International Journal of Modern Physics C, Vol. 0, No. 0 (1992) 000–000

CELLULAR AUTOMATA AS A COMPUTATIONAL MODEL FOR LOW-LEVEL VISION

ALBERTO BROGGI, VINCENZO D'ANDREA and GIULIO DESTRI
Dipartimento di Ingegneria dell'Informazione
Università degli Studi di Parma
Viale delle Scienze
Parma, I-43100, Italy

In this paper we discuss the use of the Cellular Automata (CA) computational model in computer vision applications on massively parallel architectures. Motivations and guidelines of this approach to low-level vision in the frame of the PROMETHEUS project are discussed. The hard real-time requirement of actual application can be only satisfied using an ad hoc VLSI massively parallel architecture (PAPRICA). The hardware solutions and the specific algorithms can be efficiently verified and tested only using, as a simulator, a general purpose machine with a parent architecture (CM-2). An example of application related to feature extraction is discussed.

1. Introduction

The PROMETHEUS[1] project goal of increasing traffic safety can be obtained increasing the quality of information provided to the car driver. To this end, a Computer Vision system plays an important role, since most of the information available when driving has visual nature. Such a vision system, integrated into a "smart" sensor, should be able to provide the driver many different kinds of information. The information could range, to give only a few examples, from the detection of obstacles on the car's path and of approaching crossing cars, to the possibility of a safe overtaking.

All these tasks should be accomplished in real time, because the limit to the maximum operating speed is the response time of the system. An approach based on massively parallel computers can be successful for many of the algorithms used, providing very fast processing for data parallel algorithms.[2]

A prominent point in this case is the cost of such a smart sensor. It should be comparable to the cost of other car instruments, rather than to the cost of the car. Both very fast serial computers and commercially available massively parallel systems exceed this cost constraint, even though hardware price decreases every year.

These considerations lead to consider the use of PAPRICA for Image Processing. PAPRICA, a Cellular Automata based machine developed at the Politecnico di Torino,[3,4] is potentially capable of performing, in real time, many tasks relevant to image analysis. The architecture of PAPRICA can be efficiently simulated on the Connection Machine, and such a simulator was used for developing and testing algorithms and programming tools for PAPRICA.

The next chapter will introduce the basics of Cellular Automata, with a special regard to the application of this computational paradigm to Image Processing. PAPRICA programming environment and the simulator on the Connection Machine, are described in chapter 3. Finally, chapter 4 will sketch an example of feature extraction, for road images, using PAPRICA.

2. Cellular Automata: a paradigm for Image Processing

Images are processed for several tasks, spanning many applicative fields: from robotics to geophysics, from medicine to quality control in industry. In many cases, it is necessary to go beyond simple processing and filtering of images, introducing some "understanding" of scene contents by the automated system. The task of Computer Vision is to devise satisfactory strategies and working systems capable to reach this goal.[5]

In order to recognize objects and their configuration in a scene, it is necessary to identify and extract the significant information from an image. An image is represented, within a computer, by an array of numbers, representing the light intensity reaching a spatially distributed array of sensors. The number of points (pixels) can range from several thousands to a few millions.

Processing such an amount of information raises the problem of correctly determining the subset of data useful for understanding the scene. This process is seldom a sequential and univocal computation. It is often necessary to analyze intermediate or final results of the computation, in order to repeat some parts of the process, with different methods or parameters, trying to obtain better global results.

Image Processing methods can be classified according to several paradigms: for example, a distinction, in terms of control strategies, could be made between data-driven and expectation-driven approaches. In the former, image characteristics influence the choice of the kind of processing to be performed, while in the latter it is the requirement on the result to influence the choice of the processing methods.

Regardless of the approach chosen, it is necessary a low-level processing of the image. This term indicates the kind of processing that does not alter the image data structure, i.e., the pixel array, but changes only individual pixel values. Other forms of computation can have different data structures as input or output. Examples of low-level processing are early filtering, contour extraction, image smoothing, noise reduction.

Most low-level operations are local and uniform, i.e., every pixel changes according to a function depending only on neighboring pixels values, and executed for every pixel. Massively data-parallel computers with bidimensional mesh connected architectures (or MC^2) are the hardware capable of optimizing such operating method. Cellular Automata computational paradigm is the most appropriate for describing MC^2 algorithms. The use of massively parallel algorithms can give very good results, but has the disadvantage of being easily applicable only for simple operations. A more complex example of application of these techniques to feature extraction problems will be shown in the last part of this paper.

