Journal of VLSI Signal Processing 19, 5–18 (1998)
© 1998 Kluwer Academic Publishers. Manufactured in The Netherlands.

Design and Implementation of the PAPRICA Parallel Architecture

A. BROGGI AND G. CONTE
Dipartimento di Ingegneria dell'Informazione, Università di Parma, Italy

F. GREGORETTI, C. SANSOÈ, R. PASSERONE AND L.M. REYNERI
Dipartimento di Elettronica, Politecnico di Torino, Italy

Received August 7, 1996; Revised March 13, 1997

Abstract. In this paper PAPRICA, a massively parallel coprocessor devoted to the analysis of bitmapped images is presented considering first the computational model, then the architecture and its implementation, and finally the performance analysis. The main goal of the project was to develop a subsystem to be attached to a standard workstation and to operate as a specialized processing module in dedicated systems. The computational model is strongly related to the concepts of mathematical morphology, and therefore the instruction set of the processing units implements basic morphological transformations. Moreover, the specific processor virtualization mechanism allows to handle and process multiresolution data sets. The actual implementation consists of a mesh of 256 single bit processing units operating in a SIMD style and is based on a set of custom VLSI circuits. The architecture comprises specific hardware extensions that significantly improved performances in real-time applications.

1. Introduction

A number of processor arrays with a bidimensional grid interconnection scheme and a SIMD processing paradigm have been conceived starting from the original ideas of Unger [1]. The availability and widespread use of VLSI technologies of the 80s and the natural mapping of a bidimensional interconnection scheme over the planar structure of a silicon chip led to an explosion of different proposals and implementations [2–6]. Most of the designs share common characteristics such as the bidimensional mesh interconnection scheme and a bit serial computation paradigm while other ones have either increased the complexity of the interconnection network, as the Connection Machine CM-2 [5], or widened the data path of the elementary PE, as the CLIP7 [7] or MasPar [8] systems.

The architecture of the PAPRICA system [9, 10], described in this paper and shown in Fig. 1(a), has the main characteristics of a conventional mesh-connected SIMD array but it has been specialized to the following objectives:

- to directly support a computational paradigm based on mathematical morphology [11] also on data structures larger than the physical size of the machine;
- to support hierarchical data structures;
- to provide a low cost experimental workbench for research in the fields of image processing, VLSI design automation, neural algorithms, etc.

The kernel of the system is a Processor Array (PA) of single-bit Processing Elements (PEs) with a bidimensional mesh interconnection to the 8 neighbors as shown in Fig. 1(b).

The instruction set of each PE can be described in terms of mathematical morphology operators [11], augmented with logical operations. All PEs operate according to a SIMD paradigm and a set of control flow statements are also provided. The virtualization of the operation of the physically limited array over a large bidimensional data structure stored in a linearly addressable memory is managed at the hardware level. An additional mechanism allows to map more complex hierarchical data structures on the same bidimensional processing grid.

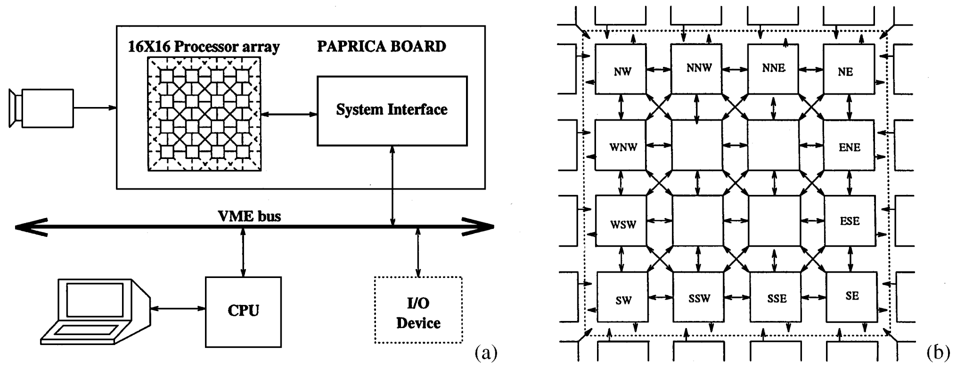


Figure 1. (a) The complete system; (b) interconnection topology of the SIMD array.

The PAPRICA system has been designed to operate as a coprocessor of a general purpose host workstation. Data and instructions are transferred to and from the host through a multi-port memory connected to a standard VME bus, whereas processing is concurrent with host activities. The current implementation is based on a number of custom integrated circuits, which have been designed with a particular emphasis on global project management issues such as cost and development time. For this reason conservative design and engineering choices have often been taken leading to performance limitations both in terms of processing speed and in the maximum number of PEs. The original target of the PAPRICA system was the acceleration of tasks related to the design and verification of IC layouts and masks [9] and this led to the implementation of ad hoc instructions which are seldom used in other applications. Nevertheless the generality of the mathematical morphology computational paradigm [11] allows an efficient use of the architecture in many tasks related to the processing of bidimensional data structures [12–14].

The paper is organized as follows. Section 2 introduces the PAPRICA computational paradigm and describe the external architecture and the instruction set. Section 3 describes the current hardware implementation, presents an overview of the major building blocks and explains the main choices which have been taken at the system level, while Section 4 presents a synthetic theoretical evaluation of the performances of the system and measurements on benchmarks and real applications. Finally, Section 5 presents the conclusions.

2. The Instruction Set of PAPRICA

The computational model which has inspired the design of PAPRICA derives from Mathematical Morphology [11, 15], a bitmap approach to the processing of discrete images derived from the set theory. The computational paradigm of PAPRICA is based on the concept of *matching operators*, which are derived from the *hit-miss transform* described by Serra [11], a rather general approach which includes the other morphological operators as special cases. In particluar, it is possible to synthesize more complex grey-level morphological operations as chains of binary ones.

Matching Operators are applied by each PE to its own neighborhood and the results are stored back in an internal register. The operation is based upon a *matching template*, a ternary mask of the same size of the neighborhood, whose values may be 0, 1 or $-$ (meaning *don't care*); a matching template can be sketched using the following notation:

$$\begin{array}{|c|c|c|}\hline \mathrm{x} & \mathrm{x} & \mathrm{x} \\ \hline \mathrm{x} & \mathrm{x} & \mathrm{x} \\ \hline \mathrm{x} & \mathrm{x} & \mathrm{x} \\ \hline \end{array},$$

where "x" $(0, 1, -)$ is the mask value. The center of the matrix corresponds to the PE itself. The application of a matching operator over a binary image corresponds to the bitwise comparison between the template and the values of the pixels of the image: the comparison leads to a *hit* if the template equals the PE's neighborhood completely (don't care pixels always match the image) giving 1 as a result; conversely, it leads to a *miss* if at least one pixel of the template does not match the corresponding pixel on the image, giving 0 as a result. The resulting image is given by the set of the individual comparison results.

