Parallel Image Clustering: a Real-Time Application for a Special-Purpose Architecture [1]

A.Broggi and A.Gandini
Dipartimento di Ingegneria dell'Informazione
Università di Parma
Parma, ITALY, I-43100

Abstract

The main topic of this work is the early visual processing of images grabbed by a camera placed on a vehicle, a car, moving in a real road traffic scenario. More precisely, the paper presents a parallel algorithm for real-time segmentation of images.

Due to the constraints imposed by this kind of applications (real-time response), a special-purpose massively parallel SIMD architecture, PAPRICA, has been used; the segmentation algorithm has been conceived and designed to be implemented on such an architecture, and on parallel mesh-connected machines in general.

This work is just an example of the application of PAPRICA architecture to real-time problems, where the reduction of the processing time and the performance analysis play a fundamental role in the algorithm design stage. In fact, as often happens in industrial applications, the solution to real problems that require high performances needs to be simple and well tuned.

Since PAPRICA hardware is now under testing, the first version of this algorithm has been implemented on a simulator running on a Connection Machine CM-2. This simulation allowed the studying and tuning of the algorithm, in order to achieve the real-time version for PAPRICA system.

1 Introduction

The main target of the PROMETHEUS Eureka project is to increase the vehicle safety using the facilities offered by the present information technology. This work focuses on the use of a camera placed on a moving vehicle in order to derive useful information to assist the driver. The camera captures sequences of images and pipelines them to an on-board computer for processing. To cope with the high throughput needed for low-level processing, a special purpose parallel architecture (PAPRICA, PArallel PRocessor for Image Check and Analysis, [7, 2, 10, 16]) is used: it is the engine for low-level tasks such as image restoration, image enhancement, simple feature extraction, as well as massive parallel tasks in general. PAPRICA architecture is a SIMD board connected to a host computer, with up to 1024 Processing Elements disposed on a 2-dimensional mesh, and implements a multiresolution Cellular Automata computational paradigm [25, 1, 8].

Since the main target of this work is to reach real-time performances on a specific architecture, a lot of time was dedicated to the tuning of the algorithm, in order to take advantage of the special features of the system.

The presented algorithm can be compared to the *Weak Membrane* bayesian approach [3, 5, 13, 14, 15, 19, 20, 21, 22] which tends to maximize the *a-posteriori* probability following some simple hypothesis. It is based on the *weak continuity* assumption, namely the algorithm tends to clusterize the image, while enhancing the edges strength. Unfortunately the bayesian approach is based on iterative processes which need a lot of floating point computations, and thus it is not suitable for real-time applications. This work presents a simpler clustering algorithm, based on the *weak continuity* assumption as well, which improves greatly the performances and needs only a few iterations on integer data. The proposed algorithm is a revision of the Adaptive Smoothing [12, 23], where a fixed thresholding factor is replaced with a function of the neighborhood, in order to enhance also the weak and isolated intensity discontinuities. Moreover, the strong clustering allows the substitution of heavy edge

[1] This work was partially supported by CNR Progetto Finalizzato Trasporti in the frame of the Eureka PROMETHEUS project.

detection algorithms (i.e. the one proposed by Canny [11]) with some simpler gradient-based filters.

In section 2, the formal description of the proposed image clustering algorithm is given, together with some real-time approximations; in section 3 a brief description of PAPRICA architecture is presented, while section 4 shows the implementation of the algorithm on PAPRICA system. Finally, section 5 and 6 present some performance analysis results together with some concluding remarks.

2 Image clustering

Due to the corrupting effects of noise, camera blur, and other artifacts of the grabbing process, each original image needs a pre-filtering operation in order to increase its signal-to-noise ratio before edge extraction. For real-time applications, on mesh connected SIMD architectures, this step is typically accomplished by the application of a simple average filter, which is known in literature as a convolution with a 3×3 fixed mask; namely, the new value that will be stored in each pixel is the result of a fixed weighted average of the values stored in its eight neighboring pixels. The main disadvantage of this step is the global smoothing of the whole image, which causes a sensible degradation in the edges sharpness. In this work, a different low-level algorithm is proposed in order to keep the edges original strength unchanged: the average will be computed weighting the values stored in the neighboring pixels with a function of their gradient magnitude. In this way, the sharp edges will be far less smoothed than the almost uniform areas, and the output image will consist of a segmented version of the initial image, with blobs of uniform color, capable of generating a sort of closed-contours gradient image.