2.1. *Cellular Automata*

Massively parallel computation on a regular mesh, with data parallelism, can be efficiently described by the Cellular Automata (CA) paradigm.[6,7]

In the most general case, a CA is a discrete dynamical system defined on a set of cells, satisfying the following properties:

(i) cells are the elementary atoms of the system; they are placed on the nodes of a regular network. Cells connectivity defines the topology of the system. Neighborhoods are defined in terms of this topology. Each cell can be in a state chosen from a finite state set.

(ii) the system dynamic is defined in terms of a local and uniform rule. Local means that the rule changes the state of a cell according only to the state of the cells in the neighborhood. Uniform means that the same rule is applied to every cell in the CA.

(iii) the system evolution is based on synchronous application of the rule to all the cells, in discrete time steps. The state of a cell at time T_i depends only on the state of the neighboring cells at time T_{i-1}.

Assuming that S is the set of states, F is the rule and I_n a neighborhood with n cells in the network R; if $U, V \in S$ are the values of cells $u, v \in R$ respectively; then a more formal definition of the above properties is the following:

$$F : S^n \rightarrow S \tag{1}$$

$$\forall u \in R : U(T_i) = F\big(V(T_{i-1}) : v \in I_n(u)\big) . \tag{2}$$

The CA paradigm is nowadays used in many fields of science;[8] in the rest of this paper the use of this paradigm in low-level Image Processing will be discussed.

2.2. *Cellular Automata for Image Processing*

In the case of Image Processing a CA is defined in the following way: the network is a bidimensional array, having the same geometry as image pixels, and the cells correspond to pixels. The neighborhoods are defined in terms of 4 or 8-connectivity and are usually represented by $n \times n$ windows centered on the pixels. The states of the cells represent the possible luminance values. The rule is the istantiation of the algorithm performing Image Processing. As already mentioned, CA are generally used for low-level processing or filtering.[9,10]

Filtering operators are classified as pointwise or local operators. Pointwise operators are degenerate CA. They do not rely on neighborhood values; or better, their neighborhood contains only the pixel itself. Examples of this class of filters are represented by enhancement operations.

Local operators are CA, with the terminology introduced above. Many well known filtering algorithms are defined (or easily translated) in terms of local operators.

For example, a low-pass filter is aimed to reduce noise in an image. The filter is based on the hypothesis that fast spatial variations of luminance are caused by noise. The visual effect of such a filter is a blurring of the image. This filter can be implemented averaging the values in the neighborhood and substituting the result in the pixel value. The cutoff frequency of the filter decreases when the neighborhood area increases, the blurring effect also increases with the neighborhood area. Using the formalism introduced in the previous paragraph, the filter can be written as an average of neighboring pixels values:

$$U(T_i) = \frac{\sum_{v \,\in\, I_n(u)} V(T_{i-1})}{\sum_{v \,\in\, I_n(u)} 1} \;, \tag{3}$$

where the denominator simply counts the number of neighbors.

Another example of filter is the high-pass filter. This filter substitutes the pixel value with a measure of the luminance variation in the neighborhood. Higher values will correspond to points where there is a fast spatial variation in luminance. If the noise is negligible, the result is the highlighting of the border between homogeneus regions. The filter can be implemented with discrete differential operators, like the gradient, or by convolution with appropriate kernels. An example of discrete approximation to the gradient is the following:

$$U(T_i) = \sqrt{\left[V_n(T_{i-1}) - V_s(T_{i-1})\right]^2 + \left[V_e(T_{i-1}) - V_w(T_{i-1})\right]^2} \;, \tag{4}$$

where $V_n,\ V_s,\ V_e,\ V_w\ \in\ I_4(u)$; $I_4\,(u)$ is the neighborhood containing the cells 4-connected to u, and V_n, V_s, V_e and V_w are the north, south, east and west cells respectively.

In chapter 4, it will be shown another example of local operator for low-level processing.