As an example, the elementary matching

$$\begin{array}{|c|c|c|}\hline - & - & - \\ \hline 1 & - & - \\ \hline - & - & - \\ \hline\end{array}$$

produces an image which is displaced by one pixel in the east direction. Conventionally, the superscript c declares that the result be logically inverted, while the subsequent application of different matching templates is denoted by a simple comma-separated list. A numeric constant $K \in [2, 4, 8]$ placed in front of a template is a short form for the list of its possible rotations by $360/K$ degrees.

The instruction set of PAPRICA can be partitioned into three different classes, namely:

- *Morphological and logical statements* which are executed every PE of the array.
- *Mapping statements* to virtualize the PA on images larger than its size.
- *Control flow* statements.

2.1. Morphological and Logical Statements

Each PAPRICA instruction is the cascade of a *graphic operator* $G(\cdot)$ and a *logic operator* $*$. A PAPRICA instruction in assembly format is

$$L_D = G(L_S1) * L_S2\ [\%A], \qquad (1)$$

where $L_D, L_S1, L_S2 \in [0, \ldots, 63]$ are the identifiers of three 1-bit registers of each PE. This specific *structure-matching* operates on L_S1 and L_S2 and stores the result into L_D. The graphic operator G is one of the 16 composite matching operators listed in Table 1, while the logical operator '$*$' is one of the 8 unary, binary or ternary Boolean operators listed in Table 2. The optional switch $\%A$ introduces an additional OR operation between the result of the instruction and an internal accumulator. A sample sequence of vertical and horizontal expansion/erosion is shown in Fig. 2.

Table 1. List of PAPRICA graphic operators $G(L_1)$.

Name	Description	Structuring element
NOP (L_1)	No OPeration	$\begin{array}{ccc} - & - & - \\ - & 1 & - \\ - & - & - \end{array}$
INV (L_1)	INVersion	$\begin{array}{ccc} - & - & - \\ - & 0 & - \\ - & - & - \end{array}$
NMOV (L_1)	North MOVe	$\begin{array}{ccc} - & - & - \\ - & - & - \\ - & 1 & - \end{array}$
SMOV (L_1)	South MOVe	$\begin{array}{ccc} - & 1 & - \\ - & - & - \\ - & - & - \end{array}$
WMOV (L_1)	West MOVe	$\begin{array}{ccc} - & - & - \\ - & - & 1 \\ - & - & - \end{array}$
EMOV (L_1)	East MOVe	$\begin{array}{ccc} - & - & - \\ 1 & - & - \\ - & - & - \end{array}$
EXP (L_1)	EXPansion	$\begin{array}{ccc} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{array}^c$
VEXP (L_1)	Vertical EXPansion	$\begin{array}{ccc} - & 0 & - \\ - & 0 & - \\ - & 0 & - \end{array}^c$
HEXP (L_1)	Horizontal EXPansion	$\begin{array}{ccc} - & - & - \\ 0 & 0 & 0 \\ - & - & - \end{array}^c$
NEEXP (L_1)	NorthEast EXPansion	$\begin{array}{ccc} - & - & - \\ 0 & 0 & - \\ 0 & 0 & - \end{array}^c$
ERS (L_1)	ERoSion	$\begin{array}{ccc} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{array}$
VERS (L_1)	Vertical ERoSion	$\begin{array}{ccc} - & 1 & - \\ - & 1 & - \\ - & 1 & - \end{array}$
HERS (L_1)	Horizontal ERoSion	$\begin{array}{ccc} - & - & - \\ 1 & 1 & 1 \\ - & - & - \end{array}$
NEERS (L_1)	NorthEast ERoSion	$\begin{array}{ccc} - & 1 & 1 \\ - & 1 & 1 \\ - & - & - \end{array}$
BOR (L_1)	BORder	$8\begin{array}{ccc} - & - & - \\ - & 1 & 0 \\ - & - & - \end{array}$
LS2 (L_1)	LesS than 2	$\left(4\begin{array}{ccc} 1 & 1 & - \\ 1 & 1 & - \\ 1 & 1 & - \end{array}, 4\begin{array}{ccc} 1 & 1 & - \\ 1 & 1 & - \\ - & - & 0 \end{array}, \begin{array}{ccc} - & - & - \\ - & 0 & - \\ - & - & - \end{array}\right)^c$

Table 2. List of PAPRICA logic operators $*$.

Name	Description	Name	Description
(Omitted)	No operation	$\sim$	NOT
&	AND	&$\sim$	ANDNOT
|	OR	|$\sim$	ORNOT
$\wedge$	EXOR	$+$	PLUS (Arithm. Plus)

2.2. Mapping Statements

PAPRICA uses a specific construct $\mathcal{U}$ called *Update Block* which is a sequence of several simpler elementary statements taken from the *instruction set*. As shown in Fig. 3, operations on data structures larger than PA size are "virtualized" by splitting an image $\mathcal{I}$ in a set of adjacent *windows* $\mathcal{I}^k$ and by

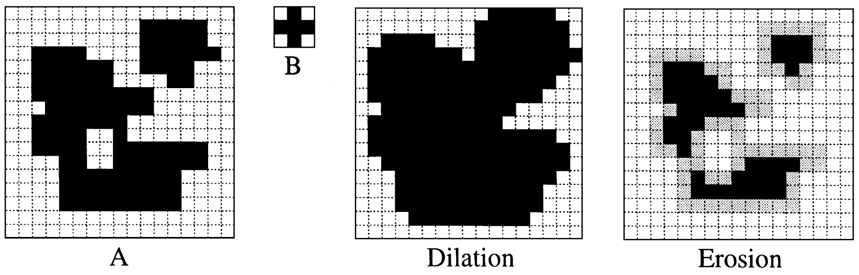


Figure 2. Examples of graphical operators.

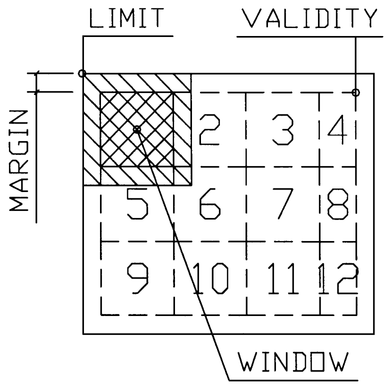


Figure 3. Vitualization of larger images on the PAPRICA array.

sequentially applying the same Update Block $\mathcal{U}$ to all the windows $\mathcal{I}^k$.

The result is independent of the PA size provided that none of the elementary morphological instructions of the Update Block modifies the input image $\mathcal{I}$. If the algorithm requires the input data be modified, then the program must be split into consecutive Update Blocks, storing intermediate results in temporary registers.