The algorithm is a revision of the Adaptive Smoothing [12] algorithm, introduced by Saint-Marc and Medioni [23]. The definition of this filter's behavior is shown in the following: let's define $I^n(x,y)$ the luminance intensity of the pixel at coordinates (x,y) and iteration number n, belonging to image I; the value $I^{n+1}(x,y)$ is defined as follows:

$$I^{n+1}(x,y) \triangleq \frac{\sum_{i=-1}^{+1}\sum_{j=-1}^{+1} I^n(x+i,y+j) \times w^n_{x,y}(x+i,y+j)}{\sum_{i=-1}^{+1}\sum_{j=-1}^{+1} w^n_{x,y}(x+i,y+j)} \tag{1}$$

where $w^n_{x,y}(u,v)$ represents the weight of the pixel at coordinates (u,v) at iteration number n, when computing the new value of $I(x,y)$; $w^n_{x,y}(u,v)$ is a function of the gradient magnitude $g(u,v)$. In the original Adaptive Smoothing, the definition of $w^n_{x,y}(u,v)$ (independent of x and y)

$$\Big[w^n_{x,y}(u,v)\Big]_{\text{Adaptive Smoothing}} \triangleq e^{-\frac{|\ g^n(u,v)\ |^2}{2K^2}} \tag{2}$$

comprises the constant parameter K (which is equivalent to σ in Gaussian Smoothing [11]), which fixes the discontinuities to be preserved; the parameter K acts essentially as a fixed threshold. The revision of this algorithm eliminates the lack of dependence of this threshold K on the neighborhood, in order to enhance also the weak and isolated intensity discontinuities. $w^n_{x,y}(u,v)$ is thus defined as follows:

$$w^n_{x,y}(u,v) \triangleq e^{-\frac{|g^n(u,v)|^2}{2|k^n(x,y)|^2}} \tag{3}$$

$$\text{with} \quad k^n(x,y) = M \cdot \max_{i,j=-1,0,+1} g^n(x+i,y+j)$$

where $g^n(x,y)$ represents the magnitude of the gradient defined as:

$$g^n(x\prime,y\prime) = \sqrt{\left(\frac{\partial I^n}{\partial x}\right)^2_{x=x\prime} + \left(\frac{\partial I^n}{\partial y}\right)^2_{y=y\prime}} \tag{4}$$

The parameter M measures the deviation from the classical average filter (where $M \to \infty$): the lower is M, the higher is the ratio between the weights $w_{x,y}(u+i,v+j)$, with $i,j = -1,0,+1$.

In figure 1, the first curve represents the brightness distribution over a linear image profile (100 pixels) with 3 edges (with strength respectively 20, 100, and 50, each one spread in a 16 pixels interval) and with additive white gaussian noise ($\sigma = 2$, $\mu = 0$); the second curve represents the filtered output; and the last one depicts the output gradient magnitude. Figures 1.a and 1.b show the application of the classical Adaptive Smoothing algorithm (with threshold K equal to 3 and 0.2 respectively), using the definition of w shown in equation (2) [12, 23]. In figure 1.a, the high threshold K detects clearly only two of the three edges, while smoothes out the third; on the contrary, in figure 1.b, K has been made small enought to detect the first weak edge, but now the algorithm identifies as edges also the ascending and descending ramps of the other two edges, decreasing their strength. Figure 1.c shows the application of the proposed revision of