3. PAPRICA: A Cellular Automata Machine

Current VLSI technology allows the integration of many Processing Elements (PEs) on a single chip, making it possible to build a CA machine.[11,12,13] The main problem with this high integration is the interconnecting network between the PEs. In the case of low-level Image Processing, a two-dimensional grid can be used as interconnecting topology for the inter-processor network, to form the set of cells of the CA. This chapter presents the PAPRICA system, based on a SIMD (Single

Instruction Multiple Data) architecture whose processing elements are placed on a bidimensional mesh.

In the present implementation the Processor Array is a 16×16 square matrix of single-bit PEs placed on a bidimensional regular grid, each one with full 8-neighbor connectivity. The maximum configuration for PAPRICA system consists of a square grid of 1024 PEs, composed by 8×8 chips. The inter-processor communication is an enhanced version of the classical 2-dimensional mesh connected network (MC^2), which implements in hardware only the 4 links to the North, South, East, and West cells.

3.1. *Simulating PAPRICA on a CM-2*

PAPRICA programming environment gives the possibility of writing application programs starting from standard C language. A generic application is written using C language with the help of two sets of functions (see figure 1):

- the System Library is a standard set of functions handling the interaction between the application and the physical or emulation layer;
- the Macro Library is an open set of functions that the user can use and augment to generate the PAPRICA code required by the application. These functions parametrically build segments of PAPRICA code to perform a specific task.

The environment has been designed to isolate the applications from the hardware details of the implementation. In fact, the same program can run on different platforms:

- PAPRICA hardware;
- software simulator running on a UNIX workstation or on an MS-DOS PC;
- software simulator running on a Connection Machine CM-2.

A block diagram showing the programming environment is depicted in figure 1, which shows the 3 different implementations of the processing unit. The low-level hardware-dependent processing units are accessed by the high-level blocks through a few calls to a standard interface. With this solution, the software simulators and the hardware itself are completely interchangeable, without any modifications to the high-level applications.

The first version of the software simulator, written in standard C at the Politecnico di Torino, has been compiled on SUN workstations under UNIX and on PCs under MS-DOS. The availability of a complete software simulator at system level allowed to start developing applications before the completion of the hardware. Using low-cost systems, based on slow and general purpose machines, with an obvious speed penalty, it is possible to run only limited size applications.

To overcome the speed-limitation problems, the second version of the simulator was written using the Connection Machine CM-2 available at the University of Parma. PAPRICA mesh interconnecting network has been mapped on the CM-2 hypercube. The parallel operations, virtualized in the first version of the simulator, are performed simultaneously, reaching a considerably high processing speed. The high performances of the CM-2 simulator allow both the study, development, and analysis of PAPRICA applications, and the study and optimization of the instruction set for the next version of PAPRICA.

The parallel version of the simulator was written using C$*$ language and it is fully compatible with the serial version. The user interface is the same and all the utilities written for the serial version work with the parallel version.