In the PAPRICA architecture the windowing mechanism is directly implemented at the hardware level by loading data from memory into the PA, executing an Update Block and storing back the results in the memory. The mapping between a logical pixel $\mathcal{P}$ and its physical address in the computer memory (e.g., word address and bit) is given by

$$\mathcal{O}(\mathcal{P}) = B + \mathcal{P}_x \cdot S_X + \mathcal{P}_y \cdot N_l \cdot S_Y, \qquad (2)$$

where B is the base address of the image in memory, N_l is the number of lines, and S_X and S_Y are two other programmable parameters, called *skip* factors. The mapping may be separately programmed for the load and store operations.

When a subwindow is loaded in the array and a graphical operation is executed, input data from neighbors PEs which are on the border of the array are undefined and this causes the evaluation of incorrect results in the regions between adjacent windows. The windowing mechanism compensates these errors by overlapping each loaded window with the previous one and by storing back into memory only the correct results. A few mapping instructions allow to program this mechanism. The `LIMIT` instruction sets the base address of the memory area (*Limit Area*) over which the input image is defined. The `MARGIN` instruction sets the *Margin* parameter, that is the maximum size of the error region produced by the sequence of instructions in a given Update Block. The `VALIDITY` statement defines the base address of the memory area (*Validity Area*) where the correct processed data are stored. In addition, the `SKIP.READ` and `SKIP.WRITE` statements allow to program the load and store image mapping.

2.3. *Flow Control Statements*

PAPRICA *flow control statements* are listed in Table 3. During processing, three *global flags* are computed for each Update Block, which are set according to the result of the last operator of that block. The *SET*, *RESET* and *NOCHANGE flags* are respectively set *iff* the result of the last operation of the block for *all* the pixels of the image is respectively one, zero or has not changed with respect to the value of the destination register before the operator has been computed. A common use of such flags is to detect the end of an algorithm.

A *PAPRICA Program* is the cascade of one or more Update Blocks $\mathcal{U}_i$, some *control instructions* and optionally one or more Mappings Statements, which concur to perform a desired function.

2.4. *Mapping Pyramidal Architectures: An Example*

Pyramidal architectures have shown several advantages in the field of image processing [16–18]. One major

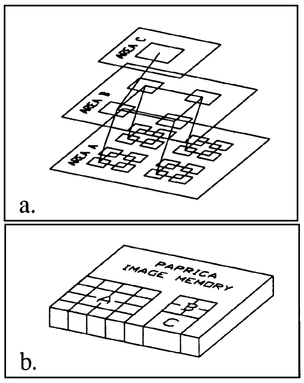
Table 3. List of PAPRICA flow control instructions.

Statement	Meaning
UPDATE	Separates two Update Blocks. All the other mapping and flow control instructions are also *implicit updates* which separate Update Blocks.
FOR ⟨*iter*⟩ instructions ENDFOR	Repeats *instructions* ⟨*iter*⟩ times.
REPEAT instructions UNTIL [NOT] ⟨*flag*⟩	Repeats *instructions* until ⟨*flag*⟩ is [NOT] set.
CALL ⟨*address*⟩	Calls a subroutine at ⟨*address*⟩.
RET	Returns from subroutine.
JUMP ⟨*address*⟩	Jumps Program Counter to ⟨*address*⟩.

drawback of such architectures is that they are not easily scaled beyond a certain size. This is because interconnections among processors have an intrinsic 3D structure (see Fig. 4(a)) which causes problems when mapped onto a 2D silicon surface.

PAPRICA solves the problem of interconnections because its hardware allows dynamic mapping of limit and validity areas (see Section 2.2). Using skip factors different from one, it is possible to map several pyramidal architectures such as the "4 children per parent" topology shown in Fig. 4(a). Figure 4(b) shows how this architecture can be mapped on the PAPRICA Image Memory.

Figure 4(c) lists the PAPRICA code necessary to emulate such a pyramidal structure. At the beginning, the program elaborates the lowest level of the pyramid,



```
                      ; ==== 1st LAYER ====
LIMIT <AREA A>        ; takes input data from area A
VALIDITY <AREA A>     ; writes results into area A
SKIP.READ 1 1         ; one input and
SKIP.WRITE 1 1        ; one output per PE
...
...                   ; 1st LAYER ROUTINE
...
                      ; ==== 2nd LAYER ====
LIMIT <AREA A>        ; takes input data from area A
VALIDITY <AREA B>     ; writes results into area B
SKIP.READ NPx NPy     ; input data organized
                      ; as NPx NPy pixels,
SKIP.WRITE 1 1        ; just one output per each PE
                      ; (can be more than one)
...
...                   ; 2nd LAYER ROUTINE
...
                      ; ==== 3rd LAYER ====
LIMIT <AREA B>        ; takes input data from area B
VALIDITY <AREA C>     ; writes results into area C
SKIP.READ NPx NPy     ; input data organized
                      ; as NPx NPy pixels,
SKIP.WRITE 1 1        ; just one output per PE
                      ; (can be more than one)
...
...                   ; 3rd LAYER ROUTINE
...
                      ; == OTHER LAYERS ==
LIMIT ...             ; same as above
...                   ;
```

c.

Figure 4. Mapping of a pyramidal architecture on PAPRICA.

e.g., performing initial filtering of input data. This is done by having *Limit* and *Validity* Areas coincident with input data. Then, *Validity Area* base address and *Skip* parameters are changed respectively to point to area *B* and to obtain the amount of decimation necessary (e.g., 2 × 2), so that the output data of the lowest level are used as input for the computations required by the second-last one.

This process is iterated, each time executing different `LIMIT`, `VALIDITY` and `SKIP` instructions, up to the root level. It has to be noted that processing on different levels can be completely different in nature, and that merely changing the above-mentioned parameters it is possible to change the type of logic connections mapped by the system. In any case, the overhead introduced to emulate pyramidal structures is very small because the host processor does not have to reorganize or move data in memory to process the different levels.

3. System Implementation

As already mentioned the PAPRICA system has been designed as a specialized coprocessor for a general purpose host workstation and, as a whole, it is composed of the following main functional parts: the Image (IM) and Program (PM) Memories, the Processor Array (PA), the Control Unit (CU), the Camera Interface (CI) and the Image Remapper (IR). The logical relations between these units are shown in Fig. 5. Figure 6 shows the photograph of the complete system.

