented in the framework of Membrane Computing [3,4,5].

The algorithm has been implemented by using a novel device architecture called CUDA™, (Compute Unified Device Architecture) [9,10,14]. CUDA™ is a general purpose parallel computing architecture that allows the parallel NVIDIA Graphics Processors Units (GPUs) to solve many complex computational problems in a more efficient way than on a CPU.

The paper is organised as follows: Firstly, we briefly recall some basics on the theoretical Membrane Computing framework and, in Section 2.1, the family of P systems which solves the thresholding problem is presented. Next, we present our parallel implementation, with several examples and comparisons. The paper finishes with some final remarks.

2 Methods

There are different models of P systems in the Membrane Computing framework. In this paper we will consider the so-called *tissue-like P systems* [8]. They 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

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

where

1. Γ is a finite *alphabet*, whose symbols will be called *objects*, $\Sigma (\subset \Gamma)$ is the input alphabet, $\mathcal{E} \subseteq \Gamma$ is the alphabet of the objects in the environment,
2. $w_1, \ldots, w_q$ are strings over Γ representing the multisets of objects associated with the cells at the initial configuration,
3. $\mathcal{R}$ is a finite set of communication rules of the following form: $(i, u/v, j)$ for $i, j \in \{0, 1, 2, \ldots, q\}, i \neq j$, $u, v \in \Gamma^*$,
4. $i_\Pi \in \{1, 2, \ldots, q\}$ is the input cell and $o_\Pi \in \{0, 1, 2, \ldots, q\}$ 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 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 system at the initial configuration. 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.

In what follows we assume the reader is already familiar with the basic notions and the terminology underlying P systems [12].

2.1 Thresholding of Digital Images

Thresholding is a method of image segmentation whose basic aim is to obtain a binary image from a color one. The idea is to split the set of pixels into two sets (black and white) depending on its bright and a fixed value, the *threshold*. If the bright of the pixel is greater than the threshold, then the pixel is labeled as *object*. Otherwise, it is labeled as *background*. After the labeling, a new binary image is created by coloring each pixel white or black.

The basic thresholding method can be generalized in a natural way to *quantization*. Instead of using $\{0, 1\}$ as labels to obtain a binary image, we can consider a larger set of labels, $\{1, \ldots, k\}$ and get a final image with k levels. Another natural generalization is to replace the color information by another scale on the features of the pixel (bright, intensity, gray scale, etc.). In this section, we present a family of tissue-like P systems which solves the thresholding problem by considering a 4-neighborhood between pixels.

Let $\mathcal{C}$ be the alphabet of colors. The key idea is to divide $\mathcal{C}$ into classes and to assign a representative of each class. After this choosing, the color of each pixel is changed by the representatives of their class. In this paper, we choose the first color of each class as representatives of them.

2.2 A Family of Tissue-Like P Systems

Given a digital image I with n^2 pixels and $n \in \mathbb{N}$, 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 $a \in \mathcal{C}$. For each image of size n^2 and k the number of intervals in which the set of colors $\mathcal{C}$ is divided, $(k, m, n \in \mathbb{N}$, with $n = k \times m)$, we consider the following tissue-like P system with input of degree 1, $\mathbf{\Pi}(k, n) = (\Gamma, \Sigma, \mathcal{E}, w_1, \mathcal{R}, i_\Pi, o_\Pi)$, where

- $\Gamma = \Sigma = \mathcal{E} = \{a_{ij} : a \in \mathcal{C},\ 1 \leq i, j \leq n\}$,
- $w_1 = \emptyset$,

– R is the following set of communication rules:
  $(1, b_{ij}/a_{ij}, 0)$, for $1 \leq i, j \leq n$; $l = 0, 1, 2, \ldots, n - m$, $a = m \cdot l$ and $b \in \mathcal{C}$, $a < b \leq a + (m - 1)$.
  These rules are used to divide the set of colors in k intervals of length m.
– $i_\Pi = o_\Pi = 1$.