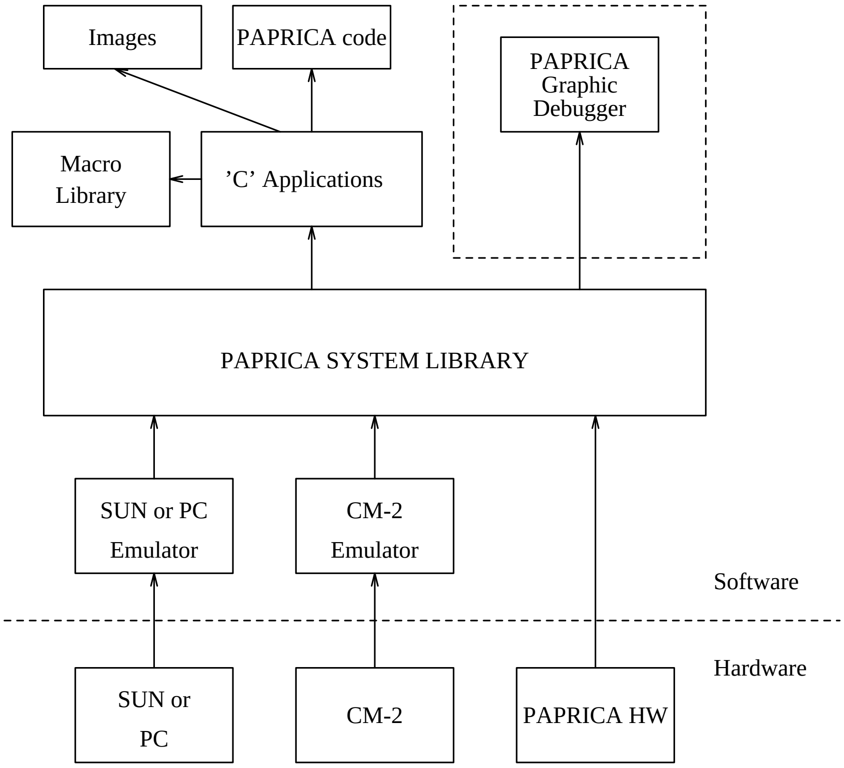

Fig. 1. PAPRICA Operating Environment.

A filter for edge detection of the type described in the previous chapter was used as test for the performances of the simulator. A Kirsch-Prewitt operator was chosen, using a 256×256 pixels image with 256 gray levels. Using the serial version of the simulator on a Sun SPARC workstation, the execution of the filter takes slightly less than 30 minutes. The parallel version needs only 30 seconds of CM-2 to perform the same operation. An estimate of the performance of PAPRICA board with the same computation gives around half a second.

Using the CM-2 as a simulator of PAPRICA imposes some constraint on the algorithms implementation (such as the virtualization mechanism), giving a speed penalty. The same filter executed on the CM-2 directly (without the simulator) takes about 0.2 seconds.

3.2. *PAPRICA Graphic Debugger*

The current version of the user interface (called PAPRICA Graphic Debugger) is based on X11-R4 windowing system. This environment was selected both for its wide availability on different workstation platforms, and for the associated communication facilities. For example, the Debugger (which uses the simulator on the CM or PAPRICA hardware) can be run from a remote host through the net, while the resulting images can be displayed on any host running X11.

PAPRICA Graphic Debugger is written using standard calls to PAPRICA library and allows direct interaction with the coprocessor board. Using the Debugger it is possible to load the program memory from a file, written in PAPRICA assembly language, or to modify its contents "on the fly", using the included one-line assembler. Programs can be executed at full speed or in single-step mode. The Graphic Debugger allows to load the image memory from files in several standard formats. It is also possible to display the contents of this memory in graphics windows with selected color palettes; on these windows zoom and pan operations are allowed.

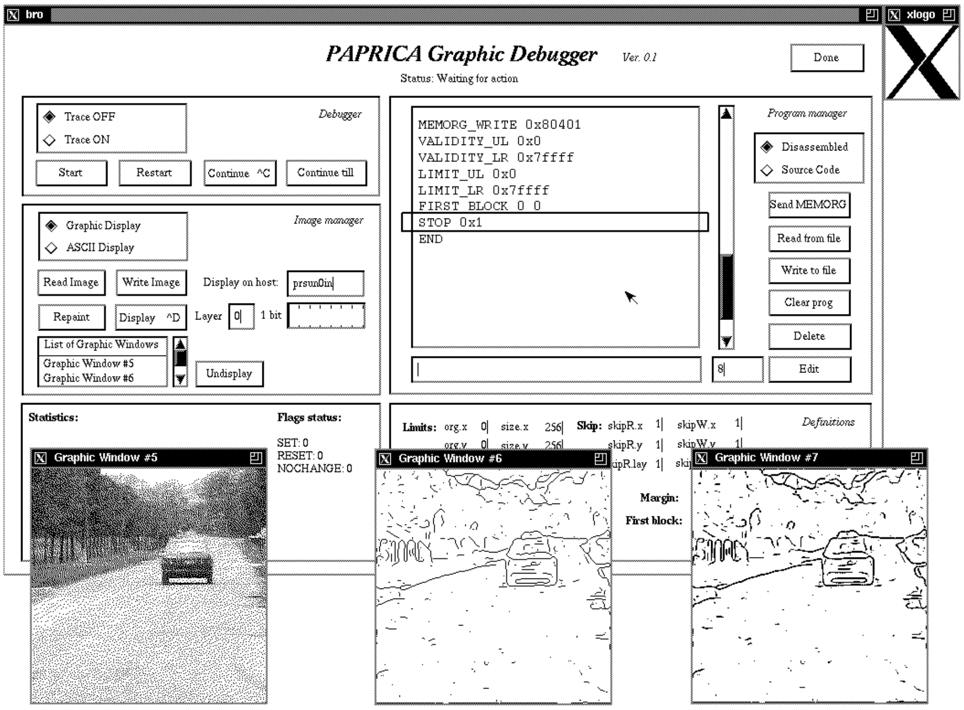

Fig. 2. PAPRICA Graphic Debugger.