Using the system bus the host transfers coprocessor code and data into the Program and Image Memories

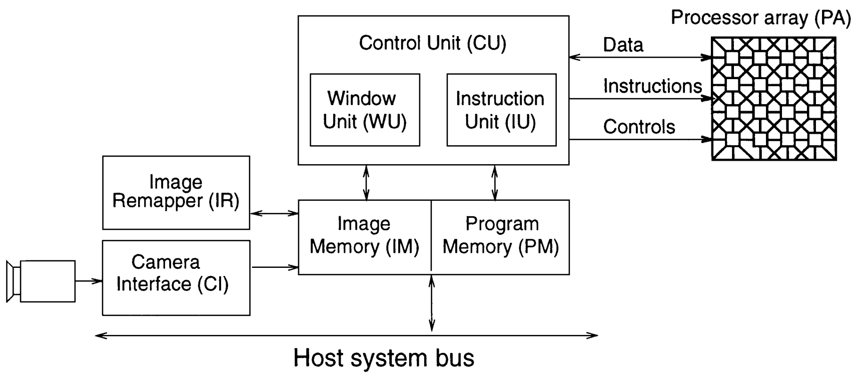


Figure 5. Block diagram of the PAPRICA system.

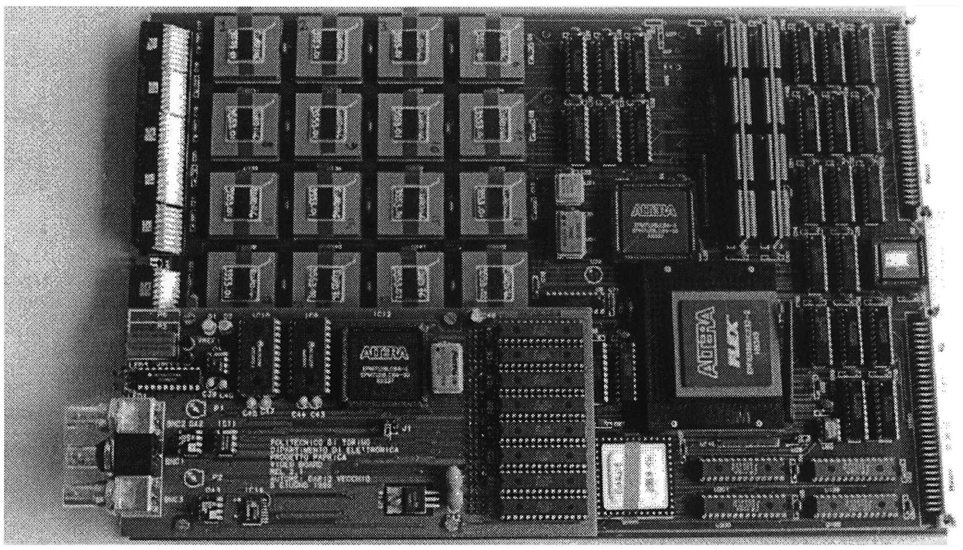


Figure 6. Photograph of the complete PAPRICA board.

(up to 8 MB of fast static RAM). Grey scale image data can also be acquired directly from a video camera connected to the PAPRICA board under the host supervision. The CU fetches instructions from the Program Memory, directly executes control flow and mapping statements and broadcasts the elementary instructions to the PA. A second task of the CU is the management of the data transfers between IM and PA in order to implement the windowing mechanism. The contents of the IM can also be reorganized using the Image Remapper: this unit can create a new image from an existing one reorganizing the pixels of the source image according to the contents of a look-up table stored in another portion of the IM. In the current implementation, the PAPRICA system is composed of a single 6U $\times$ 340 mm VME board; the video camera interface is provided through a small piggy-back board hosting an 8-bit video ADC and a controller.

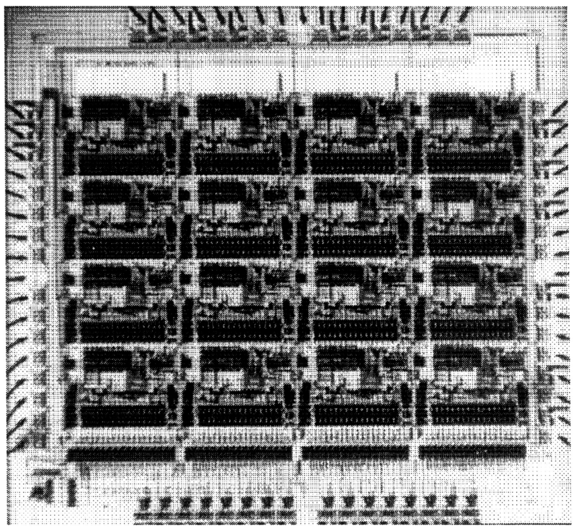


Figure 7. Microphotograph of the 4 $\times$ 4 PE chip.

3.1. *The Processor Array*

The PA is presently a 16 $\times$ 16 square matrix of 1-bit PEs with full 8-neighbors connectivity. The PA as a whole may operate in two different modes:

- **Memory Mode.** It behaves like a standard RAM device composed of the collection of the individual PE registers for a total of 16 Kbits organized as 1K $\times$ 16-bit words.
- **Processor Mode.** All PEs execute the same *array* instruction fetched by the CU.

The main PE components are the 64-bit *Register File* and the *Execution Unit*. The Register File has one read/write and one read-only 1-bit ports allowing the fetch of two operand at the same time in Program Mode and one 16-bit read/write port for accesses in Memory Mode. The Execution Unit is composed of two separate blocks, named GOP and LOP, one for the evaluation of graphic operators and another for logic operators, both implemented as two-level combinatorial logic by the use of semistatic PLAs. A common multiplexed *Memory/Instruction Bus* is used to broadcast the instruction to all PEs and to transfer data to/from the external memory.

A VLSI implementation of the PA is a natural choice and the number of the PE integrated on a single chip strongly depends on the available technology, therefore this number has been chosen as a compromise among feasible die area, cost and yield functions, design complexity and power supply requirements. Although with the technology available (1.5 μm CMOS) when the design was started an array of 8 $\times$ 8 PEs would have fit into a 1 cm^2 chip, it has been preferred, as a conservative choice to obtain a higher yield and reduce design complexity, to integrate into a single chip only an array of 4 $\times$ 4 PEs.

The chip has been designed using a full-custom methodology and fabricated with a 1.5 μm CMOS technology, with a total complexity of approximately 35,000 transistors on an area of 45 mm^2 and an average power consumption is approximately 0.3 W. A microphotograph of the chip is shown in Fig. 7.

3.2. *The Control Unit*

The three main tasks of the Control Unit are:

- the implementation of the *windowing* mechanism described in Section 2.2;
- the instruction fetch, execution of the program control flow statements and transfer to the PA of the morphological and logical statements;
- the handling of the accesses to image memory from VME host, PA, and camera interface.