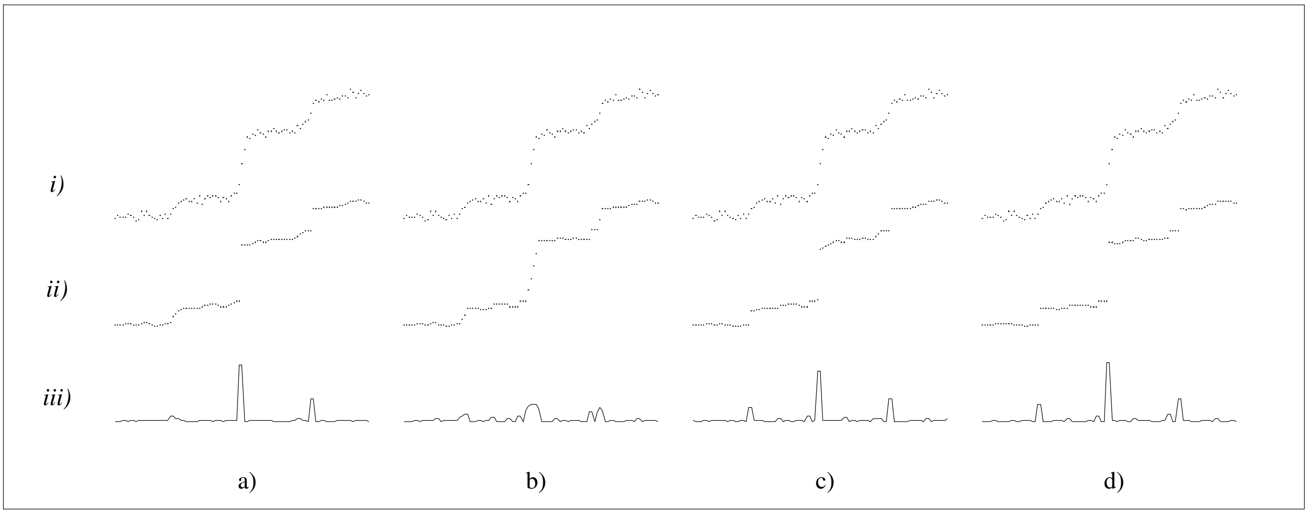

Figure 1: (i) brightness profile along an image with 3 edges; (ii) filtered image: (iii) gradient.

the Adaptive Smoothing algorithm, with the parameter M equal to 0.35 (see section 2.1): since the filter keeps also the weak edges, in the regions far from the real edges the output gradient is not as flat as in the previous case, but now a small threshold will enable the detection of all of the three edges.

The proposed filter differs from the classical *Weak Membrane* [20, 3, 5, 13, 14, 15, 19, 21, 22] algorithm mainly in the two following aspects:

- in the bayesian approach, the image grabbed by the camera constitutes the *observable* which, together with some hypothesis on weak continuity, is used to maximize the *a-posteriori* probability: namely, the observable is used to identify the original image which generated it. During this iterative computation, the grabbed image drives every iteration, keeping each intermediate image close to itself: each step is a function of the previous step and of the grabbed image.

  The proposed filter, on the other hand, takes the grabbed image as the starting image of an iterative computation, and modifies it step by step, inducing the appearance of clusters. In this second approach, the discarding of small details is due basically to the fact that each iteration is a function only of the previous step, loosing contact with the grabbed image; anyway, this loss of details is not very important when compared to the typical approximations of the following processing steps in real-time applications.

- Moreover, the proposed filter has the great advantage of being more simple, allowing a real-time implementation: the more sophisticated bayesian approach requires, in fact, a lot of floating point computations that decrease considerably its processing speed.