A dump of PAPRICA Graphic Debugger is shown in figure 2. The main window is the Debugger, and is divided in a control panel, a manager for the graphic windows, a program manager with the editor, and two windows with statistics and current values of PAPRICA parameters. The three additional windows are the output of the image memory in three points of the program.

3.3. *Mapping of PAPRICA architecture on the CM-2*

PAPRICA instruction set comprises simple CA rules such as MOV, EXP, ERS, corresponding to the definition used in mathematical morphology,[14,15] and instructions such as BOR and LS2, used for Design Rule Checking (DRC) algorithms. Indeed PAPRICA was originally designed to run DRC algorithms for the verification of layouts of Integrated Circuits.

One of PAPRICA main features is that it can synthesize by software any CA with any neighborhood, using the direct links with the 8 neighbors and boolean operations within a single PE.[16]

The two main reasons that led to the choice of the CM-2 as the hardware platform for the implementation of PAPRICA simulator are:

- the CM-2 embeds PAPRICA interconnecting topology, and
- PAPRICA assembly language and the CM-2 C-PARIS are semantically equivalent.

The equivalence between the two languages, and the fact that PAPRICA communication topology can be seen as a sub-set of the CM-2 communication topology, makes the CM-2 a very efficient simulator of PAPRICA architecture. Table 1 shows a simple comparison between the architecture of PAPRICA and of the CM-2 available at the University of Parma.

Since the number of PEs is normally less than the number of image pixels used in generic low-level Image Processing applications, a significant part of the controller has been dedicated to the implementation of virtual processors. Due to the little memory available for each processor, it is not possible to use the virtual processor approach implemented on the Connection Machine, where the computation is serialized within each processor. PAPRICA serializes the computation in windows: the processing array is loaded with a sub-window of the image, then the computation is performed until a special instruction (`UPDATE`) is reached, and finally the result is stored back into a different image plane. These steps are iterated until all the sub-windows have been processed. PAPRICA control unit drives the sequential scanning of the image sub-windows.

Within the PROMETHEUS project, where one important feature is the architecture physical size, a small, simple, and low-cost architecture like PAPRICA, is preferable with respect to a more sophisticated general-purpose architecture, like the CM-2, which will be used just for tuning purposes.

Table 1. PAPRICA vs CM-2

Main Features

	PAPRICA	**PARMA CM-2**
Number of PEs	1k	8k
PE complexity	1 bit	1 bit
Floating Point Unit	No	Yes
Memory	4 Mbytes	256 Mbytes
Virtual PEs	elaboration serialized in windows	instructions serialized within a single PE
Network topology	2D grid	13D - Hypercube
Clock	10 MHz	7 MHz
Programming Language	low-level	high and low level
Size	2 VME boards	3 m^3

4. A Case Study: Parallel Feature Extraction with PAPRICA

To assist the car driver, an important information is the position of the car with respect to road boundaries. This chapter describes a filter for the extraction of these boundaries from road images.

4.1. *Feature metric*

The definition of the behavior of the filter is equivalent to the characterization of the feature in terms of local properties of pixels. Each pixel in the image should then be able to determine if it is part of the feature or not, i.e., should measure the significance of the information contained in its neighborhood with respect to the local properties of the feature.

Each pixel[a] makes use of an appropriate function f to calculate its Descriptive State D, using the luminance values V of pixels v in his neighborhood I_n:

$$D(u) = f\big(V \;:\; v \in I_n(u)\big) \; . \tag{5}$$

Each pixel should also compute its Characteristics State C, applying the same function f to a neighborhood configuration where pixel values V' correspond to the presence of the feature of interest:

$$C(u) = f\big(V' \;:\; v \in I_n(u)\big) \; . \tag{6}$$

[a] In Eq. (5), (6), (7) and (8) u and v are cells (pixels in the image), and the value of a cell contains the pixel luminance U or V, the reference state V', the computed values D, C, S and a copy of the hypothesis set H.