Being rather independent both from the functional and the temporal point of view, they have been assigned to three different blocks:

- **Window Unit (WU).** This unit sequentially generates the image memory physical addresses corresponding to a given subwindow and the PE register addresses to/from which data are to be transferred according to the programmed mapping. At the end of the transfer of each subwindow the control is passed to the Instruction Unit for the execution of the the instructions which comprise the *Update Block*. The complexity of the WU in terms of elementary gates is comparable to that of the PA chip. It has been implemented as a standard cell circuit using the same technology as the PEs giving a 30 mm^2 chip with 110 I/O pins and a complexity of approximately 6000 gates.
- **Instruction Unit (IU).** This unit is a rather conventional sequencer and its main components are: a 32-bit *Instruction Register*, a 24-bit *Program Counter*, an 8-bit *Stack Pointer*, a 32-bit wide *Stack* plus the hardware necessary to directly execute flow-control instructions (`FOR`, `REPEAT`, `CALL`). It has been implemented using a single EP81188 Altera 244 pin FPGA. This choice was preferred to the use of a standard microcontroller because of the system speed requirements: only a high-performance DSP could have enough processing power to run the IU tasks at equivalent speed but the resulting board complexity would be considerably higher than that of the dedicated HW solution.
- **Memory Controller.** This unit implements all the functionality necessary to control memory accesses from the various sources (VME host, WU, CU, CI); this unit has also been implemented using a FPGA device.

3.3. The Camera Interface

The video camera interface provides a way to acquire images with the lowest possible delay. It is a 256 grey levels frame grabber interacting directly with the IM, programmed from the host through dedicated control and status registers. The frame grabber can be programmed to acquire a full-frame (2-fields) 512 × 512 pixels image in 40 ms or a faster single-field 512 × 256 pixels image in 20 ms. The acquisition can be from a single camera or from an externally synchronized stereo camera pair. The advantage of using this built-in frame grabber is the improvement of real-time algorithms execution speed because the image is made immediately available to the PA so that no memory transfers have to be done on the VME bus.

3.4. The Image Remapper

The Image Remapper (IR) reorganizes the contents of an image by relocating each pixel in accordance to the contents of a mapping look-up table, previously stored by the host in the IM. The remapping speed is of about 200 ns per pixel. The IR implements a global communications network, which is seldom available in mesh connected SIMD architectures. It sequentially reads all the pixels of the images and writes each of them to an address derived from the mapping look-up tabel. It has been implemented as part of the same FPGA used for the IU.

4. Performance Analysis

The processing speed $S_{\rm pr}$ is a function of parameters that depend on system technology and architecture (i.e., number of Processing Elements Q^2, memory cycle time T_M, and processor cycle time T_C), on the particular computational task (program length L and sum G of the margin values of each Update Block), and on the number $n_{\rm upd}$ of Update Blocks in which the application is partitioned.

The processing speed $S_{\rm pr}$ (in pixel/s) is defined as the number of processed pixels over the total time needed for processing and has been determined as [10].

$$S_{\rm pr} \simeq \frac{\left(Q - \frac{2G}{n_{\rm upd}}\right)^2}{2Q^2 T_M n_{\rm upd} + L T_C}. \tag{3}$$

$n_{\rm upd}$ is the only parameter that can be tuned with the aim to maximize $S_{\rm pr}$.

The diagram presented in Fig. 8 ($S_{\rm pr}$ vs. $n_{\rm upd}$) exhibits a peak in the processing speed for a given value of $n_{\rm upd}$. In fact a small $n_{\rm upd}$ corresponds to a large number of graphical operators applied on the same image subwindow, thus sensibly reducing the validity area, and requiring a large overlapping in the loading of successive subwindows into the PA. The low processing speed is due to the fact that the same data flow more than once from the IM to the PA and vice versa. However, a large value of $n_{\rm upd}$ corresponds to partitioning the application in a large number of update blocks, thus requiring the virtualization to be performed with multiple successive transfers of the image between the IM and the PA. The optimal value of $n_{\rm upd}$ is then given by

$$[n_{\rm upd}]_{\rm opt} = 6\,\frac{G}{Q}. \tag{4}$$

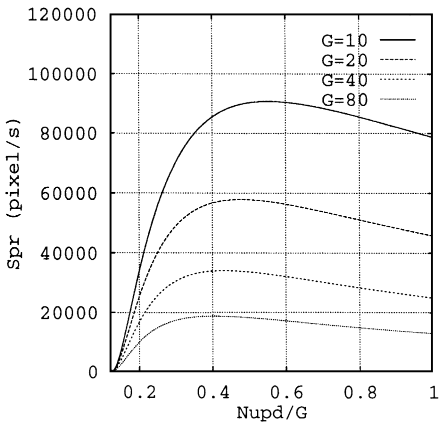


Figure 8. Processing speed S_{pr} vs. n_{upd}/G considering $Q = 16$, $T_M = T_C = 350$ ns, and $L = 2000$.

Thus, the maximum processing speed becomes

$$[S_{pr}]_{max} \simeq \frac{4Q^2}{100QGT_M + 9LT_C}, \quad (5)$$

which, for L small compared to G, becomes

$$[S_{pr}]_{max} \simeq \frac{Q}{25GT_M}, \quad (6)$$

showing that in the general case, when a large number of graphic operators is used, the speedup is proportional to Q, that is the square root of the number of PEs. This theoretical result is supported by experimental data as shown in the next section.

4.1. A Case Study

As an example, a SIMD implementation [19] of a binary thinning [20] algorithm is presented: it is composed of 8 matches with 3×3 patterns. This corresponds to a program with $G = 8$. The PAPRICA implementation of this filter is made of about 50 instructions ($L \simeq 50$).

As shown in the previous sections, a specific application must be split in a number of different Update Blocks depending on the PA dimension, according to Eq. (4). In order to achieve the best performances, each Update Block must have a margin value as close as possible to the optimal value of $\frac{Q}{6}$.

Table 4 shows the optimal margin value corresponding to different PA sizes, together with the best partitioning of one thinning iteration (expressed as the list of the margin values of each Update Block). Moreover, for the processing of a 256×256 binary image, Table 4 compares also the value of the execution time evaluated theoretically from Eq. (3), the value determined through the use of a software simulator, and finally the value measured on the real hardware system (only for $Q = 16$).

Table 4. Performances of one thinning iteration ($G = 8$) on architectures with different PA sizes.

	Optimal	Margin	T_{1_iter} (ms)		
Q^2	margin	values	Theoretical	Simulated	Measured
4^2	1	1,1,1,1,1,1,1,1	1744	1220	—
8^2	1	1,1,1,1,1,1,1,1	683	540.5	—
16^2	3	3,3,2	326	227.8	$232 \pm 2\%$
32^2	5	4,4	165	124.2	—
64^2	10	8	82	56.43	—
128^2	21	8	60	49.35	—
256^2	42	8	52	43.12	—

Note that from $Q = 4 \div 64$, Update Blocks in which the application has been partitioned has a margin value comparable to the optimal one. In these cases the computational time is proportional to $\frac{1}{Q}$, according to Eq. (6). For $Q > 64$ it is not possible to obtain the optimal partitioning since the optimal margin is larger than G. Optimal partitioning can be achieved for sufficiently high values of G: for example, an application consisting of 10 iterations of a thinning filter ($G = 80$) can be nearly optimally partitioned even in case of an architecture composed of a 256×256 PA.