Next, we will give some outlines how to prove that the *thresholding problem* can be solved in one step using this family of tissue-like P systems $\mathbf{\Pi}(k, n)$: the alphabet of colors is split into k intervals and the first element of each interval is chosen as a representative. The objects representing pixels are placed in the membrane labeled by 1 and the representatives are placed in the environment. The rules are applied in parallel and simultaneously each pixel is traded against the corresponding representative.

The question is how we can decide the number of intervals dividing $\mathcal{C}$. We could divide the alphabet in two intervals to obtain a *binary image*. If we divide a color image in 255 intervals then we will obtain a *grey scale image*. Other possibility is, given a digital image I, to obtain the medium color of I (μ_I).

3 Results

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. The way GPUs exploit parallelism differ from multi-core CPUs, which raises new challenges to take advantage of its tremendous computing power. In this paper, we present a parallel software tool based on our membrane solution for image quantization and binarization. It has been developed by using Microsoft Visual Studio 2008 Professional Edition (C++) with the plugging Parallel Nsight (CUDA™) under Microsoft Windows 7 Professional with 32 bits. CUDA™ C, an extension of C for implementations of executable kernels in parallel with graphical cards NVIDIA has been used to implement the P systems. It has been necessary the *nvcc compiler* of CUDA™ Toolkit and some libraries from openCV to the treatment of input and output images.

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. The computer has 2 MB of L2 cache memory and 1 GB DDR SDRAM of main memory with 64 bits bus wide to 200 MHz.

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 (It is not a good data for a good CUDA™ graphical card). 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.

Fig. 1. Quantization with classes of same size

3.1 Examples

Although our algorithm can work with color images, our software is implemented to work with grey scale. In this section, we show the results obtained by our tool to do a quantization of images. We consider an image of size 468×351 (see left up image of Fig. 1). Notice that, when we work with different thresholds we obtain different results, as the rest of images of Fig. 1 show.

The simplest thresholding method is binarization where the image is divided into two set of pixels. We use our tool to do a binarization of images with grey scale, but we need to choice a threshold to divide the image into two sets of different sizes. In Fig. 2 we show three original images (see up) with sizes 3456×2592, 1536×2048 and 960×1280, respectively. As we can see, only the third image has been binarized in an appropriate way. In fact, we can perfectly distinguish the climber in the resulting image in our software. But, when we see the first two images, it is clear that they are not good binarizations. In the first one, when the binarization is done, we keep one of the fishes, but the figure of the bird is partially lost with the background of the image and, in the second one, when the binarization is done, we see, for example, the problems with the legs of the dog. The processing time for each image of Fig. 2 are 363.161 ms, 307.558 ms and 243.461 ms, respectively.

Then, the question is to select an appropriate threshold to the binarization. So, we take the Hamadani algorithm [6] for the chosen thresholds and implement this in our tool (in a sequential way). In this case we consider a linear combination of

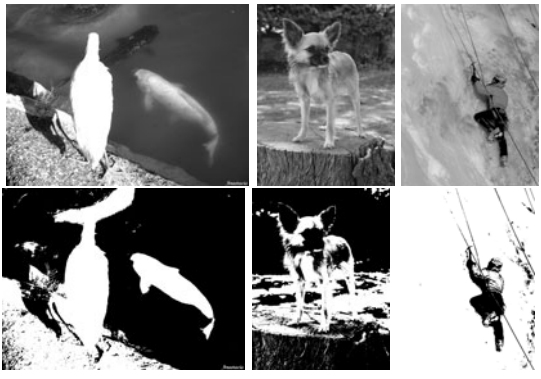

Fig. 2. Binarization of three images using as threshold k=80, 100 and 115, respectively

Fig. 3. Binarization using the Hamadani Algorithm. Thresholds are 172 and 155 to each image, respectively.