From the computational point of view, Eq. (4.2) is not of practical use. Since the specification of all the possible configurations of V' is not feasible, another form is used:

$$C(u) = g\big(u, I_n(u), H\big) \ . \tag{7}$$

Here it is used a set H of hypothesis, and the function g is such that its values are the same that would be obtained by f in Eq. (4.2).

The two states are then compared and, if they are properly encoded, the pixel significance (with respect to the feature of interest) is represented by the Significance State S, which measures the distance between D and C:

$$S(u) = \| \ D(u) \ - \ C(u) \| \ . \tag{8}$$

The determination of g, and the choice of H are the main points in the definition of the feature extraction algorithm.

4.2. *Identifying road boundaries*

In the present example, the main hypothesis is that images are taken by a camera mounted on a car. In this case, the street boundaries can be defined as the two lines connecting the bottom of the image with the Focus Of Expansion (FOE)[17].

The image digitized by the camera is first processed by filters such as the ones defined in Eq. (3) and (4) for edge enhancement.[18] The edges thickness is reduced to one pixel by a skeletrization algorithm.[19] Figure 3a shows the result of this pre-processing; this is the input to the feature extraction algorithm.

To compute the Descriptive State $D(u)$, every pixel computes the local direction of the edge to which it belongs. Figure 3b shows the image of $D(u)$; the intensity is proportional to the direction.

The Characteristics State $C(u)$ measures the slope of the geometrical line connecting the pixel to the FOE. The position of the FOE can be determined by optical flow analysis, and its coordinates are broadcasted to all the image pixels. Figure 3c shows the image of $C(u)$; the intensity is proportional to the slope.

In this filter, the Significance State $S(u)$ is the difference between the direction in $D(u)$ and the slope in $C(u)$.

To improve the result, a threshold is also applied to $S(u)$ to eliminate pixels with low significance. The edges shorter than 20% of the image side are also discarded (in the image represented in figure 3, the side is 512 pixel and the minimal length is about 100 pixels). A final step eliminates all the pixels in the upper half-plane, with regard to the FOE, since street boundaries are supposed to be below. The result of the whole filtering process is shown in figure 3d.