The values presented in Table 4 refer to a non-optimized version of the thinning filter. In [21] an optimization technique has been presented: it is based on the consideration that the data bus, which links the PM and the PA, has a 16 bit parallelism, while the data flowing through it are binary, producing a data bus efficiency of about 6% only. The improvement of the data bus efficiency is the goal of such a technique, and it is achieved by a data packing procedure. The data packing process reads from the image memory a $N \times N \times 1$ image and writes a $\frac{N}{4} \times \frac{N}{4} \times 16$ image. Besides incrementing the bus efficiency to 100%, it decreases the image linear dimensions from N to $\frac{N}{4}$, and it increases the pixels neighborhood accessed by an individual read instruction.

This data packing technique allows to achieve a speed-up which is proportional to the number of iterations of the thinning filter. The performances (measured

Table 5. Performance of a thinning filter working on a 256 × 256 plain and a 64 × 64 packed version of the same image.

No. of iterations	Execution time (ms) Normal	Execution time (ms) Optimized	Improvement (%)
1	228	85	170
2	475	98	380
4	712	111	540
8	950	124	660

on the real hardware system) obtained through the use of this optimizing technique are presented in Table 5.

4.2. *The Abingdon Cross Benchmark*

The main target of the Abingdon Cross benchmark [22] is to compare the performance of computer architectures devoted to image processing. This benchmark has been defined to be implemented on any computer architecture, regardless of its computational paradigm. The benchmark target is the determination of the computational time required by the filtering of 128 × 128 pixels, 8 bit/pixel, test image shown in Fig. 9(a).

There are no constraints at all on the implementation of the Abingdon Cross benchmark: the best performances are achieved when the algorithm matches the computer architecture. Moreover, the design of the algorithm can take advantage of the knowledge of the key parameters used in the generation of the test image.

The quality factor QF (the computer architecture performance index) is defined as the ratio between the test image size (N) and time required to filter it (T).

For commercial systems the price-performance factor is also defined. Since PAPRICA system is a noncommercial prototype, only the quality factor will be derived.

4.2.1. Benchmarking PAPRICA System. Since PAPRICA system is composed of single bit PEs, the processing of binary images is preferred with respect to the handling of data structures with higher word parallelism. Moreover, due to the specific instruction set implemented on PAPRICA, high performance levels are obtained when the algorithm is based on morphological operations. Thus the first step of the filtering is (a) a thresholding operation, which produces a binary image. Then (b) a number of morphological operations are performed. Finally (c) a thinning filter [20] is applied on the resulting image. In the implementation of points (b) and (c) the data-packing optimizing technique mentioned in Section 4.1 has been used.

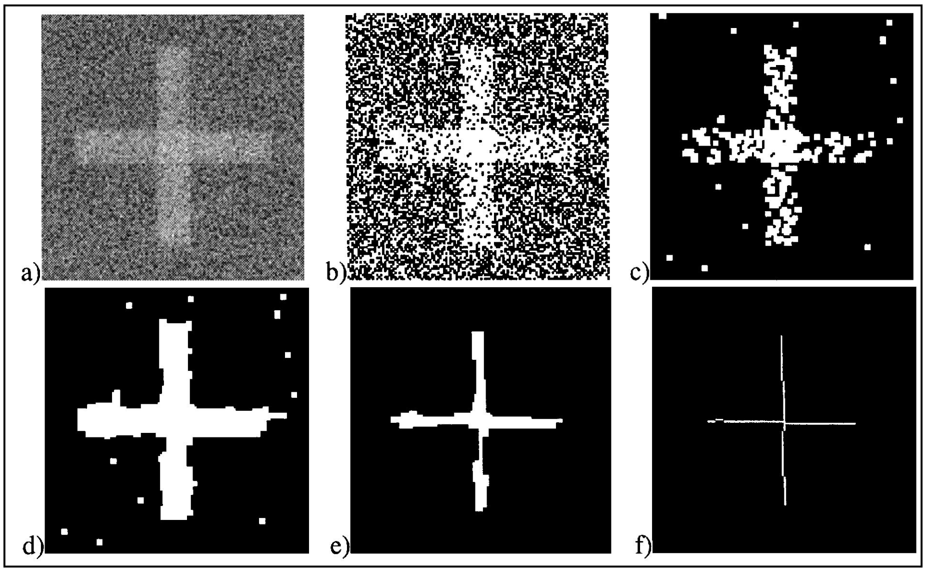


Figure 9. The Abingdon Cross benchmark on a 128 × 128 image: (a) input image; (b) thresholded image; (c) after one morphological opening with the elementary anisotropic structuring element (3 × 3); (d) after one morphological closing with a 7 × 7 anisotropic structuring element; (e) after four morphological erosions with the elementary anisotropic structuring element (3 × 3); (f) after three thinning iterations.

Table 6. The quality factor for some non-commercial machines.

Machine	Quality factor
CLIP4 (96 × 96)	7.3×10^3
Cyto III	1.0×10^2
DIP	4.7×10
PHP	7.1
POP II	2.9
VAP	1.8×10
WARP	5.0×10^2
PAPRICA	9.8×10^2

The hardware prototype runs the Abingdon Cross Benchmark on a 128×128 pixel image in about 130 ms. The quality factor is $\mathrm{QF} = 9.8 \times 10^2$. Table 6 reports the results shown in [22] for the non-commercial machines only, together with the result for the PAPRICA system, while the complete sequence of intermediate results of the Abingdon Cross Benchmark is depicted in Fig. 9.

4.3. *A Sample Application*

The PAPRICA system has been extensively used and tested for different applications, ranging from vision-based automotive applications [12, 13, 23, 24] to the analysis of images of integrated circuits acquired from a SEM [14], from the lossy compression of images [25] to the emulation of cellular neural networks.

Within the Eureka PROMETHEUS project a few applications have been developed, aimed to the identification of the road markings, to the computation of the optical flow field and the time-to-impact. In the road detection algorithm each frame is reorganized, removing the perspective effect in order to generate a new image with a uniform information distribution among its pixels. This step simplifies the detection of road markings which is performed by a morphological algorithm. A single frame is loaded in about 20 ms and processed in less than 100 ms, thus allowing the processing of about 8 frames/s. Figure 10 presents the original and processed images.