2.1 A computational approach

Since the computation of nine exponential values for each pixel is very time-consuming and not easily implementable on the extremely simple processing elements of massively parallel architectures, an approximation is needed in order to make the algorithm more suitable for real-time processing. The definition of w, can be rewritten as:

$$w^n_{x,y}(u,v) = e^{-\frac{1}{2M^2}\left|\frac{g^n(u,v)}{\max\limits_{i,j\,=\,-1,0,+1} g^n(x+i,y+j)}\right|^2} \qquad (5)$$

The fraction inside the modulus can assume values in the range $[0 \div 1]$, thus the argument of the exponential can range from 0 to $-\frac{1}{2M^2}$. Therefore the greater is M, the shorter is the exponential argument range interval, and thus the more successful (sharper) is the approximation of w with a polynomial expression. Unfortunately, the greater is M, the less effective this kind of filtering is, since it gets closer to the classical fixed-weight average filter. Experimental tests have shown that $M = 0.3 \div 0.4$ represents a good trade-off, and w can be efficiently approximated by a first order polynomial expression [6]: thus, with $M = 0.35$,

$$w^n_{x,y}(u,v) \simeq 1 - \frac{g^n(u,v)}{\max\limits_{i,j\,=\,-1,0,+1} g^n(x+i,y+j)} \qquad (6)$$

Since w represents a weight, it can be multiplied by an arbitrary factor $f^n_{x,y}$: using

$$f^n_{x,y} = \frac{1}{\max\limits_{i,j\,=\,-1,0,+1} g^n(x+i,y+j)}\ , \qquad (7)$$

equation (6) becomes

$$w^n_{x,y}(u,v) \simeq \max_{i,j\,=\,-1,0,+1} g^n(x+i,y+j) - g^n(u,v) \quad (8)$$

Figure 1.d depicts the result of the smoothing using the approximated weights shown in equation (8): the root mean square computed using the values of figure 1.c and 1.d (100 values each) is about 0.9, showing a fairly good approximation.

The approximation due to the application of the (8) instead of the (3), is mainly restricted to the first few iterations, when the image clusters begin to appear and the argument of the modulus in the (5) is almost equally distributed in the range $[0 \div 1]$; on the other hand, in the following iterations, it will be polarized around 0 or 1, and the approximation error will be smaller.

Moreover, to simplify each iteration, the gradient field computation can be approximated in a 3×3 window using the well-known following expression [26, 4]:

$$g\prime^n(x,y) = \max\Big(\big|I^n(x,y-1) - I^n(x,y+1)\big|, \big|I^n(x-1,y) - I^n(x+1,y)\big|\Big) \quad (9)$$

Thus, the approximated expression of $I^{n+1}(x,y)$ becomes:

$$I^{n+1}(x,y) \simeq \qquad (10)$$
$$\frac{\sum\limits_{i=-1}^{+1}\sum\limits_{j=-1}^{+1} I^n(x+i,y+j) \times \Big(G^n_{xy} - g^n(x+i,y+j)\Big)}{\sum\limits_{i=-1}^{+1}\sum\limits_{j=-1}^{+1} \Big(G^n_{xy} - g^n(x+i,y+j)\Big)}$$

$$\text{with} \quad G^n_{xy} = \max_{u,v=-1,0,+1} g^n(x+u,y+v)$$

Figures 2 and 3 show the original and the output images after 6 iterations, while figure 4 shows the application of a simple and fast gradient filter (such as the one defined by equation (9)) to the output image, followed by a threshold.

3 PAPRICA system architecture

PAPRICA is a massively parallel SIMD architecture, which has been designed as a specialized coprocessor to be attached to a general purpose host workstation and, as a whole, comprises 4 major functional parts namely the Image and Program Memories, the Processor Array (PA) and the Control Unit.

The first prototype of the PA is composed of an array of ICs, each of them containing a sub-array of 4×4 Processing Elements (PE). In the present implementation, the PA is a 16×16 square matrix of 1-bit PEs each one with full 8-neighbors connectivity, while the maximum configuration will be achieved in the next version, and will consist of a square grid of 1024 PEs. The inter-processor communication network is an enhanced version of the classical 2-dimensional mesh connected network (MC^2), in which only the 4 main links (Top, Bottom, Left, and Right) are implemented in hardware. PAPRICA computational paradigm [16, 9] is based on the concept of *matching operator*, derived from the *hit-miss transform* described in [24]. This is a rather general approach which includes the other morphological operators as special cases [17].