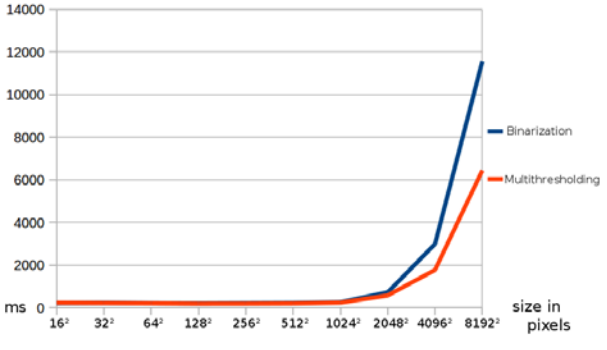

Fig. 4. Comparative

the mean, μ, and the standard deviation, σ, of the values of pixels of the images: $k = k_1 \cdot \mu + k_2 \cdot \sigma$. So, the question is the chosen of the parameters k_1 and k_2. For example, we have taken $k_1 = k_2 = 1$ in the two images with problems in the binarization, and we can see in Fig. 3 how the binarizations are better with this chosen. We need 619.889 ms and 346.034 ms to each binarization, respectively.

Finally, we have some proofs with our software to check the needed running time to process images of different sizes. We show the results in the Fig. 4. Obviously, we need more time to the binarization because we have added the (sequential) implementation of the Hamadani algorithm. Moreover, we should advise, our tool needs much more time to very weighted images because our graphical card is not professional, for example, the shared memory is very small.

4 Conclusions

The bio-inspired computing techniques have 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 these paradigms can hardly 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™. This new technology provides the hardware needed for a real parallel implementation of Membrane Computing algorithms.

By following this research line, several questions are open: In this paper, the Hamadani algorithm has been implemented in sequential mode. In order to exploit all the possibilities of the parallel architecture, a new parallel implementation should be designed.

¿From Digital Imagery, new parallel algorithms (for example, the Otsu algorithm [7] for multilevel thresholding), can be 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™ with the new boards Tesla and Fermi open new possibilities for going on with the research.

Acknowledgements. 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 Andalucía, the "CATAM" PAICYT research group FQM-296 and EHU09/04 project of University of the Basque Country.

References

1. 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)

2. 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 1996), vol. IV, pp. 190–194. IEEE Computer Society, Los Alamitos (1996)
3. Christinal, H.A., Díaz-Pernil, D., Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J.: Thresholding of 2D images with cell-like P systems. Romanian Journal of Information Science and Technology (ROMJIST) 13(2), 131–140 (2010)
4. 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 2009. LNCS, vol. 5856, pp. 169–176. Springer, Heidelberg (2009)
5. 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, Los Alamitos (2010)
6. Hamadani, N.: Automatic target cueing in IR imagery. Master's thesis, Air Force Institute of Technology, WAFP (December 1981)
7. Liao, P.S., Chen, T.S., Chung, P.C.: A fast algorithm for multilevel thresholding. Journal of Information Scence and Engineering 17(5), 713–727 (2001)
8. Martín-Vide, C., Păun, G., Pazos, J., Rodríguez-Patón, A.: Tissue P systems. Theoretical Computer Science 296(2), 295–326 (2003)
9. Nickolls, J., Buck, I., Garland, M., Skadron, K.: Scalable parallel programming with cuda. Queue 6, 40–53 (2008)
10. Owens, J.D., Houston, M., Luebke, D., Green, S., Stone, J.E., Phillips, J.C.: GPU Computing. Proceedings of the IEEE 96(5), 879–899 (2008)
11. Păun, A., Păun, G.: The power of communication: P systems with symport/antiport. New Generation Computing 20(3), 295–306 (2002)
12. Păun, G., Rozenberg, G., Salomaa, A. (eds.): The Oxford Handbook of Membrane Computing. Oxford University Press, Oxford (2010)
13. Shapiro, L.G., Stockman, G.C.: Computer Vision. Prentice Hall PTR, Upper Saddle River (2001)
14. NVIDIA Corporation. NVIDIA CUDA™ Programming Guide, `http://www.nvidia.com/object/cuda_home_new.html`
15. P system web page, `http://ppage.psystems.eu`