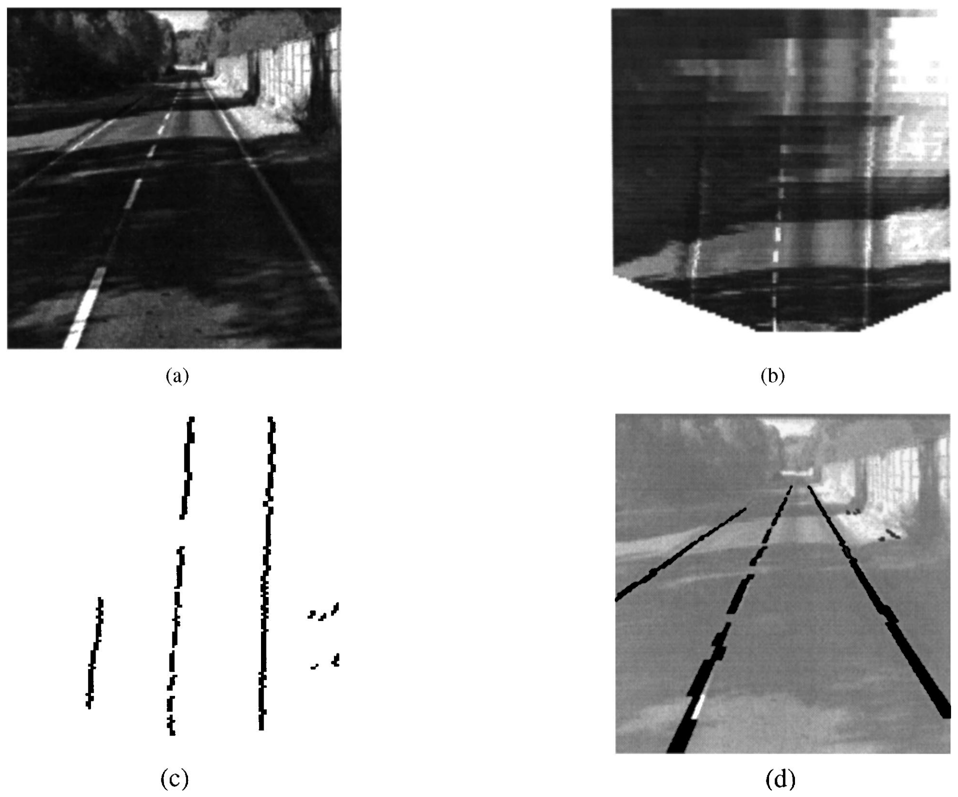


Figure 10. (a) Original image; (b) remapped image; (c) binarized image; (d) result superimposed onto the original image.

5. Conclusions

A massively parallel architecture supporting an image processing paradigm based on mathematical morphology has been described starting from the underlying computational model up to the physical implementation.

The performance limitations of the machine are in part related to the architecture and in part derive from implementation issues. As shown inSection 4 if the size of the array is far smaller than the size of the images processed, then the speed-up is proportional only to the square root of the number of processing elements. The choice of well-assessed technology has limited the number of processing elements per chip and conservative engineering solutions such as the use of programmable arrays, and some timing bottlenecks in the chip design have led to an implementation which is slower by a factor of 4 than what would have been achievable with the same architectural solution and an optimal design. The effectiveness of the system could be improved both by a redesign of the chip set with minor modifications and by exploiting the possibilities offered by modern interconnection technologies such as Multi Chip Modules (MCM). A feasibility study [26] has shown that using a 1 μm technology it is possible to integrate in a single die an array of 8×8 PEs with a cycle time of 100 ns in Program and Memory modes and that with MCM techniques a complete array of 8×8 chips (4096 PEs) would fit in a single package with a size of 8×8 cm. For most of the developed algorithms [12, 13, 23, 25] the speed-up over the current implementation would be more than two orders of magnitude.

The implementation of the PAPRICA prototype required to face all the issues related to the design of systems for real-time processing and has proved extremely useful under many aspects. The system, now installed in two academic institutes and on board of an experimental vehicle, is a workbench for the use of cellular and morphological algorithms and for the analysis of the potentials and the limitations of 2D SIMD arrays. The experience gained in the project is at the base of the design of a new morphological coprocessor [27] which is now nearly completed.

We strongly believe that the effort of developing an architecture down to the hardware implementation and to use it in real applications allows to discover implementation critical points such as cost, power, size, memory and I/O timing. In fact, considering the performance at the system level, the influence of implementation constraints may significantly degrade the performance figures which can be derived by a theoretical evaluation of a given architecture.

Acknowledgments

This work was partially supported by CNR (Italian National Research Council) under the frame of Progetto Finalizzato Trasporti.

References

1. S. Unger, “A computer oriented toward spatial problems,” *Proceedings IRE*, Vol. 46, pp. 1744–1750, 1958.
2. T. Fountain, *Processor Arrays: Architectures and applications*, Academic-Press, London, 1987.
3. I.N. Robinson and W.R. Moore, “A parallel processor array architecture and its implementation in silicon,” *Proceedings of IEEE Custom Integrated Circuits Conference*, Rochester, New York, pp. 41–45, 1982.
4. *Geometric Arithmetic Parallel Processor*, NCR Corporation, Dayton, Ohio, 1984.
5. *Connection Machine CM-200 Series—Technical Summary*, Thinking Machines Corporation, Cambridge, MA, 1991.
6. T. Sudo, T. Nakashima, M. Aoki, and T. Kondo, “An LSI adaptive array processor,” *Proceedings IEEE International Solid-State Circuits Conference*, pp. 122, 123, 307, 1982.
7. T. Fountain and K. Matthews, “The CLIP 7A image processor,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 10, No. 3, pp. 310–319, May 1988.
8. *MP-1 Family Data-Parallel Computers*, MasPar Computer Corporation, Sunnyvale, California, 1990.
9. G. Conte, F. Gregoretti, L. Reyneri, and C. Sansoé, “PAPRICA: A parallel architecture for VLSI CAD,” in *CAD accelerators*, A.P. Ambler, P. Agrawal, and W.R. Moore (Eds.), North Holland, Amsterdam, pp. 177–189, 1991.
10. A. Broggi, G. Conte, F. Gregoretti, C. Sansoè, and L.M. Reyneri, “The PAPRICA massively parallel processor,” *Proceedings MPCS—IEEE International Conference on Massively Parallel Computing Systems*, IEEE Computer Society—EuroMicro, Ischia, Italy, pp. 16–30, May 1994.
11. J. Serra, *Image Analysis and Mathematical Morphology*, Academic Press, London, 1982.
12. M. Bertozzi and A. Broggi, “GOLD: A parallel real-time stereo vision system for generic obstacle and lane detection,” *IEEE Transactions on Image Processing*, 1997, Vol. 7, pp. 62–81, 1998.
13. A. Broggi and S. Bertè, “Vision-based road detection in automotive systems: A real-time expectation-driven approach,” *Journal of Artificial Intelligence Research*, Vol. 3, pp. 325–348, Dec. 1995.
14. F. Gregoretti and C. Passerone, “Using a massively parallel architecture for integrated circuits testing,” *Proceedings 3rd EuroMicro Workshop on Parallel and Distributed Processing*, IEEE Computer Society—EuroMicro, Sanremo, Italy, pp. 332–338, Jan. 1995.
15. R.M. Haralick, S.R. Sternberg, and X. Zhuang, “Image analysis using mathematical morphology,” *IEEE Transaction on Pattern*