Since the number of PEs is normally less than the number of image pixels used in generic low-level image processing applications, a significative part of the controller has been dedicated to the implementation of virtual processors. Due to the little memory available for each PE (64 bits), it is not possible to use the virtual processor approach implemented on the Connection Machine [18], where the computation is serialized within each processor (which contains several data belonging to several pixels). PAPRICA, in fact, 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 sent back to the Image Memory, and stored 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. The main problem with this concept of virtual processors is mainly due to the limited dimensions of the PA, where the border processors can't access their complete neighborhood. In fact, after the execution of an instruction which requires full (3×3) neighborhood access, the values stored in the PA border processors of the PA are no more valid. Thus, the next windows that will be transferred into the PA will be partially overlapped with the previous one, in order to evaluate the previously invalidated results.

PAPRICA Image Memory can be rearranged run-time using special instructions, which set the image height, width and deepness. Each PE has 64 bits of internal memory; since the Image Memory deepness is often more than 64 bits, some PAPRICA control instructions are devoted to the definition of Image Memory portion to be loaded and stored back. Moreover, since for each parallel computation two sequential accesses to the Image Memory are required, it is possible to fetch and store back the data in a useful way. In fact, the data to be transferred into the PA can be non logically adjacent to each other, allowing to undersample or oversample the image modifying its resolution. This behavior is controlled by a set of registers which can be altered run-time. The possibility to reduce and increase the image dimensions and resolution allows a very efficient use of PAPRICA architecture as a pyramid. Moreover, some simple software algorithms allow also to simulate any kind of pyramidal interconnections between the different pyramid layers.

PAPRICA processing speed S_{pr} is a function of parameters that depend on the system technology and architecture (the number of PEs Q^2, the memory cycle time T_M, the PA cycle time T_C), on the particular computational task (the length of the program L, and the number of Graphic Operators G in the program, namely the number of operators that require full neighborhood access), and on the number and displacement of special instructions (`UPDATE`).

If the whole image to be processed fits into a PA of Q^2 elements, considering $T_C \simeq T_M$ and $L \ll 2Q^2$, S_{pr} becomes (see [7]):

$$S_{pr} \simeq \frac{1}{2\ T_M} \tag{11}$$

showing that the computational time is bounded by the I/O time. If, on the other hand, $L \gg 2Q^2$, then S_{pr} reduces to:

$$S_{pr} \simeq \frac{Q^2}{L\ T_C} \tag{12}$$

showing that for processing bounded problems the speedup is linear with the number of processors.

Moreover, considering also the fact that each PE in a border site in the PA cannot perform neighborhood accesses and thus it cannot produce significative results, each graphic operator reduces the significative area (*Validity Area*). Equation (13), obtained after few algebraic manipulations, shows the dependence of S_{pr} from different parameters; n_{upd} indicates the total number of `UPDATE` instructions into the program.

$$S_{pr} = \frac{\left(Q - \frac{2\ G}{n_{upd}}\right)^2}{2\ Q^2\ T_M\ n_{upd} + L\ T_C} \tag{13}$$

After the optimization of the n_{upd} parameter, it can be shown that, for sufficiently large programs, the maximum processing speed becomes:

$$(S_{pr})_{max} \simeq \frac{4\ Q^2}{100\ Q\ G\ T_M + 9\ L\ T_C} \tag{14}$$

This result shows that in the general case, where the number of processors is far smaller than the image pixels and a large number of graphic operators is used, the speedup is proportional to Q, that is the square root of the number of PEs. The complete discussion about PAPRICA performances can be found in [7].

The current hardware implementation of PAPRICA architecture is composed by 16×16 PEs and has a memory cycle time T_M of 250 ns and an array cycle time T_C of 500 ns; the typical values for the application-dependent parameters are: $L \simeq 200 \div 300$ and $G \simeq 20 \div 40$; the correspondent computational speed is about 70 kpixel/s, while the maximum computational speed (achieved with $G = 0$) is about 2 MPixel/s.