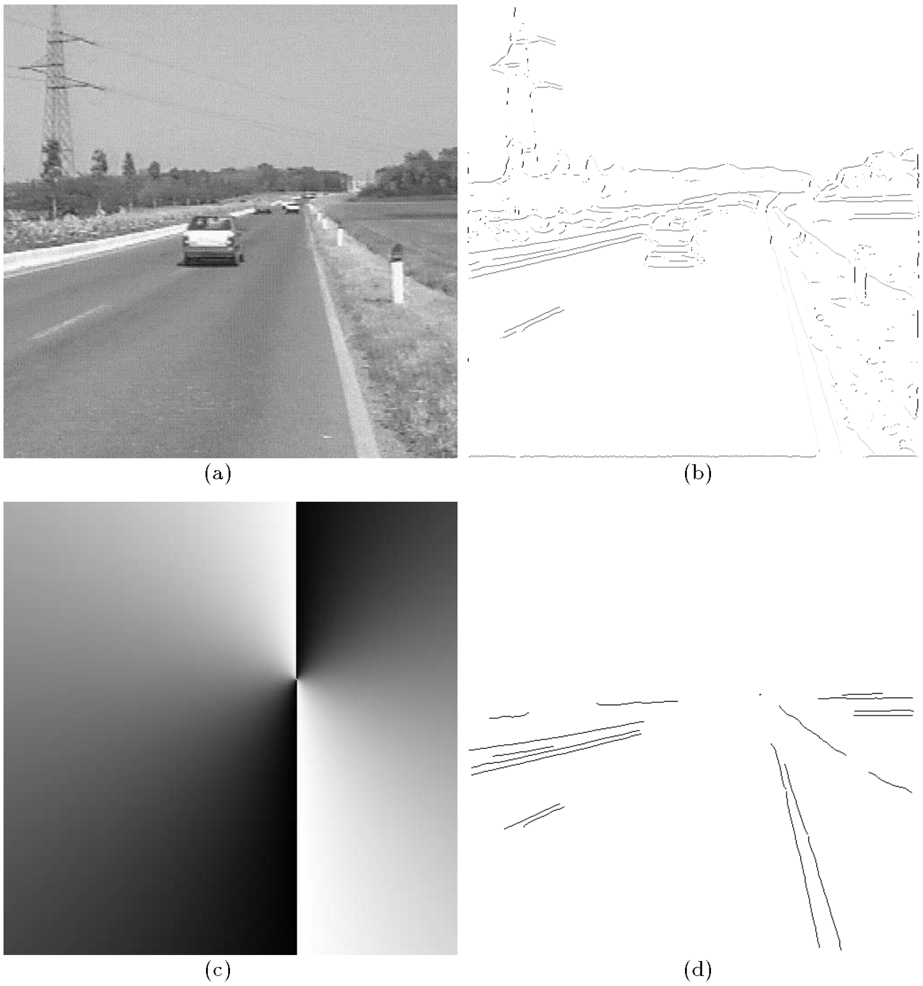

Fig. 3. (a) Input image. (b) Descriptive State.
(c) Characteristics State. (d) Output, Significance state.

The first version of this algorithm has been implemented in C$*$ on a Connection Machine CM-2 with a 7 MHz clock cycle, 8192 Processing Elements, and no floating point units (FPU); starting with a 256×256 pixels image, a non-optimized version of the algorithm runs in more than 4 seconds. The preprocessing stage takes about 2 seconds, computing the Descriptive State takes about 1.2 seconds, while the determination of the Characteristics State needs about 0.25 seconds. The final filtering of the short lines takes about 0.75 seconds.

Acknowledgements

This work was partially supported by CNR, Progetto Finalizzato Trasporti, in the frame of PROMETHEUS program, under project number CNR 9001591.PF93, and by CNR under project number CNR 9001955.PF61.

References

1. G. Adorni and G. Conte, *Int. J. Mod. Phys.*, **this issue**.
2. J. J. Little, G. E. Blelloch and T. A. Cass, *IEEE Trans. on PAMI*, **11**, 244, (1989).
3. L. Reyneri, G. Conte and F. Gregoretti, in *Proceedings of IFIP-TC 10 Conference*, Pisa, 19-21 September 1988.
4. G. Conte *et al.*, in *CAD accelerators*, ed. Abmler, Agrawal and Moore, (Elsevier Science Publishers, 1991), p. 177
5. D. H. Ballard, C. M. Brown, *Computer Vision*, (Prentice-Hall, Englewood Cliffs, 1982).
6. D. W. Hillis, *The Connection Machine*, (MIT Press, Cambridge, 1985).
7. T. Toffoli and N. Margolus, *Cellular Automata Machines: a new environment for modeling*, (MIT Press, Cambridge, 1987).
8. S. Wolfram (ed.), *Theory and applications of Cellular Automata*, (World Scientific, Singapore, 1986).
9. G. Adorni and V. D'Andrea, in *Proceedings of the Workshop on Mobile Robotics for Civil Works*, (Assan, Bruxelles, 1991).
10. G. Adorni *et al.*, in *Trends in Artificial Intelligence*, eds. E. Ardizzone, S. Gaglio and F. Sorbello, (Spriger-Verlag, Berlin, 1991), p. 322.
11. T. Toffoli, *Physica*, **D10**, 195, (1984).
12. T. Toffoli and N. Margolus, *Physica*, **D47**, 264, (1991).
13. A. Broggi *et al.*, in *CAD and Architectures: Report on Architectures and Algorithms for VLSI Design*, (CNR, Roma, 1990), p. 21.
14. J. Serra, *Image Analysis and Mathematical Morphology*, (Academic Press, London, 1982).
15. R. M. Haralick, S. R. Sternberg and X. Zhuang, *IEEE Trans. on PAMI*, **9**, 532, (1987).
16. A. Broggi, N. Patrignani and A. Ambriola, in *CAD and Architectures: Report on Architectures and Algorithms for VLSI Design*, (CNR, Roma, 1990), p. 143.
17. B. Bhanu *et al.*, in *Proceedings of DARPA Image Understanding Workshop*, (Morgan Kaufmann, San Mateo, 1989), p. 370.
18. A. Broggi, *Parallel and Local Simple Features Extraction: Street Boundary Detection in Traffic Images*, University of Parma Technical Report, CS91-2, (1991).
19. C. Arcelli, L. P. Cordella and S. Levialdi, in *Electronics Letters*, **11**, 148, (1975).