Analysis and Machine Intelligence, Vol. 9, No. 4, pp. 532–550, 1987.
16. V. Cantoni and M. Ferretti, *Pyramidal Architectures for Computer Vision*, Plenum Press, London, 1993.
17. A. Rosenfeld, *Multiresolution Image Processing and Analysis*, Springer Verlag, Berlin, 1984.
18. S.L. Tanimoto, T.J. Ligocki, and R. Ling, "A prototype pyramid machine for hierarchical cellular logic," *Parallel Computer Vision*, Academic Press, London, 1987.
19. C. Arcelli, L. Cordella, and S. Levialdi, "Parallel thinning of binary pictures," *Electronics Letters*, Vol. 11, pp. 148–149, July 1975.
20. A. Rosenfeld, "A characterization of parallel thinning algorithms," *Information Control*, Vol. 29, pp. 286–291, 1975.
21. A. Broggi, "Word Parallelism vs. Spatial Parallelism: A performance optimization technique on the PAPRICA system," *Proceedings 3rd EuroMicro Workshop on Parallel and Distributed Processing*, IEEE Computer Society—EuroMicro, Sanremo, Italy, pp. 236–243, Jan. 1995.
22. K. Preston, "The abingdon cross benchmark survey," *Computer*, pp. 9–18, July 1989.
23. G. Adorni, A. Broggi, G. Conte, and V. D'Andrea. "Real-time image processing for automotive applications," in *Real-Time Imaging: Theory, Techniques, and Applications*, P.A. Laplante and A.D. Stoyenko (Eds.), IEEE Press, pp. 161–194, 1996.
24. M. Bertozzi and A. Broggi, "Real-time lane and obstacle detection on the GOLD system," in *Proceedings IEEE Intelligent Vehicles'96*, I. Masaky (Ed.), Tokyo, Japan, IEEE Computer Society, pp. 213–218, Sept. 1996.
25. A. Broggi, "A novel approach to lossy real-time image compression: Hierarchical data reorganization on a low-cost massively parallel system," *Real-Time Imaging Journal*, Vol. 1, No. 5, pp. 339–354, Nov. 1995.
26. F. Gregoretti, L.M. Reyneri, C. Sansoè, A. Broggi, and G. Conte, "The PAPRICA SIMD array: Critical reviews and perspectives," in *Proceedings ASAP'93—IEEE Computer Society International Conference on Application Specific Array Processors*, L. Dadda and B. Wah (Eds.), IEEE Computer Society—EuroMicro, Venezia, Italy, pp. 309–320, Oct. 1993.
27. A. Broggi, G. Conte, G. Burzio, L. Lavagno, F. Gregoretti, C. Sansoè, and L.M. Reyneri, "PAPRICA-3: A real-time morphological image processor," *Proceedings ICIP—First IEEE International Conference on Image Processing*, IEEE Computer Society, Austin, TX, pp. 654–658, Nov. 1994.

Alberto Broggi is Researcher at the Dipartimento di Ingegneria dell'Informazione of the University of Parma since 1994. He received both the Dr. Eng. (Master) degree in Electronic Engineering (1990) and the Ph.D. degree in Information Technology (1994) from the same University. His research interests include real-time computer vision algorithms for the navigation of unmanned vehicles, and the development of low-cost computer systems to be used on autonomous robots. He is the author of more than 70 papers published in referred journals, international conference proceedings and books. He is the Editor of the Newsletter and member of the Executive Committee of the IEEE Technical Committee on Complexity in Computing, and member of the Editorial Board of Real-Time Imaging Journal. He has guest edited a number of special-issues of international journals on Machine Vision (IEEE Expert, Real-Time Imaging, Image and Vision Computing, Engineering Applications of Artificial Intelligence), and has been invited as a track leader, session organizer and tutorial chairman in several IEEE-sponsored international conferences, and served on the Program Committee of many major conferences.
`broggi@CE.UniPR.IT`
`http://WWW.CE.UniPR.IT/people/broggi`

Gianni Conte is professor of Computer Science at the School of Engineering of the University of Parma, where he teaches computer architectures and parallel processing. Before joining the University of Parma he has held the same position at the University of Salerno, and the positions of research associate and associate professor at the Politecnico of Torino where he received his Electronic Engineering in 1970. He is author of over 100 technical papers in refereed journals and conferences as well as of three books on parallel computer architectures and performance modeling. His research interests include parallel and distributed processing, performance evaluation of distributed systems, Petri net modeling, and VLSI system architectures. He is a member of IEEE.

Francesco Gregoretti was born in Torino, Italy and received in 1975 the Dr. Eng. degree in Electronic Engineering from Politecnico di

Torino where he started his academic activity as assistant professor in 1978 and where now is associate professor of Microelectronics. From 1976 to 1977 he was at the Laboratoire de Calculatrices Digitales at the Swiss Federal Institute of Technology in Lausanne (Switzerland) and from 1983 to 1984 visiting scientist at the Department of Computer Science of Carnegie Mellon University, Pittsburgh (USA). His main research interest are in digital electronics and in particular in MOS VLSI circuits, in massively parallel multimicroprocessor systems for VLSI CAD tools, in low level image processing architectures and in hardware/software codesign tools. Prof. Gregoretti is a member of IEEE and has authored and coauthored more than 90 papers and three books.

Claudio Sansoè is an assistant professor at the Politecnico di Torino, Italy. He teaches applied electronics and carries out research on the design of VLSI implementations of high-performance parallel SIMD architectures and on hardware-software codesign of embedded electronic systems. He holds a degree in electronic engineering from the Politecnico di Torino and has published more than 40 papers.

Roberto Passerone was born in Torino, Italy, in 1969. He attended the Politecnico di Torino, Italy, from 1988 to 1994 and received the Laurea in Ingegneria Elettronica. Since 1995 he has been with the Department of Electronics, Politecnico di Torino, where he is currently a Ph.D. student. His research interests include digital VLSI design, parallel computers and high performance computer architectures.

Leonardo M. Reyneri is an associate professor at the Politecnico di Torino, Italy. He teaches applied electronics and neural networks, and carries out research on applications of neural networks to intelligent control and pattern recognition. He is also involved in the design of VLSI implementations of high-performance neural networks and parallel SIMD architectures. His fields of personal research interest include the design of low-power mixed analog/digital integrated circuits for robotic, and the development of dedicated architectures for neural networks and massively parallel systems. Reyneri holds a Ph.D. in electronic engineering from the Politecnico di Torino. He is a member of IEEE, and has published more than 70 papers and holds five patents.