Currently PAPRICA hardware board is under testing at the Politecnico di Torino; the development of applications is possible with the help of the two versions of PAPRICA simulator: the parallel one, running on a Connection Machine CM-2 with up to 8k PEs, whose high performances allow the design and interactive testing of algorithms; and its serial version, running on a SUN workstation, which simulates PAPRICA at the system level, giving some execution statistics.

4 PAPRICA system implementation

The implementation of this algorithm on PAPRICA system has been written directly in PAPRICA Assembly Language, optimizing the length of the code and the number of neighborhood accesses. The clustering algorithm can be divided into:

- Gradient computation: it needs 8 neighborhood accesses for each pixel (or PE) and stores the result in 8 binary layers; in order to reduce the number of neighborhood accesses of the following processing steps, each PE stores in its memory also a copy of the values fetched from its 8 neighbors; this require $8 \cdot 8 = 64$ additional binary layers.

- Maximum gradient computation: each PE fetches and stores in memory its 8 neighboring gradient values; then a simple algorithm is used to get in parallel the maximum value among the 9 values.

- Final computation: at this point every PE has in its memory a copy of the brightness and gradient values of its neighbors and the value of the maximum gradient (using 136 binary layers), and can compute the final value according to equation (10) without any other neighborhood access, and using only boolean operations. For these operations 36 additional temporary layers are needed.

Since PAPRICA Image Memory consists of 32 Mbits and can be rearranged according to the requirements of the specific case, some simple considerations impose the following structure: 256×256 pixels for each one of the 512 layers of depth.

5 Performance analysis

Since PAPRICA hardware board is currently still under testing, it is possible to evaluate the performances of this specific implementation of the algorithm, only using PAPRICA simulator or the equations derived theoretically [7]. Table 1 shows the time required for the simulation on different platforms, using both PAPRICA serial and parallel simulators.

Simulation platforms	**times**
Sun 4/65 serial simulator, 16×16 PA	29'35"
CM-2 parallel sim., 64×64 PA, 4k PEs	5'32"
CM-2 parallel sim., 128×128 PA, 4k PEs	4'28"
CM-2 parallel sim., 256×256 PA, 4k PEs	1'44"
CM-2 parallel sim., 128×128 PA, 8k PEs	2'43"
CM-2 parallel sim., 256×256 PA, 8k PEs	1'04"

Table 1: Simulation performances on different platforms: SUN, CM-2 with 4k and 8k PEs

From a theoretical point of view, the main performance index for the computational speed is given by both the computational complexity and the communication usage, because the quantity of I/O required by the computation grows with the cardinality of the neighborhood used. In this specific case, each clustering step is made of two 3×3 neighborhood-based Cellular Automata, which attains high performances on PAPRICA.

Table 2 shows some results of simulation of this algorithm, written in PAPRICA Assembly language, on architectures with different PA dimensions. For a given image dimension, it shows the number of memory accesses and clock cycles, as well as the total time and computational speed, needed to perform a single iteration on a 256×256 pixel image with 8 bits/pixel. The different image dimensions shown in table 2 have been chosen in order to subdivide the images in an integer number of windows to load the PA with. Using $G = 2$, $n_{upd} = 1$, and $L = 2068$ in equation (13), it is possible to compare the theoretical computational speed (table 2, last column) with the simulation results (table 2, 5^{th} column).

Hardware platforms and algorithms	**times**
C* program on a 4k PEs CM-2	390 ms
Approx. C* program on a 4k PEs CM-2	320 ms
C* program on a 8k PEs CM-2	200 ms
Approx. C* program on a 8k PEs CM-2	160 ms

Table 3: Performances of a C* version of the algorithm on a Connection Machine CM-2

The first version of this algorithm has been implemented in C* on a Connection Machine CM-2 with a 7 MHz clock cycle, and up to 8192 PEs, and floating point units, both in its plain and approximated versions; table 3 allows a comparison between PAPRICA and CM-2 performances.

6 Conclusion

In this paper a fast and highly parallelizable clustering algorithm is discussed. It can be expressed using the Cellular Automata computational model, and therefore can be implemented on SIMD mesh-connected parallel machines as a low-level filter. The original contribution of this work is mainly centered on the effort to improve the performances of well-known basic filters, in order to achieve real-time processing speed on a specific architecture (PAPRICA). This filter needs in fact only a few iterations ($4 \div 6$) on 6 bit integer data to produce a heavily segmented output.

The typical following processing steps in a vision system (such as the one described in the Introduction) require as input data a binary image containing the luminance discontinuities; since the image obtained with this algorithm consists of a set of disjoint clusters with well-marked borders, a simple and fast gradient-based

PA dimensions	Image dimensions	Memory cycles	Array cycles	simulated times	simulated speed	computed speed
16×16	256×256	705600	911988	632.4 ms	103 kp/s	123 kp/s
16×16	192×192	131072	529408	297.4 ms	124 kp/s	123 kp/s
64×64	240×240	131072	33088	49.3 ms	1168 kp/s	1166 kp/s
128×128	248×248	131072	8272	36.9 ms	1666 kp/s	1665 kp/s
256×256	256×256	131072	2068	33.8 ms	1878 kp/s	1878 kp/s

Table 2: Performances on PAPRICA architectures with different PA dimensions

filter is sufficient to determine the image edges. Moreover, the growing of the blobs tends to reduce the luminance discontinuities thickness, heavily decreasing the number of thinning iterations generally needed on the gradient image. Another effect of this filter is a sort of hysteresis process: a weak edge, spread over a fairly large interval, becomes sharper and sharper with the expansion of its two adjacent clusters untill they face each other; this fact contributes to the generation of continuous edges even when in the original image they are not very strong. On the other hand, the main problem of this filter is that, increasing the number of iterations, the output image tends to round the sharp corners and gets fairly different from the original one.

References

[1] G.Adorni, A.Broggi, G.Conte, V.D'Andrea, C.Sansoè, "High-level and low-level computer vision: towards an integrated approach," in *Trends in Artificial Intelligence*, Lecture Notes in Artificial Intelligence, Springer-Verlag, vol.549, pp.322-331, 1991.

[2] G.Adorni, A.Broggi, G.Conte, V.D'Andrea, "Massively parallel architectures for image processing: a case study," in *Perception,* vol.20, n.1, pp.68-69, 1991.

[3] L.Ambrosio, "Variational problems in svb," *Technical Report,* Lecture Notes, MIT, Cambridge, MA, 1988.

[4] D.Ballard, C.Brown, *Computer Vision.* Englewood Cliffs: Prentice-Hall, 1982.

[5] A.Blake, A.Zisserman, "Visual Reconstruction," MIT Press, Cambridge, MA, 1987.

[6] A.Broggi, "Parallel and Local Simple Features Extraction: Street Boundary Detection in Traffic Images", *Internal Report CS91-2,* University of Parma, Italy, 1991.

[7] A.Broggi, G.Conte, F.Gregoretti, L.Reyneri, L.Rigazio, C.Sansoè, C.Zamiri, *PAPRICA,* in 'CAD and Architectures: Reports on Architectures and Algorithms for VLSI Design', Rome, CNR, Annex A, pp.21-141, 1990.

[8] A.Broggi, V.D'Andrea, "Cellular Automata Machines as a Computational Paradigm for low-level Vision," in International Journal of Modern Physics C, V.4, No.1, pp.5-16, Singapore, World Scientific, 1993.

[9] A.Broggi, V.D'Andrea, F.Gregoretti, "A low-cost parallel VLSI architecture for low-level vision," in *Proceedings IAPR Workshop on Machine Vision and Applications*, Tokyo, Japan, Dec. 1992.

[10] A.Broggi, N.Patrignani, A.Ambriola, *Image Analysis on PAPRICA,* in 'CAD and Architectures: Reports on Architectures and Algorithms for VLSI Design', Rome, CNR, Annex B, pp.143-193, 1990.

[11] J.F.Canny, "A computational approach to edge detection," in *IEEE PAMI* 8 (6), pp.679-698, November 1986.

[12] S.L.Gazit, G.Medioni, "Multi-scale contour matching in a motion sequence," in *Proceedings of DARPA Image Understanding Workshop*, San Mateo, California, Morgan Kaufmann Pub. Inc., May 1989.

[13] D.Geiger, F.Girosi, "Parallel and deterministic algorithms for markov random fields: surface reconstruction and integration," in *IEEE PAMI* 13, pp.401-412, 1991.

[14] D.Geiger, A.Yuille, "A common framework for image segmentation," International Journal of Computer Vision, IJCV 6, pp.227-243, 1991.

[15] S.Geman, D.Geman, "Stochastic relaxation, gibbs distributions and bayesian restoration of images," in *IEEE PAMI* 6, pp.721-741, 1984.

[16] F.Gregoretti, L.M.Reyneri, C.Sansoè, A.Broggi, G.Conte, *PAPRICA: a VLSI accelerator for hierarchical morphology,* submitted to IEEE Trans. on Circuits and Systems.

[17] R.M.Haralick, S.R.Sternberg, X.Zhuang. "Image analysis using mathematical morphology," in *IEEE PAMI,* vol.9, N.4, July 1987.

[18] W.D.Hillis, *The Connection Machine.* Cambridge, Mass., MIT Press, 1985.

[19] C.Koch, J.L.Marroquin, A.Yuille, "Analog 'neuronal' networks in early vision," in Proc. Natl. Acad. of Sci. 83, pp.4263-4267, 1985.

[20] R.A.Marques Pereira, A.Broggi, D.Geiger, "Beyond step edges: 2^{nd} order Image Restoration," *Internal Report CS91-3,* University of Parma, Italy, 1991.

[21] J.L.Marroquin, "Deterministic bayesian estimation of markovian random fields with applications to computational vision," in Proc. 1st International Conference on Computer Vision, London, June 1987.

[22] D.Mumford J.Shah, "Boundary detection by minimizing functionals," in *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, San Francisco, CA, 1985.

[23] P.Saint-Marc, G.Medioni, "Adaptive smoothing for feature extraction," in *Proceedings of Image Understanding Workshop*, Boston, Mass, pp.1100-1113, Apr. 1988.

[24] J.Serra, *Image Analysis and Mathematical Morphology,* London, Academic Press, 1982.

[25] T.Toffoli, N.Margolus, *Cellular Automata Machines.* Cambridge, Mass., MIT Press, 1987.

[26] P.Zamperoni, *Metodi dell'elaborazione digitale di immagini*, Masson, Milano, 1990.

Figure 2: Original Image

Figure 3: Output Image

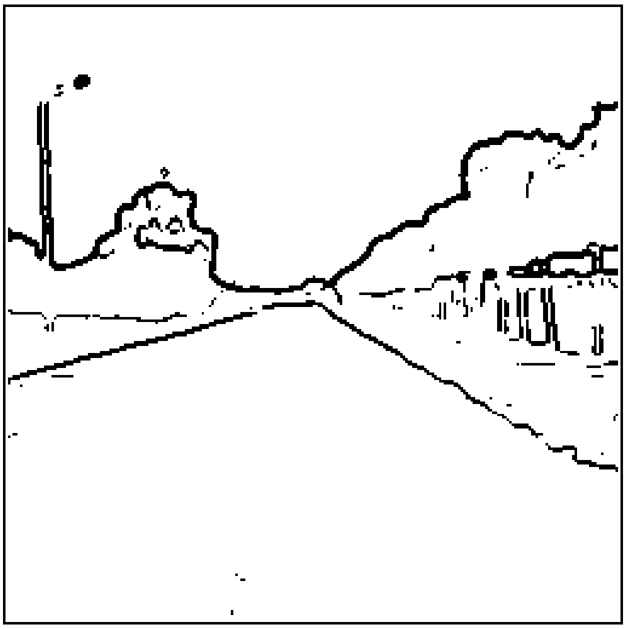

Figure 4: Thresholded Gradient Image